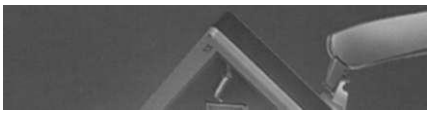
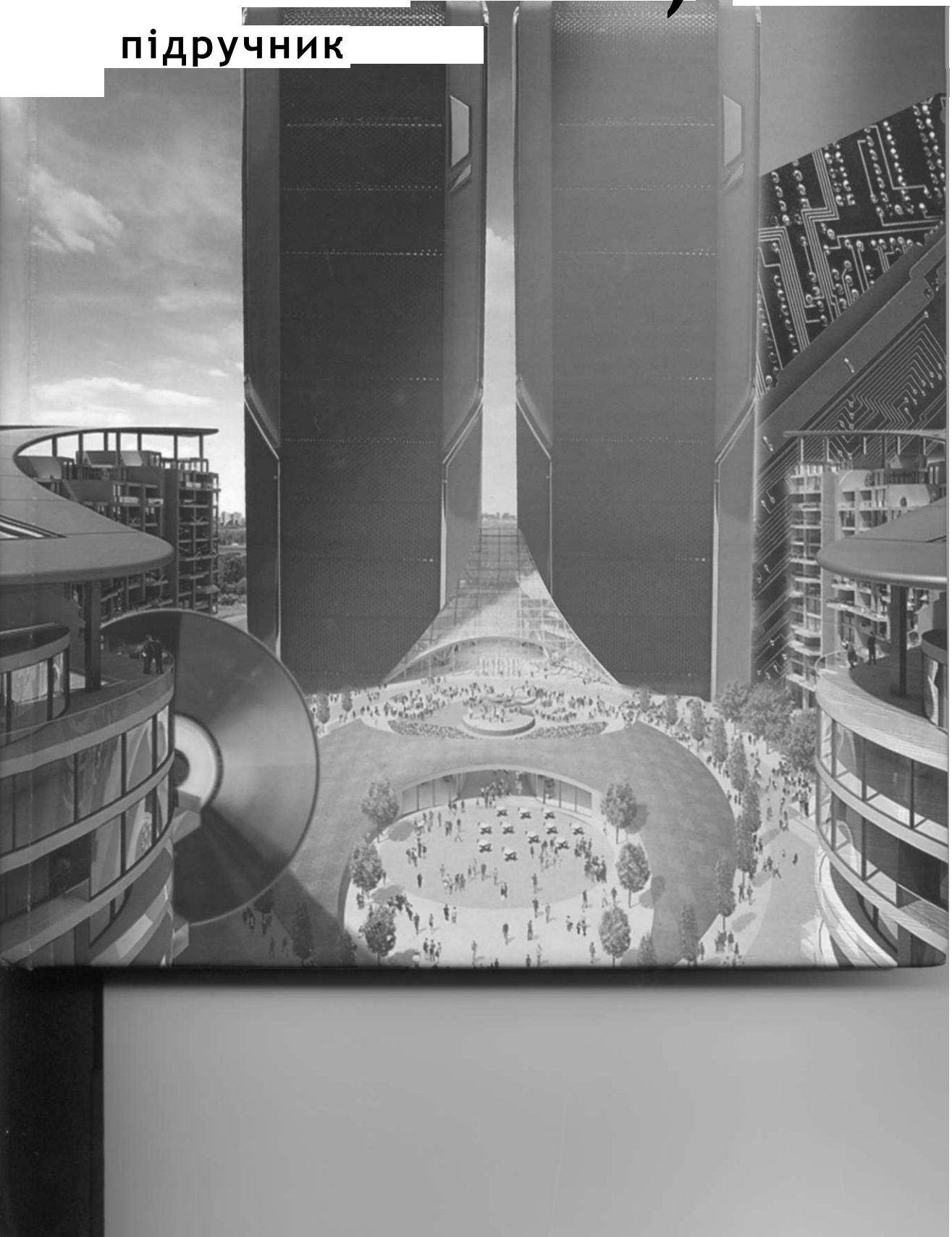
Анатолій МЕЛЬНИК



АРХІТЕКТУРА КОМП'ЮТЕР)А

підручник



ЗМІСТ

Цередмо&си

[ШІ|ШІ|ВІ|еІіві!**ВІ**|вІ|в)|ві[тІ|вІ|ві|вІ|ві|шІ|вІівШів)[шШ

Комп'ютерні технології все стрімкішими темпами входять в усі сфери життя суспільства, що викликає потребу підготовки фахівців за відповідними напрямами, в першу чергу за напрямами "Комп'ютерна інженерія", "Комп'ютерні науки", "Програмна інженерія", "Прикладна математика", "Захист інформації в комп'ютерних системах та мережах", "Комп'ютеризовані системи, автоматика і управління". Однією з базових дисциплін підготовки бакалаврів за названими напрямами є дисципліна "Архітектура комп'ютера". Цей підручник повною мірою відповідає змісту міністерських програм за вищеназваними напрямами.

Запропонований підручник орієнтований на студентів вищих навчальних закладів, викладачів та аспірантів комп'ютерних напрямів підготовки та покликаний прищепити талановитій молоді розуміння основних принципів побудови та організації сучасних комп'ютерів. Він також може бути корисним для фахівців в області комп'ютерних та інформаційних технологій.

Підручних охоплює весь комплекс питань, пов'язаних з теорією, принципами та методами побудови й організації функціонування комп'ютерів та складається з дванадцяти розділів.

В першому розділі розглянуто історичні аспекти розвитку комп'ютерів, подано основні поняття, функції та основні функціональні вузли комп'ютера, їх взаємозв'язок, а також загальну організацію роботи комп'ютера. Розглянуто характерні риси та сфери застосування різних типів комп'ютерів. Обгрунтовано місце предмета даної книги серед суміжних дисциплін та порядок його розгляду.

В наступному, другому, розділі розглянуто основні елементи архітектури комп'ютера. Показано як кодуються та виконуються команди в комп'ютері. Проведена класифікація команд відповідно до ініційованих ними типів операцій та детально розглянуті команди обробки даних, переміщення даних, передачі керування, введення-виведення. Введено поняття конвеєрного виконання команд - одного з видів паралелізму на рівні команди. Розглянуто три типи архітектур комп'ютера за типом адресованої пам'яті: стекова, акумуляторна, та на основі регістрів загального користування, їх переваги і недоліки. Наведено різні способи адресації, включаючи безпосередню, пряму, непряму, базову, індексну, сторінкову і стекову. Летально розглянуто архітектури комп'ютерів з складною, з простою, з доповненою та з орієнтованою системою команд.

В третьому розділі розкрито основні питання представлення даних в комп'ютері, які є важливими для розуміння матеріалу наступних розділів. Наведено правила

подання даних в позиційних системах числення та переведення чисел із однієї системи числення до іншої. Описано представлення чисел зі знаком в прямому, оберненому та доповняльному кодах а також формати даних з фіксованою та з рухомою комою, включаючи стандарт IEEE-754. Розглянуто питання кодування алфавітно-цифрової інформації кодами ASCII, EBCDIC та Unicode.

В четвертому розділі розкрито основні алгоритми виконання в комп'ютері операцій обробки даних: логічних, зсуву, відношення, арифметичних, обчислення елементарних функцій, перетворення даних (перетворення із формату з фіксованою в формат з рухомою комою і навпаки, перетворення з двійково-десяткового коду в двійковий і навпаки), реорганізації масивів і визначення їх параметрів (сортування, пошук максимуму або мінімуму, вибір заданого масиву, зсув елементів масиву, стиск масиву), обробки символів та стрічок символів (пошук символу, зсув, заміна символів в стрічці, пакування стрічок символів, порівняння стрічок символів).

В п'ятому розділі розглянуто принципи побудови арифметико-логічного пристрою (АЛП) сучасних комп'ютерів, який є одним з основних вузлів процесора. Розкрито структури АЛП для виконання елементарних та складних операцій. Описано багатоблокові АЛП на основі табличних, однотактових, багатотактових та конвеєрних операційних пристроїв для виконання операцій додавання, віднімання, множення, ділення та обчислення елементарних функцій над двійковими числами в форматах з фіксованою та рухомою комою.

В шостому розділі розглянуто структуру та організацію роботи пристрою керування з жорсткою логікою, а також методи його проектування: на основі таблиць станів, на основі елементів часової затримки та на основі лічильників. Описано роботу та основні принципи, покладені в основу побудови пристрою мікропрограмного керування. Розглянуто питання розміщення мікрокоманд в пам'яті, формат мікрокоманди і способи його оптимізації.

В сьомому розділі описано місце процесора в комп'ютері, його функції та склад, розглянуто виконання основних операцій процесора. Виділено базові принципи побудови процесора комп'ютера з складною та з простою системою команд. Описано конвеєрну структуру процесора з простою системою команд та принципи побудови суперконвеєрних, суперскалярних та векторних процесорів.

У восьмому розділі розглянуто конфлікти в конвеєрі команд та методи їх усунення. Проведено аналіз методів запобігання появі трьох класів конфліктів: структурних, за даними та керування. Наведено приклади структур конвеєрних процесорів, в тому числі суперскалярних, в яких зменшено імовірність виникнення конфліктів. Розглянуто архітектури комп'ютерів, в яких відсутні конфлікти команд, а саме комп'ютерів з довгим форматом команди.

В дев'ятому розділі розглянуто структуру та основні характеристики пам'яті комп'ютера. Проаналізовано можливі варіанти організації регістрових файлів процесорів та пам'яті з асоціативним доступом. Описано будову запам'ятовуючих пристроїв, які може містити основна пам'ять, способи прискорення доступу до інфор-

мації, нарощуваннярозрядності та ємності основної пам'яті. Подано типи сучасних дискових систем та обгрунтування використання масивів магнітних дисків з надлишковістю, а також робота шести базових типів дискових масивів RAID: RAID О, RAID I, ... , RAID 5 та дискових масивів, створених на їх основі. Описано роботу оптичної пам'яті та пам'яті на магнітних стрічках.

Виходячи з принципу ієрархічної організації пам'яті, в десятому розділі описано організацію взаємодії між рівнями ієрархічної пам'яті. Зокрема, наведено принципи обміну інформацією між рівнями ієрархічної пам'яті, характеристики, які використовуються для оцінки ефективності ієрархічної пам'яті, пояснено місце кеш пам'яті в складі комп'ютера, призначення і логіку роботи кеш пам'яті, описано переваги поділу кеш пам'яті першого рівня на кеш пам'ять даних та кеш пам'ять команд. Обгрунтовано різноманітність методів відображення основної пам'яті на кеш пам'ять, показано якими методами забезпечується узгодженість вмісту основної і кеш пам'яті, чим обумовлене введення додаткових рівнів кеш пам'яті й які чинники впливають на вибір ємності кеш пам'яті і розміру блоку. Введено поняття статичного і динамічного розподілу пам'яті, поняття віртуальної пам'яті та сторінкової організації пам'яті. Описано алгоритми заміщення, які використовуються при завантаженні в основну пам'ять вмісту зовнішньої пам'яті, призначення буфера швидкого перетворення адреси. Розглянуто сегментну організацію пам'яті та питання захисту пам'яті.

В одинадцятому розділі наведено пояснення способів розпізнавання пристроїв введення-виведення з використанням шини введення-виведення, лінії активації та прихованого пам'яттю введення-виведення. Наведено схему та описано функції інтерфейсної схеми пристроїв введення-виведення. Наведено чотири загальних методи керування введенням-виведенням і пояснено суть, переваги та недоліки програмно керованого введення-виведення. Введено основні поняття та характеристики системи переривання програм. Наведено опис прямого доступу до пам'яті, його переваги і недоліки, описано організацію введення-виведення під керуванням периферійних процесорів.

У дванадцятому розділі розглянуто питання подальшого підвищення продуктивності обробки інформації шляхом створення паралельних комп'ютерних систем. Наведено основні положення класифікації комп'ютерних систем, запропоновані Шором та Фліном, і відповідні класифікаціям структури комп'ютерних систем, зокрема з спільною та з розподіленою пам'яттю, з однорідним доступом до пам'яті та з неоднорідним доступом до пам'яті, а також лише з кеш пам'яттю. Наведено типи комунікаційних мереж багатопроцесорних систем.

Головною ідеєю книги є концентроване викладення та пояснення базових концепцій побудови сучасних комп'ютерів без заглиблення в роботу конкретних комп'ютерних систем, що дає змогу зорієнтуватися у взаємозалежностях та зв'язках елементів архітектури комп'ютера з метою забезпечення її цілісного бачення і розуміння, та закласти фундамент, який дозволяє в разі потреби набути детальнішої спеціалізації.

Матеріал підручника написано на основі курсів лекцій, прочитаних автором в Національному університеті "Львівська політехніка" та в кількох закордонних університетах, та на основі його багаторічної наукової діяльності. Хотів би виразити щиру подяку багатьом колегам і друзям за підтримку та допомогу в цій моїй діяльності. Окрема подяка доцентам кафедри ЕОМ Національного університету "Львівська політехніка" В. С. Глухову та В. В. Троценку за обговорення та поради, зроблені під час написання підручника. І, звичайно, велика подяка моїй дружині Тетяні та синам Віктору і Юрію за терпіння та розуміння необхідності виконання цієї роботи.

Ми з вдячністю приймемо всі зауваження та побажання.

Автор

Розділ 1

Сучасний колт'ютер. О сші&ні поняття

В цьому розділі розглядаються історичні аспекти розвитку комп'ютерів, подаються основні поняття, функції та основні функціональні вузли комп'ютера, їх взаємозв'язок, а також загальна організація роботи комп'ютера. Проводиться аналіз та даються пояснення одиниць вимірювання технічних характеристик комп'ютера та аналізуються закономірності їх зміни з розвитком елементної бази. Вводиться поняття архітектури комп'ютера та виділяються особливості архітектури Джона фон Неймана та відмінні риси комп'ютерів ненейманівських архітектур. Розглядаються характерні риси та сфери застосування різних типів комп'ютерів: персональних, робочих станцій, багатотермінальних систем, серверів, великих універсальних комп'ютерних систем, кластерів, мікроконтролерів, суперкомп'ютерів та спеціалізованих комп'ютерів. Обґрунтовується місце предмету даної книги серед суміжних дисциплін та порядок його розгляду.

1.1. Історичні аспекти розвитку комп'ютерів

В 1946 році Джон фон Нейман (John von Neumann) описав архітектуру комп'ютера, яка є найпоширенішою і в даний час. В своїй науковій роботі він назвав новий пристрій обчислювальним інструментом (computing instrument), звідки логічно появилася сучасна назва комп'ютера. Ця архітектура дістала назву архітектури Джона фон Неймана, хоча в зарубіжній літературі частіше використовується термін "Instruction Set Architecture" ("архітектура системи команд"), що вказує як на рівень опису архітектури комп'ютера, так і на програмний принцип його роботи. До перших розробників та впроваджувачів цієї архітектури належать Чарльз Бебідж (Charles Babbedz), конструктор двох механічних комп'ютерів - різницевої та аналітичної машини (1822-1833 рр.), Джон Атанасов (John Atanasofr) - автор першого спеціалізованого комп'ютера ABC (1939 рік), Конрад Цузе (Konrad Cuze) (1936-1944 рр.) та Ховард Айкен (Howard Aiken) (1941-1946 рр.) конструктори перших електромеханічних комп'ютерів відповідно Z1-Z4 та Магк1-Марк2, Алан Тюрінг (Allan Turing) - один з розробників першого електронного комп'ютера Colossus (1943 рік), Преспер Екерт (Presper Eckert) та Джон Моучлі (John Mauchly) - розробники першого універсального комп'ютера ENIAC (1946 рік), який був описаний в науковій роботі Джона фон Неймана, Маурiс Уілкс (M. Wilkes) - розробник проекту комп'ютера EDSAC (1946 рік).

До перших електронних комп'ютерів з програмним керуванням належить і мала електронно-лічильна машина (МЕЛМ), створена в 1951 році в Інституті електродинаміки Академії Наук України, м. Київ під керівництвом академіка С. А. Лебедева (рис. 1.1).

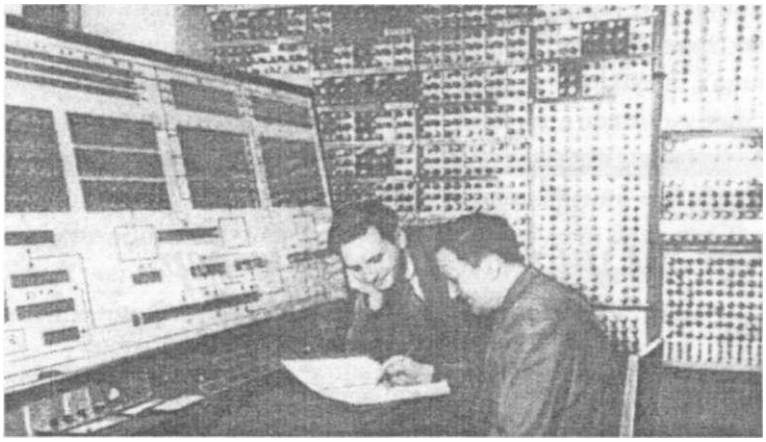


Рис. 1.1. Перший в СРСРуніверсальний електронний комп'ютер МЕЛМ (Україна, 1951)

Сучасний стан технології проектування комп'ютерів базується на теоретичних роботах ряду видатних дослідників, в першу чергу С. Крея, засновника фірми Cray Research, яка випустила кілька поколінь найпотужніших в світі комп'ютерів; М. Фліна, який розробив найуживанішу класифікацію комп'ютерів, Г. Амдаля, який сформував ряд новітніх положень розвитку комп'ютерів, Р. Томасуло, який запропонував підхід до вирішення питання ефективного завантаження комп'ютера, Д. Коука, який запропонував архітектуру "Америка", на основі якої збудовано комп'ютер з спрощеною системою команд RS/6000 фірми IBM (усі вони є вихідцями з цієї фірми), Д. Хеннессі (Стенфордський університет), автора архітектури комп'ютера з спрощеною системою команд MIPS, засновника фірми MIPS (зараз це підрозділ фірми Silicon Graphics, однієї з провідних фірм по виготовленню суперкомп'ютерів), Д. Паттерсона (Каліфорнійський університет), автора архітектури комп'ютерів з спрощеною системою команд RISC I та RISC II, технології RAID збереження даних тощо. їх ідеї були апробовані шляхом створення та практичного використання комп'ютерів фірм Cray Research, IBM, Silicon Graphics, Intel, MIPS, SUN Microsystems, Digital Equipment Corporation, Transmeta та багатьох інших.

1.2. Функції, структура та характеристики комп'ютера

1.2.1. Функції та основні функціональні вузли комп'ютера

Комп'ютер представляє собою електронний пристрій, який містить апаратні засоби і програмне забезпечення та автоматично, відповідно до програми, виконує алгоритм вирішення заданої задачі. Під алгоритмом розуміють точний припис, що задає обчислювальний процес вирішення задачі, а під задачею - сформульоване намагання отримати з множини вхідних даних і початкових умов та з множини можливих вихідних даних підмножину вихідних даних, що повністю задовольняють початкові умови і вхідні дані. Алгоритм характеризується множиною параметрів вхідних, проміжних та вихідних даних, правилом вводу даних, правилом початку, правилом обробки даних, правилом закінчення, правилом виводу даних. Для виконання алгоритму комп'ютер приймає вхідну

інформацію в цифровій формі, обробляє її відповідно до вказівок команд програми виконання алгоритму, та видає результати обчислень.

До основних функцій, які виконує комп'ютер, належать наступні:

> сприйняття вхідної інформації - вхідних даних, які підлягають обробці, та програм вирішення задач (програм обробки вхідних даних);

> зберігання інформації, тобто вхідних і проміжних даних та результатів обчислень, програм вирішення задач, довідникової інформації, програм операційної системи комп'ютера і т. д.;

> виконання арифметичних, логічних та інших операцій;

> автоматичне керування роботою складових частин комп'ютера, їх взаємодією між собою та з зовнішніми пристроями відповідно до програми;

> виведення результатів обчислень.

Для забезпечення виконання цих функцій до складу комп'ютера повинні входити такі основні функціональні вузли: пристрої введення інформації, пристрої виведення інформації, пам'ять та процесор, як це показано на рис. 1.2.

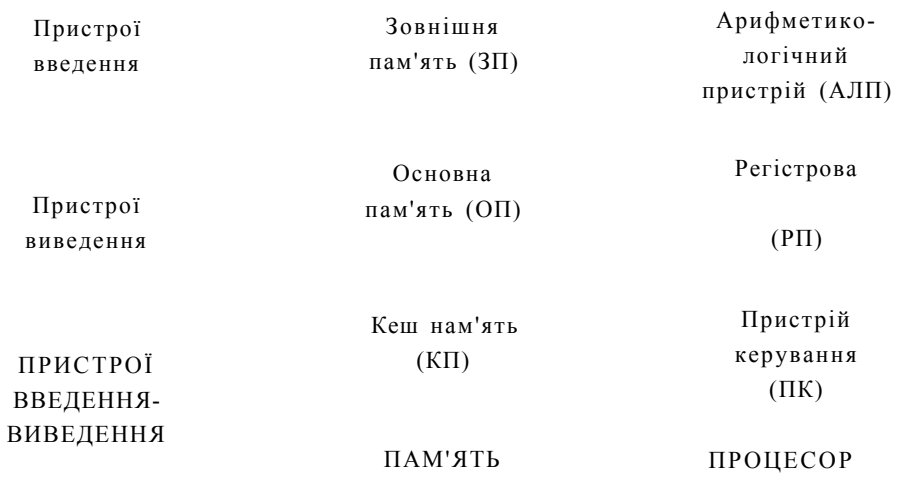


Рис. 1.2. Основні функціональні вузли комп'ютера

Коротко зупинимось на функціях та складі кожного вузла комп'ютера.

Пристрої введення-виведення виконують введення та виведення інформації. До числа пристроїв введення-виведення належать:

> пристрої введення - клавіатура, миша, сканер, відеокамера і т. д.;

> пристрої виведення - монітори (з електронно-променевою трубкою та рідкокристалічні), принтер, графопобудовувач і т. д.

Пам'ять призначена для зберігання інформації. Типи пам'яті:

> кеш пам'ять (КП) - високошвидкісна пам'ять невеликої ємності, використання якої дозволяє прискорити обмін інформацією між основною пам'яттю і процесором.

> основна пам'ять (ОП) - пам'ять великої ємності, яка зберігає інформацію, що підлягає обробці в процесорі. До інформації, яка зберігається в ОП, належать вхідні дані, які підлягають обробці відповідно до виконуваного алгоритму, результати проміжних обчислень, вихідні дані, команди програми виконання алгоритму, активна частина операційної системи комп'ютера. Інформація в ОП постійно змінюється, тобто це пам'ять для короткотермінового зберігання інформації;

> зовнішня пам'ять (ЗП) - пам'ять великої ємності для зберігання всієї інформації комп'ютера. Якщо при вимкненні комп'ютера інформація в ОП пропадає, тобто вона є енергозалежною, то на інформацію ЗП вимкнення комп'ютера не впливає, тобто це є енергонезалежна пам'ять для довготермінового зберігання інформації.

Процесор виконує обробку інформації та керує роботою інших вузлів комп'ютера. До складу процесора входять наступні функціональні вузли:

> арифметико-логічний пристрій (АЛП) - набір комбінаційних схем, які виконують арифметичні, логічні та інші операції;

> регістрова пам'ять (РП) - набір програмно доступних регістрів, в яких зберігається найчастіше використовувана в процесорі інформація;

> пристрій керування (ПК) - керує роботою та взаємодією функціональних вузлів комп'ютера.

1.2.2. Тенденції зміни основних характеристик апаратних засобів комп'ютера

Комп'ютер складається з апаратних засобів і програмного забезпечення. Апаратні засоби є базовим рівнем комп'ютера, на якому можуть бути створені багато вищих рівнів шляхом розроблення та завантаження відповідного програмного забезпечення.

Характеристики апаратних засобів в значній мірі впливають на споживчі властивості комп'ютера. Тому розглянемо, якими показниками характеризуються основні вузли комп'ютера.

Одна з важливих характеристик пристроїв введення-виведення - швидкість введення та виведення інформації. Вона залежить від типу пристрою введення-виведення і лежить в широких межах: від одиниць байтів за секунду для клавіатури до сотень мільйонів байтів за секунду для відеокамери.

Основні характеристики регістрового файлу процесора, кеш, основної та зовнішньої пам'яті це ємність та час доступу до даних. Ємність ЗП у десятки, а часто і у тисячі разів перевищує ємність ОП. Наприклад, у сучасних персональних комп'ютерах ємність ОП знаходиться в межах від сотень мільйонів до кількох мільярдів байтів, тоді як ємність ЗП знаходиться в межах від десятків до сотень мільярдів байтів.

Розділення пам'яті на ОП та ЗП викликане сповільненням її роботи із збільшенням ємності. Так, якщо час звернення до ЗП становить тисячні частки секунди, то час звернення до ОП становить мільйонні частини секунди. Це ж саме стало причиною появи кеш пам'яті, яка має меншу ніж ОП ємність, але за часом запису-зчитування даних є значно ближчою до регістрової пам'яті процесора.

Основними характеристиками процесора є його продуктивність, точність та динамічний діапазон представлення даних. Продуктивність залежить від частоти роботи процесора та його структури і вимірюється кількістю виконуваних операцій за секунду. Сучасні процесори мають продуктивність понад 1 мільярд операцій за секунду.

Крім того, всі вузли комп'ютера характеризуються швидкістю обміну з іншими пристроями, надійністю, ціною, споживаною потужністю, масою, габаритами, стійкістю до зовнішніх факторів впливу (температурний діапазон, вологість, вібрація і т. д.). Значення цих характеристик в значній мірі визначаються досягненнями інтегральної технології, зокрема проектними розмірами напівпровідникових елементів кристалу, які впливають як на ємність кристалу, а відповідно на масу та габарити, так і на частотні параметри його елементів.

Для того, щоб успішно проектувати комп'ютери та знати перспективи їх розвитку, необхідно знати тенденції зміни з часом перерахованих вище характеристик комп'ютера, в першу чергу характеристик його елементної бази.

Продуктивність. Продуктивність комп'ютера змінюється через зміну характеристик його функціональних вузлів. Розглянемо як це відбувається на прикладі процесора.

З кожним роком зростає ємність кристалу, що дозволяє розмістити на ньому все складніші схеми. Відповідно до емпіричного закону Мура, відкритого ним в 1965 році, кількість транзисторів, які вдається розмістити на кристалі мікросхеми, подвоюється кожні 12 місяців. В середині 90-х років Мур скоригував свій закон, який тепер твердить, що кількість транзисторів, які вдається розмістити на кристалі мікросхеми, подвоюється кожні 18 місяців, що підтверджує рис. 1.3, де наведено кількість транзисторів на кристалах процесорів провідних виробників світу.

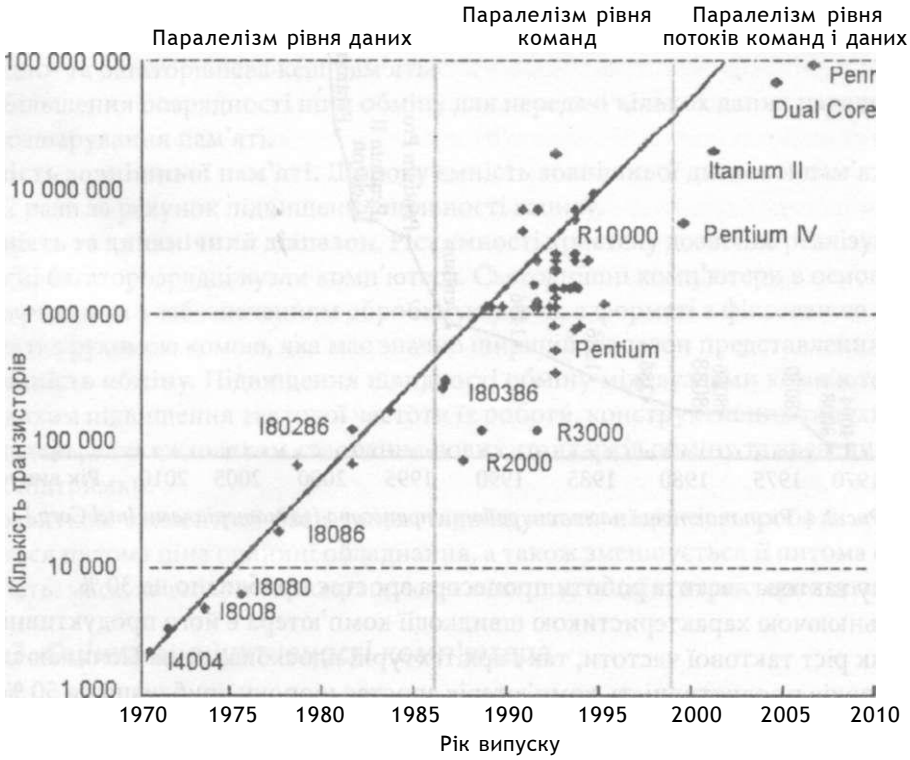


Рис. 1 3. Ріст ємності кристалу з роками

З ростом ємності кристалу з'являються можливості модифікації структури процесора шляхом ускладнення його основних вузлів та введення допоміжних вузлів, які дозволяють прискорити обробку даних. Метою такої модифікації є в першу чергу підвищення продуктивності процесора. Як видно з рис. 1.3, до середини 80-х років для цього використовувався паралелізм рівня даних, тобто суміщення в часі обробки в процесорі деякої кількості даних, далі паралелізм рівня команд, тобто суміщення в часі виконання в процесорі деякої кількості різних команд. Сьогодні це паралелізм рівня потоків команд і даних. Всі названі типи паралелізму будуть розглянуті далі при розгляді організації роботи процесора.

Зі зменшенням розмірів транзисторів, які реалізуються на кристалі мікросхеми, підвищується і тактова частота роботи процесора (рис. 1.4).

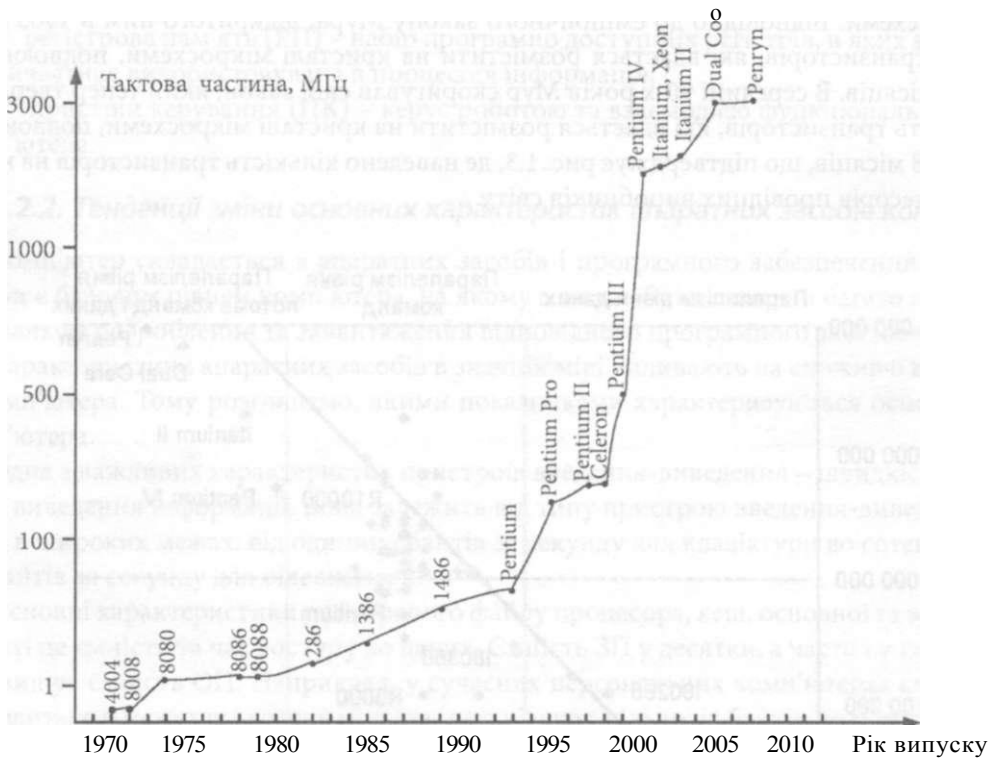


Рис. 1.4. Ріст тактової частоти роботи процесора (за матеріалами Intel Corp.)

Щороку тактова частота роботи процесора зростає приблизно на ЗО %.

Узагальнюючою характеристикою швидкодії комп'ютера є його продуктивність, яка враховує як ріст тактової частоти, так і архітектурні вдосконалення. Починаючи з середини 80-х років продуктивність комп'ютерів зростає щороку приблизно на 50 %.

Ємність основної пам'яті. Ємність динамічної напівпровідникової пам'яті з довільним доступом (Dynamic Random Access Memory - DRAM) зростає в чотири рази кожних три роки (рис. 1.5).

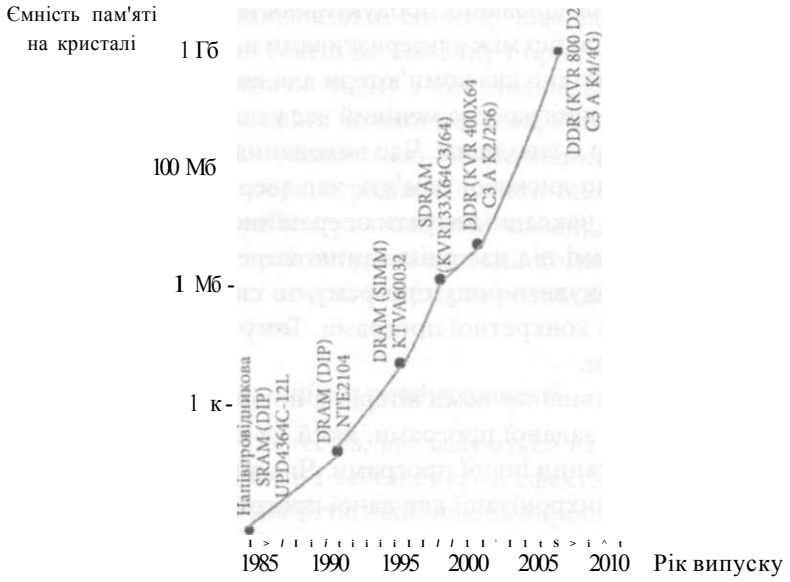


Рис. 1.5. Ріст ємності динамічної напівпровідникової пам'яті

Час читання/запису D R A M зменшується в два рази за чотири роки, тоді як в процесорі за той же час частота зростає більш як в три рази. Для узгодження швидкодії процесора та пам'яті використовуються наступні підходи та засоби:

- одно- та багаторівнева кеш пам'ять;
- збільшення розрядності шин обміну для передачі кількох даних паралельно;
- розшарування пам'яті.

Ємність зовнішньої пам'яті. Щороку ємність зовнішньої дискової пам'яті збільшується в 2 рази за рахунок підвищення щільності запису.

Точність та динамічний діапазон. Ріст ємності кристалу дозволяє реалізувати в ньому складні багаторозрядні вузли комп'ютера. Сьогоднішні комп'ютери в основному є 32- та 64-розрядними з забезпеченням обробки даних як в форматі з фіксованою комою, так і в форматі з рухомою комою, яка має значно ширший діапазон представлення даних.

Швидкість обміну. Підвищення швидкості обміну між вузлами комп'ютера досягається шляхом підвищення тактової частоти їх роботи, конструктивних та технологічних вдосконалень, а також шляхом створення нових стандартів обміну та введення пристроїв для їх підтримки.

З розвитком елементної бази також підвищується надійність роботи комп'ютера, знижується питома ціна одиниці обладнання, а також зменшується її питома споживана потужність, маса, та габарити, покращуються експлуатаційні характеристики.

1.2.3. Оцінка продуктивності комп'ютера

1.2.3.1. Одиниці оцінки продуктивності

Продуктивність є однією з основних характеристик комп'ютера, яка залежить як від технологічних, так і від архітектурних рішень. В процесі розвитку комп'ютерної техніки

з'явилося декілька методик вимірювання продуктивності. Вони дозволяють розробникам і користувачам здійснювати вибір між альтернативами на основі кількісних показників.

Припустимо, що використано два комп'ютери для виконання тієї ж програми. Якщо перший комп'ютер виконав програму за менший час у порівнянні з другим, можна говорити, що перший комп'ютер є швидшим. Час виконання програми включає час роботи процесора, час звернення до дискової пам'яті, час звернення до основної пам'яті, час введення-виведення даних і накладні витрати операційної системи. Оскільки при роботі в мультипрограмному режимі під час очікування введення-виведення для однієї програми, процесор може виконувати іншу програму, то система не обов'язково мінімізуватиме час виконання даної конкретної програми. Тому вказаний підхід до порівняння комп'ютерів не є досконалим.

Іншим підходом до порівняння комп'ютерів є порівняння часу роботи процесора, необхідного для виконання заданої програми, який не включає час очікування введення-виведення або час виконання іншої програми. Час роботи процесора може бути виражений кількістю тактів синхронізації для даної програми, помноженою на тривалість такту синхронізації.

Важливою та часто вживаною характеристикою є середня кількість тактів синхронізації процесора на одну команду CPI (clock cycles per instruction). При відомій кількості виконуваних команд в програмі ця характеристика дозволяє швидко оцінити час роботи процесора, необхідний для виконання заданої програми.

Досконалішою одиницею, яку можна використати для порівняння комп'ютерів, є продуктивність, тобто загальна кількість обчислювальної роботи, яку комп'ютер виконує за фіксований часовий інтервал. Якщо час виконання деякої програми позначити через T, то продуктивність P комп'ютера можна визначити наступним чином: P = 1/T. Тоді порівняння двох комп'ютерів X і Y можна виконати за наступними правилами: якщо 1/Tx > I/Ty тобто Ty > Tx, то комп'ютер X є швидшим. В сучасних комп'ютерах продуктивність вимірюється в мільйонах операцій за секунду - MIPS. Таким чином, продуктивність може бути визначена як зворотна до часу виконання величина, причому швидші комп'ютери при цьому матимуть вищий рейтинг кількості операцій за одиницю часу. Позитивними сторонами кількості операцій за одиницю часу як одиниці оцінки продуктивності комп'ютера є те, що цю характеристику легко зрозуміти, особливо покупцю, і що швидший комп'ютер характеризується більшим числом операцій за одиницю часу. Проте використання цієї одиниці як метрики для порівняння натрапляє на дві проблеми. По-перше, вона залежить від набору команд процесора, що ускладнює порівняння комп'ютерів з різними системами команд. По-друге, навіть на одному і тому ж комп'ютері вона змінюється від програми до програми.

Вимірювання продуктивності комп'ютерів при вирішенні науково-технічних задач, в яких переважно використовується представлення даних в форматі з рухомою комою, завжди викликало особливий інтерес. Саме для таких обчислень вперше постало питання про вимірювання продуктивності, а за досягнутими показниками часто робилися висновки про загальний рівень розробок комп'ютерів. Зазвичай для науково-технічних завдань продуктивність комп'ютера оцінюється в кількості операцій з рухомою комою за секунду FLOPS (Floating Point Operations Per Second). В сьогоднішніх комп'ютерах це мільйони та мільярди операцій з рухомою комою за секунду - MFLOPS, GFLOPS.

Потрібно відзначити, що вищеназвані одиниці вимірювання - такт (або частота) синхронізації, середня кількість тактів на команду і продуктивність комп'ютера є взаємозв'язаними. Неможливо змінити жодну з них ізольовано від іншої, оскільки базові технології, використовувані для зміни кожної з цих характеристик, взаємозв'язані: частота синхронізації визначається технологією виготовлення апаратних засобів і функціональною організацією процесора; середня кількість тактів на команду залежить від функціональної організації і архітектури системи команд; а кількість виконуваних в програмі команд визначається архітектурою системи команд і технологією компіляторів. Коли порівнюються два комп'ютери, необхідно розглядати всі три компоненти, щоб зрозуміти відносну продуктивність.

1.2.3.2. Тестові програми для оцінки продуктивності

Важливість створення пакетів тестів, що базуються на реальних прикладних програмах широкого кола користувачів і забезпечують ефективну оцінку продуктивності комп'ютерів, була усвідомлена більшістю найбільших виробників комп'ютерних засобів, які в 1988 році заснували неприбуткову корпорацію SPEC (Standard Performance Evaluation Corporation). Основною метою цієї організації є розробка і підтримка стандартизованого набору спеціально підібраних на основі досвіду тестових програм для оцінки продуктивності новітніх поколінь високопродуктивних комп'ютерів.

Основним результатом роботи SPEC є набори тестів. Ці набори розробляються SPEC з використанням кодів, що поступають з різних джерел. SPEC працює над встановленням цих кодів на різні платформи, а також створює інструментальні засоби для формування з цих кодів осмислених робочих навантажень. В даний час є два базові набори тестів SPEC, що орієнтовані на інтенсивні розрахунки і вимірюють продуктивність процесора, пам'яті, а також ефективність генерації коду компілятором. Набір тестів CIXT, що вимірює продуктивність процесора при обробці цілих чисел, складається з програм, написаних на мові С і вибраних з різних прикладних областей: теорія дробів, інтерпретатор мови LISP, розробка логічних схем, пакування текстових файлів, електронні таблиці і компіляція програм. Набір тестів CFP, що вимірює продуктивність процесора при обробці чисел з рухомою комою, складається з програм, також вибраних з різних прикладних областей: розробка аналогових схем, моделювання методом Монте-Карло, квантова хімія, оптика, робототехніка, квантова фізика, астрофізика, прогноз погоди і інші наукові та інженерні завдання. Частина програм з цього набору написана на мові С, інша частина - на Фортрані. Результати виконання кожного індивідуального тесту з цих двох наборів виражаються відношенням часу виконання однієї копії тесту на комп'ютері, що тестується, до часу її виконання на деякому еталонному комп'ютері.

Іншим підходом до оцінки продуктивності комп'ютерів за допомогою тестів є розробка спеціального програмного забезпечення (синтетичних тестів), що дозволяє створювати різні робочі навантаження, відповідно до рівня системи, що тестується, і до вимог щодо її використання. Однією з незалежних організацій, яка здійснює оцінку продуктивності комп'ютерних систем за допомогою синтетичних тестів, є приватна компанія AIM Technology, яка була заснована в 1981 році. Компанія розробляє і поставляє програмне забезпечення для вимірювання продуктивності систем, а також надає послуги з тестування систем кінцевим користувачам і постачальникам комп'ютерних систем та мереж, які використовують промислові стандартні операційні системи, такі як UNIX і

OS/2. Це програмне забезпечення складається з двох основних частин: генератора тестових пакетів і сумішей навантажень прикладних завдань. При кожному запуску генератора можуть виконуватися будь-які окремі або всі доступні тести у будь-якому порядку і при будь-якій кількості проходжень, дозволяючи тим самим створювати для системи практично довільне робоче навантаження. Якщо деякі необхідні тести відсутні у складі генератора тестових пакетів, то вони можуть бути туди легко додані. На основі оцінки продуктивності системи вибираються різні рівні збільшення навантаження. Це дозволяє з достатньою достовірністю зробити висновок про можливості роботи системи при даному навантаженні або при зміні навантаження. При цьому важливим є підбір сумішей навантажень, які ділять на дві категорії: замовлені і стандартні. Замовлені суміші створюють для точного моделювання особливостей середовища кінцевого користувача або постачальника устаткування. Стандартні суміші є звичайним середовищем прикладних завдань. До їх складу входять, зокрема, суміші для робочих станцій, геоінформаційних систем, ділових застосувань, керування базою даних та інші.

Важливим є створення тестів та оцінка продуктивності комп'ютерів на задачах з області їх конкретного застосування. Так, наприклад, фірма BDTI надає послуги з питань аналізу продуктивності комп'ютерів, призначених для вирішення задач цифрової обробки сигналів. З розширенням використання комп'ютерів при обробці транзакцій (комерційний обмін товарами, послугами або грошима) все важливішим стає забезпечення можливості їх коректного порівняння між собою. З цією метою в 1988 році була створена Рада для оцінки продуктивності обробки транзакцій (TPC - Transaction Processing Performance Council), яка є неприбутковою організацією. TPC публікує специфікації тестових пакетів, які регулюють питання, пов'язані з роботою тестів. Ці специфікації гарантують, що покупці мають об'єктивні значення даних для порівняння продуктивності різних комп'ютерних систем.

1.2.4. Організація зв'язків між функціональними вузлами комп'ютера

Для забезпечення роботи комп'ютера його функціональні вузли повинні бути відповідним чином з'єднані. Обмін інформацією між вузлами комп'ютера проводиться по шинах, до складу яких входять п ліній, де п - розрядність інформаційного слова. Є три головних типи шин: шина даних, шина керування (або шина команд) та шина адрес, які використовуються відповідно для пересилання даних, команд та їх адрес.

Три найуживаніших структури комп'ютера, які відрізняються організацією зв'язку між його функціональними вузлами, розглянуто нижче.

В двошинній структурі комп'ютера з обміном через процесор (рис. 1.6) введення інформації з пристрою введення до основної пам'яті, та виведення інформації з основної пам'яті до пристрою виведення здійснюється через процесор.

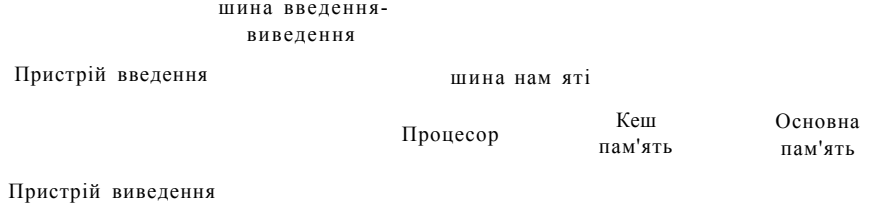


Рис. 1.6. Двошинна структура комп'ютера з обміном через процесор

Недоліком цієї структури є те, що процесор витрачає свій час на виконання операцій введення та виведення інформації, безпосередньо працюючи з пристроями введення-виведення, які в більшості випадків є значно повільнішими порівняно з ним, що знижує ефективність роботи комп'ютера.

В двошинній структурі комп'ютера з обміном через пам'ять (рис. 1.7) процесор звільнений від організації операцій введення-виведення інформації.

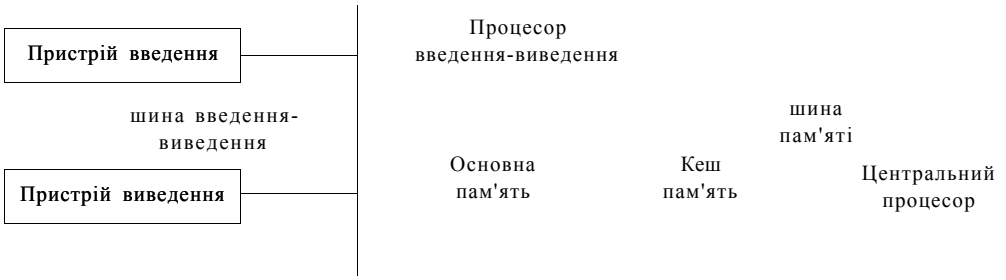


Рис. 1.7. Двошинна структура комп'ютера з обміном через пам'ять

Оскільки основна пам'ять не має засобів керування, для організації введення-виведення тут використовується спеціальний процесор введення-виведення (часто його називають каналом або периферійним процесором), який отримує завдання на виконання операцій введення-виведення від процесора (в цьому випадку останній називають центральним процесором), та керує взаємодією пам'яті та пристроїв введення-виведення. Дана структура була використана при побудові багатьох комп'ютерів, зокрема комп'ютерів серій IBM/360 та IBM/370 фірми IBM.

В одношинній структурі комп'ютера (рис. 1.8) всі його функціональні вузли з'єднані через одну спільну шину. Шина розподіляється між ними в часі, тому одночасно посилати дані на шину може тільки один вузол.

Рис. 1.8. Однотипна структура комп'ютера

Для узгодження швидкодії повільних і швидких вузлів комп'ютера в одношинній структурі на виході кожного повільного вузла (в даному випадку пристрої введення та виведення) використовують буферну пам'ять, яка дозволяє швидкий обмін блоками даних.

Кожний пристрій введення та виведення складається з двох частин - контролера та самого пристрою, наприклад, накопичувача на диску. Контролер є спеціалізованим комп'ютером, який керує своїм пристроєм та його доступом до спільної шини.

Для керування розподілом шини між підключеними до неї пристроями використовується арбітр шини. При одночасному поступленні кількох запитів арбітр шини вирі-

шує, чия черга є першою, і підключає до шини відповідний пристрій. Зазвичай перевага віддається пристроям введення-виведення, оскільки роботу дисків та інших пристроїв з рухомою механікою не можна переривати, щоб не втратити інформації.

Наявність спільної шини суттєво спрощує реалізацію комп'ютера та дозволяє легко замінювати його склад. Тому така структура отримала широке розповсюдження. Вона була покладена в основу зокрема комп'ютерів PDP-8 та PDP-11 фірми Digital Equipment Corporation, СМ-4 Київського заводу управляючих машин, та знайшла широке застосування в мікропроцесорних системах.

Для забезпечення одночасного обміну інформацією між різними пристроями в сучасних комп'ютерах використовується багатошинна організація.

1.3. Архітектура комп'ютера

1.3.1. Поняття архітектури комп'ютера

Вперше означення терміну "архітектура комп'ютера" було зроблене в 1964 році розробниками комп'ютера IBM 360 Г. Амдалем та його колегами. Архітектура комп'ютера з їх точки зору - це його структура і поведінка як 'їх бачить програміст на асемблерній мові. Вона включає наступне: формати даних і команд, методи адресації, систему команд, а також загальну організацію процесора, основної пам'яті і пристроїв введення-виведення. Пізніше А. Пейджез з тієї ж фірми запропонував розуміти під архітектурою комп'ютера інтерфейс між його апаратним та програмним забезпеченням.

Як відомо, в комп'ютері використовується двійкове представлення команд. При написанні програми крім двійкової можуть використовуватись і інші форми представлення команд: вісімкова, шістнадцяткова, символьна (мнемонічна). Використання вісімкового і шістнадцяткового кодування дозволяє скоротити записи і спростити роботу програміста. Мнемонічне кодування спрощує процес написання, читання і відлагодження програми. Основний принцип такого кодування - кожна команда представляється 3-х або 4-х буквеним символом, який показує назву команди. Деякі приклади мнемонічного кодування: add - додати, sub - відняти, load - зчитати дані з пам'яті, store - записати дані в пам'ять. Операнди також представляються символічно. Наприклад, команда add R, Y означає додавання значення вмісту комірки пам'яті Y до вмісту регістра R. Зауважимо, що операція виконується над вмістом, а не над адресою комірки пам'яті та регістра. Таким чином, з'являється можливість написання машинних програм в символічній формі. Повний набір символічних назв і правила їх використання утворюють мову програмування, відому як асемблерна мова. Запис деякої програми на асемблерній мові представляє собою символічний запис цієї ж програми, написаної на внутрішній мові комп'ютера, тобто в формі послідовності команд, представлених в двійкових кодах. Тому і з'явилось визначення архітектури комп'ютера як інтерфейсу між його апаратним та програмним забезпеченням.

З наведених вище означень можна зробити висновок про існування множини можливих варіантів архітектури комп'ютера. Розглянемо детальніше складові поняття архітектури з тим, щоб визначити ступінь зміни їх характеристик.

Під даними розуміються числа, представлені в деякій системі числення. В сучасних комп'ютерах дані в більшості випадків представлені в позиційній двійковій системі чис-

лення, яка практично витіснила інші способи представлення чисел, наприклад, представлення в позиційній десятковій системі числення, в системі залишкових класів тощо. В комп'ютерах є три класи даних: вхідні, проміжні та вихідні, які можна охарактеризувати наступною множиною параметрів: кількістю N даних, їхрозрядністю п (п= 1,2,3,...), способом кодування М (прямий, обернений чи доповняльний код), формою представлення Б (фіксована чи рухома кома), форматом Т (місце розміщення знаку, ціле чи дробове число, місце розміщення коми, розрядність порядку, основа порядку, використання знаку порядку чи зміщення, розрядність мантиси, чи використовується нормалізація мантиси). Таким чином, залежно від значення параметрів п, М, Б, Т можна виділити відповідну архітектуру комп'ютера. Наприклад, це може бути комп'ютер для виконання операцій над 16-розрядними дробовими двійковими даними в форматі з фіксованою комою в доповняльному коді, або це може бути комп'ютер для виконання операцій над 32-розрядними двійковими даними в форматі з рухомою комою.

Команда в комп'ютері зберігається в двійковій формі. Вона вказує тип операції, яка має бути виконаною, адреси операндів, над якими виконується операція, та адреси розміщення результатів виконання операції. Відповідно до цього команда складається з двох частин: коду операції та адресної частини. Поле коду операції займає к розрядів. Ним може бути закодовано до $N = 2^{к}$ різних операцій. Поле адреси (адресна частина) займає т розрядів. В ньому знаходяться адреси операндів. Кожна адреса займає т розрядів, де і - номер адреси (i=1, 2....1), 1 - кількість адресних полів. Кожною адресою можна адресувати пам'ять ємністю $2^{т}$ слів. Таким чином, залежно від формату команди комп'ютери можна поділити на: комп'ютери з постійним та змінним форматом команди, комп'ютери з одноадресними, двоадресними та триадресними командами, комп'ютери з вузьким та широким форматом команди (залежно від кількості полів коду операції та адресних полів) і т. д.

Як вказано вище, крім коду операції до складу команди входить адресна частина. Цією частиною визначається місце знаходження даних, над якими виконується операція, задана кодом операції. Даних може бути декілька, і, крім того, вони можуть знаходитися в основній пам'яті, в регістрах процесора чи в запам'ятовуючих елементах інших вузлів комп'ютера. Тому і формати команд в цих випадках будуть різними. Можна здійснити класифікацію архітектури комп'ютера за типом адресованої пам'яті. Залежно від того, який тип пам'яті адресується, розрізняють наступні типи архітектур комп'ютера: стекова, акумуляторна, на основі регістрів загального користування.

При розробці комп'ютера необхідно враховувати багато особливостей вибору системи команд. З одного боку, система команд повинна бути функціонально повною, тобто комп'ютер повинен забезпечувати виконання всіх заданих функцій. З іншого боку, система команд має бути ортогональною, тобто не повинна бути надлишковою. Для нового комп'ютера, який є розширенням відповідної серії, першочерговою є сумісність - програми, які виконуються на одному комп'ютері, повинні виконуватись і на іншому комп'ютері. Є багато рівнів повноти системи команд. Теоретично система команд комп'ютера може включати лише одну команду. Однак програми на базі простих операцій є дуже складними. Існує фундаментальний зв'язок між простотою комп'ютера і складністю програми. В реальних комп'ютерах застосовується система команд, до складу якої входить широкий спектр команд обробки даних, переміщення даних, передачі керування та введення-

виведення. За складом системи команд комп'ютери можуть бути поділені на наступні типи: комп'ютери з складною (комплексною) системою команд, комп'ютери з простою (спрощеною) системою команд, комп'ютери з доповненою системою команд, комп'ютери з орієнтованою (спеціалізованою) системою команд.

Значна кількість типів архітектури комп'ютера може бути виділена залежно від організації його вузлів, а саме процесора, пам'яті і пристроїв введення-виведення. Наприклад, це можуть бути комп'ютери з паралельною та конвеєрною обробкою даних, з ієрархічною та лінійною пам'яттю і т. д.

Архітектура комп'ютера має визначальний вплив на його споживчі характеристики: коло вирішуваних на комп'ютері задач, продуктивність, ємність основної пам'яті, ємність зовнішньої пам'яті, вартість, організація технічного обслуговування, надійність і т. д.

Для вибору кращої з множини можливих варіантів архітектури комп'ютера потрібно знати зв'язок між архітектурою комп'ютера та його характеристиками, тобто як ті чи інші структурні особливості та організація роботи комп'ютера і його вузлів зв'язані з можливостями, які надаються користувачу, які є альтернативи при створенні комп'ютера і за якими критеріями повинні прийматися ті чи інші проектні рішення, як зв'язані між собою характеристики окремих пристроїв комп'ютера і який вплив вони мають на загальні його характеристики. А це саме ті питання, які є предметом даної книги.

1.3.2. Архітектурні принципи Джона фон Неймана

Описана в 1946 році Джоном фон Нейманом архітектура комп'ютера дістала назву його імені. Оскільки це був опис реалізованого Преспером Екертом та Джоном Моучлі універсального комп'ютера EKIAC, створеного в Принстонському університеті, часом її іще називають принстонською. Головні особливості архітектури комп'ютера Джона фон Неймана:

> Інформація в комп'ютері ділиться на команди і дані.

> Команди вказують комп'ютеру, які дії і над якими операндами виконувати.

> Послідовність команд, за якою виконується алгоритм вирішення задачі, називають програмою.

> Весь набір виконуваних комп'ютером команд називають системою команд комп'ютера.

> Дані - це числа і закодовані символи, які використовуються командами як операнди. Одні команди для інших також можуть бути операндами.

> Команди і дані представлено двійковим кодом.

> Немає відмінностей в представленні команд і даних. Наприклад, двійкове число 100011011100 може бути як командою, так і даним.

> Команди і дані зберігаються в одній пам'яті.

> Команди і дані зберігаються в пам'яті за відповідними адресами.

> Пам'ять має довільну адресацію, тобто в кожному такті можна звернутися до довільної її комірки.

> Пам'ять є лінійною. її адресу кодують двійковим кодом, починаючи від молодшої, всі розряди якої рівні нулю (00...0), до старшої, всі розряди якої рівні одиниці (11..'. 1).

В основу роботи цього комп'ютера покладено принцип програмного керування. Тобто функціонування цього комп'ютера здійснюється потактово за вказівкою команд про-

грами. Програма разом з даними, які підлягають обробці, спочатку записується до основної пам'яті. В кожному такті, залежно від типу архітектури комп'ютера, виконується команда або частина команди - мікрокоманда. Команда вказує тип виконуваної операції та місце розміщення операндів і результату операції. Для виконання однієї команди необхідно провести наступні дії: записати до програмного лічильника адресу команди, зчитати із основної пам'яті команду за вмістом програмного лічильника, провести дешифрування команди з метою її розпізнання, визначити адреси комірок пам'яті, в яких знаходяться операнди та до яких мають бути записані результати, зчитати ці операнди з пам'яті та подати в арифметико-логічний пристрій для опрацювання, виконати операцію над операндами та записати результати до основної пам'яті. Таким чином проводиться виконання всіх команд програми. По закінченню в основній пам'яті будуть знаходитися результати виконання програми.

Як буде видно далі, ці особливості характерні для більшості сучасних комп'ютерів.

1.3.3. Ненейманівські архітектури комп'ютерів

До цього часу більшість універсальних комп'ютерів будують за принципами архітектури Джона фон Неймана. Але недоліки цієї архітектури, пов'язані з закладеним в ній послідовним характером організації обчислень, ставлять перепони в пошуку шляхів побудови швидких комп'ютерних систем. Тому крім архітектури Джона фон Неймана за час існування комп'ютерної техніки було створено цілий ряд інших архітектур. До них, зокрема, належать гарвардська архітектура, асоціативна машина, машина потоків даних, редукційна машина. Було запропоновано покласти в якості обчислювальної парадигми нейронні мережі, в яких використана ідея з моделей мозку та генетичні алгоритми, в яких застосовані ідеї з біології та еволюції архітектури комп'ютерних мереж. Останнім часом багато уваги приділяється квантовим комп'ютерам, елементи яких працюють за законами квантової механіки, біологічним комп'ютерам, а також паралельним комп'ютерам, оскільки практично всі сьогоднішні комп'ютери є паралельними.

Коротко виділимо основні риси деяких з названих архітектур.

Гарвардська **архітектура** вперше була реалізована Ховардом Айкеном в комп'ютері Марк-1 в Гарварді. Вона передбачає розділення пам'яті на пам'ять даних і пам'ять команд. Тим самим розділяються шини передачі керуючої і оброблюваної інформації (рис. 1.9). При цьому підвищується продуктивність комп'ютера за рахунок суміщення в часі пересилання та обробки даних і команд.

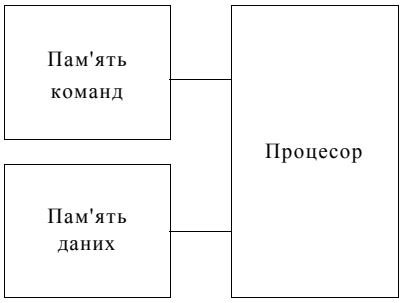


Рис. 1.9. Ядро комп'ютера гарвардської архітектури, утворене процесором та пам'яттю команд і даних

Необхідність використання гарвардської архітектури можна пояснити так. Ядро комп'ютера Джона фон Неймана складається з процесора та основної пам'яті. Бажано, аби обидві компоненти ядра не пригальмовували одна одну, тобто працювали із рівною швидкодією. На практиці вузол пам'яті є значно (на порядок) повільнішим від процесора і цей розрив у швидкодії з прогресом інтегральних технологій лише зростає. Зменшити розрив можна структурними методами, збільшуючи розрядність інформаційного слова пам'яті. Саме цей підхід реалізує гарвардська архітектура з двома запам'ятовувальними пристроями. Зрозуміло, що тут паралельно виконуються операції вибирання команд програми, з одного боку, а з другого - вибирання та запис кодів даних і результатів обчислень.

Дуальна пристонсько-гарвардська архітектура. Швидкі комп'ютери гарвардської архітектури є складнішими щодо програмування порівняно з комп'ютерами принстонської архітектури. Зрозуміло, що бажано створити комп'ютер з дуальною архітектурою, яка водночас запозичує нову якість - швидкодію від гарвардської архітектури та стандартну парадигму розробки програм від принстонської архітектури. Злиття двох архітектур виконують на рівні кеш пам'яті шляхом її поділу на кеш даних та кеш команд (рис. 1.10). Злиттям архітектур програмісту надано зручність програмних технологій принстонської архітектури, а з боку процесора реалізовано гарвардську архітектуру, в результаті чого він значно менше пригальмовується з боку основної пам'яті.

Рис. 1.10. Злиття архітектур через розділену на дві частини кеш пам'ять першого рівня

Асоціативна машина передбачає маніпуляції з даними не відповідно до їх адрес, як це є в машині Джона фон Неймана, а відповідно до значення цих даних або їх частин. Базовими тут є операції пошуку і порівняння. Основою асоціативної архітектури є асоціативна пам'ять, яка забезпечує одночасний доступ до багатьох даних, в яких співпадають значення відповідних розрядів. Тим самим за рахунок високої паралельності обробки досягається висока продуктивність на класі операцій, для виконання яких ця машина є ефективною (зокрема, логічні операції, операції пошуку та сортування). Асоціативна машина є складовою практично кожного сучасного комп'ютера.

Машина потоків даних. Керування обчислювальним процесом в машині потоків даних здійснюється даними за їх готовністю до обробки. Кожне дане в такій машині має спеціальні ознаки. За цими ознаками пристрій керування знаходить дані, які готові до обробки, і передає їх в АЛП для виконання відповідних операцій. Тим самим тут за рахунок можливого паралельного аналізу та обробки даних досягається гранично ви-

сока продуктивність. Принципи машини потоків даних використовуються в багатьох сучасних високопродуктивних комп'ютерах.

Паралельні комп'ютерні системи. Перші паралельні комп'ютерні системи, до складу яких входило лише два процесори, були побудовані в кінці 60-х років минулого століття. В 70-х роках такі системи мали в своєму складі до 64-х процесорів, в 80-х роках - до 1000, а в кінці 90-х років фірма IBM анонсувала конструкцію суперкомп'ютера з паралельною архітектурою, який включав понад мільйон процесорів і на даний час є найпродуктивнішим у світі. Паралельна обробка інформації є ключовим напрямком побудови високопродуктивних комп'ютерних систем. Однак і паралельні комп'ютерні системи мають обмеження. По-перше, зі збільшенням кількості процесорів ускладнюється задача розподілу завдань між процесорами. Для її вирішення використовуються додаткові процесори, кількість яких може значно перевищувати кількість процесорів, зайнятих безпосередньо виконанням алгоритму. По-друге, послідовна природа багатьох алгоритмів обмежує прискорення, якого можна досягти, використовуючи багатопроцесорну організацію.

1.4. Типи сучасних комп'ютерів

Аналіз сфер використання комп'ютерів показує, що можна виділити два основних напрямки їх використання. Перший напрямок - це підсилення інтелектуальних можливостей людини в досить широкому розумінні цього поняття. Мається на увазі прискорення обчислень, зберігання великих об'ємів інформації, швидкий пошук та відображення необхідної інформації і т. д. Комп'ютери цього напрямку повинні мати велику ємність пам'яті, потужне програмне забезпечення, розвинуті засоби взаємодії з людиною.

Другий напрямок - це використання комп'ютера як елемента електротехнічної системи, наприклад, системи керування, інформаційно-вимірювальної системи, системи передачі даних і т. д. На рис. 1.11 показано приклад використання комп'ютера в складі системи керування. Тут параметри об'єкта керування знімаються давачами, перетворюються в цифрову форму за допомогою аналого-цифрових перетворювачів АЦП та подаються в комп'ютер. В комп'ютері здійснюється оцінка стану об'єкта керування та виробляється послідовність керуючих кодів, які в цифро-аналогових перетворювачах ЦАП перетворюються в аналогову форму та подаються на органи керування з метою забезпечення утримання параметрів об'єкта керування в заданих межах.

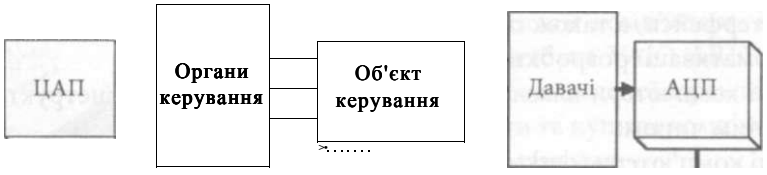


Рис. 1.11. Комп'ютер в системі керування

ЗЗам. 371.

Основна вимога до комп'ютерів даного напрямку - забезпечення необхідного часу реакції на події в системі, проведення обробки вхідних даних в темпі їх поступлення, тобто в реальному масштабі часу, і видача результатів відповідно до часових вимог електротехнічної системи. Комп'ютери цього напрямку повинні мати швидкісні зовнішні інтерфейси, володіти високою продуктивністю та надійністю роботи. Часто ці комп'ютери є вбудованими в мобільні засоби, тому додатково вимагається, щоб вони мали малі габарити та низьку споживану потужність. Вбудовані комп'ютери широко використовуються в побутовій електронній апаратурі, зокрема в відео та аудіо апаратурі, фотоапаратах, іграшках, побутових пристроях, мобільних телефонах і т.д.

Залежно від сфери застосування та технічних характеристик, в першу чергу габаритів, продуктивності, ємності пам'яті та ціни розрізняють наступні типи комп'ютерів:

> персональні комп'ютери (personal computers) - комп'ютери, орієнтовані на користування однією особою;

> робочі станції (workstations) - комп'ютери, орієнтовані на вирішення інженерних задач, в першу чергу в складі локальних комп'ютерних мереж;

> багатотермінальні системи - набір стандартних терміналів, підключених до сервера;

> сервери - центральні комп'ютери для побудови інформаційних систем;

> мейнфрейми (mainframes) - великі універсальні комп'ютерні системи;

> кластерні комп'ютерні системи - об'єднання комп'ютерів, що сприймається операційною системою, системним програмним забезпеченням, прикладними програмами і користувачами як єдине ціле;

> суперкомп'ютери - найпотужніші на даний час комп'ютери;

> мікроконтролери - комп'ютери на кристалі, призначені для керування електронними пристроями.

> спеціалізовані комп'ютери. Вони орієнтовані на вирішення задач, котрі неможливо або недоцільно виконувати на універсальних комп'ютерах.

Розглянемо названі комп'ютери детальніше.

1.4.1. Персональні комп'ютери

Персональні комп'ютери (ПК) з'явилися в результаті еволюції мінікомп'ютерів при переході елементної бази з малим і середнім ступенем інтеграції на великі і надвеликі інтегральні схеми. ПК, завдяки низькій вартості, дуже швидко завоювали тверді позиції на комп'ютерному ринку і створили передумови для розробки нових програмних засобів, що орієнтувалися на кінцевого користувача. Це, передусім, дружні по відношенню до користувача інтерфейси, а також проблемно-орієнтовані середовища і інструментальні засоби для автоматизації розробки прикладних програм.

Персональні комп'ютери класифікуються за їх розмірами та конструктивним виконанням наступним чином:

> Настільні комп'ютери (desktop computers). До складу настільного комп'ютера входить системний блок (в якому розміщена материнська плата, центральний процесор, основна пам'ять, карта розширення, блок живлення і т. д.), дисплей, клавіатура, мишка. В системний блок також вбудовані драйвер оптичного диску і зовнішня дискова пам'ять. Настільні комп'ютери призначені для офісного використання.

> Лаптопи чи ноутбуки (laptop or notebooks). Це близькі за характеристиками до настільного ПК, але конструктивно виконані в придатному для перенесення виконанні.

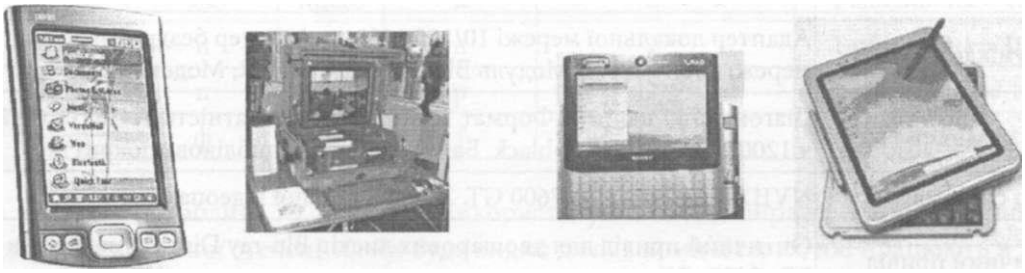
> Персональні цифрові асистенти (Personal digital assistants). Це кишенькові ПК, спеціально створені як персональні асистенти людини. Вони надають наступний сервіс: годинник, комп'ютерні ігри, доступ до мережі Інтернет, електронна пошта, записна книжка, адресна книжка, мобільний телефон, медіа плеєр та інше.

> Смартфони. Це мобільні телефони, які мають вбудовану операційну систему та можливості вище описаних персональних цифрових асистентів.

> Портативні комп'ютери (Portable computers). Це ПК типу настільних, але виконані в придатному для перенесення та роботи в не офісних умовах конструктивному виконанні.

> Переносимі комп'ютери (Wearable computers). Це комп'ютери, які надають інформаційні послуги людині під час її руху в навколишньому середовищі. До переносимих ПК належать, зокрема, комп'ютери для моніторингу стану людини.

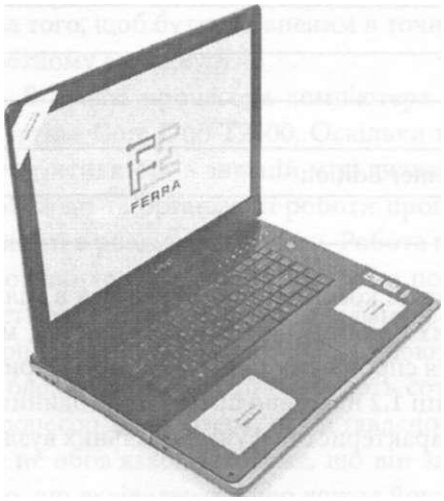
На рис. 1.12 наведено зовнішній вигляд ряду серійних персональних комп'ютерів: персональний цифровий асистент pulmone Tursten T5 (рис. 1.12a), мобільний ПК французької армії (рис. 1.12b), переносимий комп'ютер VAIO VGN-UX50 фірми Sony (рис. 1.12c), планшетний ПК фірми Hewlett Pascard (pHC.1.12d).



а ь с ci

Рис. 1.12. Зовнішній вигляд ряду серійних персональних комп'ютерів

Потрібно відзначити, що названі вище типи персональних комп'ютерів в даний час активно розвиваються та вдосконалюються, тому як термінологія так і сфера їх застосування ще не є чітко встановленими.



В якості прикладу розглянемо значення характеристик реального персонального комп'ютера. На рис. 1.13 наведено зовнішній вигляд, а в табл. 1.1 технічні характеристики, взяті з однієї із типових реклам комп'ютера, а саме мультимедійного ноутбука VAIO A R U RS. Зрозуміло, що без знання наведеної в рекламі термінології важко вибрати та купити комп'ютер з потрібними характеристиками.

Рис. 1.13. Зовнішній вигляд комп'ютера SonyVAIOARlIRS

Таблиця 1.1

Процесор	Intel Centrino Core Duo T2500: два ядра; тактова частота 2000 МГц; FSB 667 МГц
Пам'ять	Встановлено 1024 Мбайтів пам'яті DDR 2-533
Кеш пам'ять	2x32KB LI cache, 2048КБ L2 cache
Жорсткий диск	Два вінчестери ємністю по 100 ГБ кожний. Швидкість обертання 5400 об/хв. 8 МБ cache
Порти введення-виведення	3 порти USB 2.0; по одному порту: IEEE 1394 (i.LINK, FireWire); IrDA (інфрачервоний порт); PCMCIA Type II; ExpressCard|54; Secure Digital (SD/MMC); Memory Stick (MS/MS Pro/MS Duo/MS Pro Duo) - адаптер до карт пам'яті; VGA (вихід на зовнішній монітор); HDMI (High Definition Multimedia Interface); S-Video out (аналоговий відеовихід); S-Video in (аналоговий відеовхід); RJ-11 (модем); RJ-45 (мережа); оптичний S/PDIF; вихід на навушники; лінійний вхід; вхід для мікрофона
Комунікації	Адаптер локальної мережі 10/100 Мбіт/с; Адаптер бездротової мережі 802.11 a/b/g; Модуль Bluetooth 2.0 + EDR; Модем V.90, 56k
Екран	Діагональ 17 дюймів. Формат 16:9. Роздільна здатність WUXGA (1920 X 1200). Технологія X-black. Багатошарове антиблікове покриття
Відео система	NVIDIA GoForce Go 7600 GT. 256 МБ власної відеопам'яті
Оптичний привід	Оптичний привід для двошарових дисків Blu-ray Disc. Запис дисків BD-R/BD-RE
Аудіо	Intel High Definition Audio. Вбудовані стерео динаміки та мікрофон
Пристрої введення	Клавіатура. Сенсорна панель (touchpad)
Додаткові пристрої	Вбудовані ТВ-тюнер та веб-камера VGA (640x480)
Електроживлення	Літій-іонна батарея (VGP-BPS2C). Час автономної роботи до 120 хвилин
Розміри	416x299,5x33,5 мм
Вага	3,8 кг
Гарантія	12 місяців
Операційна система	Microsoft Windows XP Media Center Edition
Ціна	$3000

Як бачимо, характеристики функціональних вузлів комп'ютера оцінюються в мільйонах, мільярдах, трильйонах чи трильярдах деяких одиниць, або в їх мільйонних, мільярдних, трильйонних та трильярдних частках. Для спрощення запису цих характеристик використовуються відповідні префікси. В таблиці 1.2 наведено префікси до одиниць вимірювання великорозмірних та малорозмірних характеристик функціональних вузлів комп'ютера кирилицею та латинськими буквами.

Таблиця 1.2

Назва префікса	Позначення префікса	Назва на латині	Позначення на латині	Значення префікса, виражене через степінь двох	Близьке значення, виражене через степінь десяти
Великорозмірні характеристики					
Кіло	К	Kilo	К	$_{2}$ Ю	1 тисяча = 10^{3}
Мета	М	Mega	М	$_{,}$20	1 мільйон = 10^{6}
Гіга	Г	Giga	G	$_{,}$30	1 мільярд = 10^{9}
Тера	т	Тега	Т	$_{,}$40	1 трильйон = 10^{12}
Пета	п	Peta	Р	$_{,}$50	1 трильярд = 10^{15}
Малорозмірні характеристики					
Мілі	м	Milli	m	$_{2}$-io	1 тисячна =10°
Мікро	мк	Micro	I^{i}	$_{,}$-20	1 мільйонна = 10^{6}
Нано	н	Nano	п	$_{,}$-/o	1 мільярдна = 10^{9}
Піко	п	Pico	Р	2-40	1 трильйонна = 10^{12}
Фенто	ф	Fento	f	$_{,}$-50	1 трильярдна = 10^{15}

Наведені в таблиці 1.2 префікси використовуються для вираження відповідної додатної або від'ємної степені 10. Як буде видно далі, в комп'ютерах, які працюють в двійковій системі числення, значна частина характеристик вимірюється в одиницях, кратних степені 2. Але оскільки люди вже звикли до цих префіксів, їх же використовують і для вираження відповідної степені 2 чисел, близьких степені 10. Наприклад, ємність 1 КБ (кілобайт) означає 1024 байти, оскільки значення 210 - 1024 є близьким до 1000. В останньому стовпчику таблиці 1.2 наведено значення степені 10 для близьких значень степені 2.

Разом з тим, необхідно зауважити, що дискова пам'ять ємністю 1 ГБ має 1 мільярд байтів, а не 230 (приблизно 1.7 мільярда). Тому потрібно читати інструкцію користувача для того, щоб бути впевненим в точному значенні запису 1К, ЇМ, чи 1Г в кожному конкретному випадку.

В якості процесора комп'ютера Sony VAIO A R U RS використано процесор Intel Centrino Core Duo T2500. Оскільки процесор є основним елементом комп'ютера, його продуктивність в значній мірі визначає продуктивність всього комп'ютера. Принципи побудови та організації роботи процесора та його функціональних вузлів будуть розглянуті в розділах 4-8 книги. Робота процесора синхронізується імпульсами, які генерує блок синхронізації комп'ютера та посилає їх до всіх його основних компонентів. Кількість вироблених цим блоком імпульсів за 1 секунду є його частотою, яка вимірюється в герцах. Сучасні комп'ютери працюють на мегагерцових та гігагерцових частотах, тобто їх блоки синхронізації генерують сотні мільйонів або й мільярди імпульсів за секунду. Процесор комп'ютера, представленого на рекламі, функціонує на частоті 2 ГГц. Однак це не обов'язково означає, що він здатний виконувати 2 мільярди команд за секунду, або, що еквівалентно, що кожна його команда виконується за 0,5 наносекунди. Як буде

показано в розділі 2, кожна команда комп'ютера вимагає фіксованої кількості тактів для свого виконання, причому деякі команди виконуються за один такт, але більшість команд виконуються за декілька тактів.

В другому рядку реклами розміщено "1024 Мбайтів пам'яті DDR 2-533)". Це основна пам'ять. Число 533 вказує частоту зчитування даних з цієї пам'яті на системну шину, по якій проводиться обмін інформацією в комп'ютері. Тут частота вимірюється в МГц. Основна пам'ять має ємність 1024 мегабайти (МБ). Питання про формати представлення даних в комп'ютері будуть розглянуті в розділі 3, а питання побудови та організації роботи пам'яті - в розділі 9. Додатково до ємності пам'яті реклама вказує і її тип DDR, скорочення від dynamic random access memory (динамічна пам'ять з довільним доступом), яка підтримує двоканальний режим.

Наступний рядок в рекламі "2x32КБ LI cache, 2048КБ L2 cache" також описує тип пам'яті. Це кеш пам'ять, яка забезпечує швидкий доступ процесора до основної пам'яті. Представлений на рекламі комп'ютер має два рівні кеш пам'яті. Кеш пам'ять першого рівня (LI) є швидкою пам'яттю малої ємності, яка вбудована в кристал процесора і дозволяє прискорити доступ до часто використовуваних даних і команд. Кеш пам'ять другого рівня (L2) є набором швидких мікросхем пам'яті, розміщених між процесором і основною пам'яттю. При цьому кеш пам'ять має ємність, що вимірюється в кілобайтах (КБ), тобто значно меншу, аніж ємність основної пам'яті. В даному випадку кеш пам'ять першого рівня розділена на кеш пам'ять даних і кеш пам'ять команд ємністю по 32 КБ, кеш пам'ять другого рівня має ємність 2048 КБ.

Рекламований комп'ютер має також зовнішню дискову пам'ять ємністю 2x100 ГБ, яка має швидкість обертання дисків 5400 обертів за хвилину. Крім того, ця пам'ять має буфер ємністю 8 МБ. Швидкість обертання є лише одним з визначальних факторів загальної продуктивності диску. Важливими є також принципи його з'єднання з іншими елементами комп'ютера, тобто його інтерфейс. В рекламованому комп'ютері інтерфейс диску має назву EIDE, що є скороченням від enhanced integrated drive electronics (покращена електроніка інтегрованого дисководу). Інтерфейс EIDE має спеціальні вузли, які дозволяють підвищити швидкість обміну.

Далі в рекламі розміщені характеристики пристроїв введення-виведення та комунікацій, які будуть детально розглянуті в розділі 10. Тут показано 3 USB порти, один швидкісний послідовний порт, один інфрачервоний порт, адаптери для безпровідного зв'язку та багато інших портів. Через ці порти інформація поступає в комп'ютер та з комп'ютера. USB (universal serial bus) порт є популярною зовнішньою шиною, яка підтримує режими Plug-and-Play (тобто можливість автоматичної конфігурації пристрою) та hot plugging (можливість підключити та відключити пристрій під час роботи комп'ютера).

Також в рекламі є інформація про монітор. Він має наступні характеристики: діагональ рівна 17 дюймів, формат 16:9, роздільна здатність WUXGA (1920 x 1200), технологія X-black, багатошарове антиблікове покриття. Характеристики монітора впливають на зручність взаємодії користувача з комп'ютером. Монітор з високою роздільною здатністю забезпечує краще бачення та якісну графіку.

На рекламі є і інша потрібна інформація про комп'ютер, зокрема наведені характеристики відео та аудіо систем, оптичного приводу та інше.

Розглядаючи вузли та характеристики представленого на рекламі комп'ютера, ми не обговорювали питання організації його роботи та вплив цих характеристик на ефективність вирішення задач, що буде зроблено далі в цій книзі.

Обговорюючи принципи роботи комп'ютера, ми будемо показувати його вузли та елементи схематично, не завжди вникаючи в питання їх фізичної реалізації. Але для того, щоб мати загальну уяву про конструкцію комп'ютера, подивимось як виглядають компоненти сучасного персонального комп'ютера (рис. 1.14).

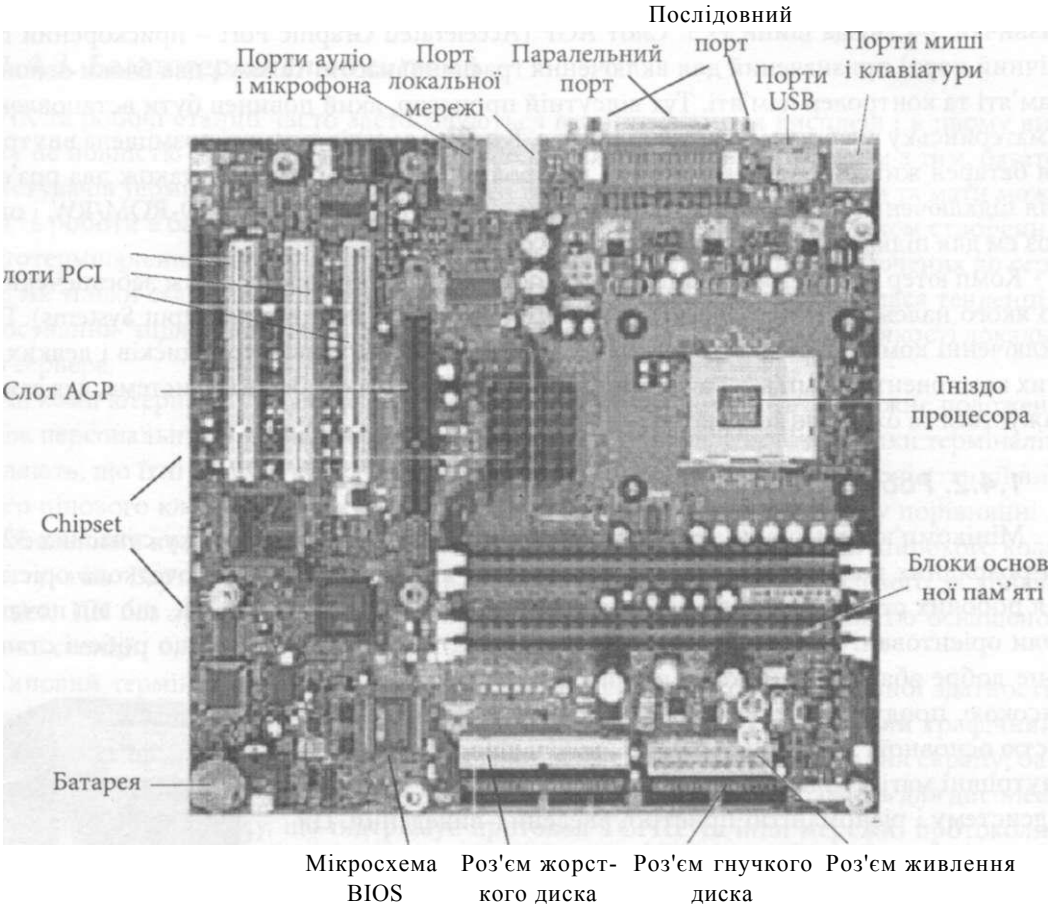


Рис. 1.14. Вигляд вузлів комп'ютера

Коли відкрити корпус персонального комп'ютера, можна побачити велику кількість різних елементів. Це, зокрема, блок живлення, який представляє собою металевий корпус з вбудованим вентилятором, різні типи дискової пам'яті, включаючи накопичувачі на жорсткому та гнучкому магнітних дисках, привід CD-ROM/RW чи DVD-ROM/RW. Всі компоненти комп'ютера, включаючи процесор та пам'ять, об'єднує системна плата. На рис. 1.14 показано системну плату Intel D850 з тлумаченням найважливіших компонентів.

Порти введення-виведення зверху плати забезпечують зв'язок комп'ютера з зовнішніми вузлами типу мікрофона та інших звукових пристроїв, миші, клавіатури, з

локальною обчислювальною мережею та іншими пристроями, які під'єднують через послідовний, паралельний та USB порти. Контролер введення-виведення, який входить до складу інших, крім процесора, пам'яті та відео, мікросхем системної плати (chipset), дозволяє всім з'єднаним пристроям функціонувати без конфліктів. Гнізда (слоти) для підключення до шини PCI (Peripheral Component Interconnect) дозволяють розширення плат, які належать до різних пристроїв, підключених до шини PCL Це 32-розрядна локальна шина для пересилання даних між процесором та зовнішніми пристроями (диски, відеоадаптер тощо) з швидкістю до 132 Мбіт/с. На материнській платі розташовують зазвичай 3-4 гнізда шини PCL Слот AGP (Accelerated Graphic Port - прискорений графічний порт) призначений для включення графічної карти. Також є два блоки основної пам'яті та контролер пам'яті. Тут відсутній процесор, який повинен бути встановлений в материнську плату, але є гніздо під нього. В нижньому лівому куті розміщена внутрішня батарея живлення для зберігання настройок BIOS. Ця плата має також два роз'єми для підключення жорсткого диска або приводу CD-ROM/RW чи DVD-ROM/RW, і один роз'єм для підключення гнучкого диска.

Комп'ютер продається з записаним в постійну пам'ять програмним забезпеченням, до якого належать POST (Power-On-Self-Test) та BIOS (Base Input/Output Systems). При включенні комп'ютера відбувається тестування пам'яті, клавіатури, дисків і деяких інших компонентів комп'ютера. Після цього викликається операційна система, яка завантажується та оживляє комп'ютер.

1.4.2. Робочі станції

Мінікомп'ютери стали прародичами й іншого напрямку розвитку сучасних 32 та 64-розрядних комп'ютерів, що сьогодні відомі як робочі станції. Початкова орієнтація робочих станцій на професійних користувачів (на відміну від ПК, що від початку були орієнтовані на споживача-непрофесіонала) призвела до того, що робочі станції - це добре збалансовані комп'ютерні системи, які, разом з високою продуктивністю, характеризуються великою ємністю основної і зовнішньої пам'яті, мають високошвидкісні внутрішні магістралі, високоякісну і швидкодіючу графічну підсистему і різноманітні пристрої введення-виведення. Ця властивість вигідно відрізняє робочі станції середнього і високого класу від ПК і сьогодні. Навіть найпотужніші ПК не в стані задовольнити зростаючі потреби інженерних задач через наявність в їхній архітектурі ряду вузьких місць.

В якості прикладу розглянемо характеристики робочої станції Sun Ultra 45 Workstation фірми Sun Microsystems, зовнішній вигляд якої наведено на рис. 1.15. До її складу входять два процесори UltraSPARC Ші з частотою 1.6GHz, кеш пам'ять другого рівня ємністю 1MB, Dual Gigabit Ethernet, до 16 GB DDR1 ECC пам'ять, два графічних прискорювачі для двовимірної та тривимірної графіки.

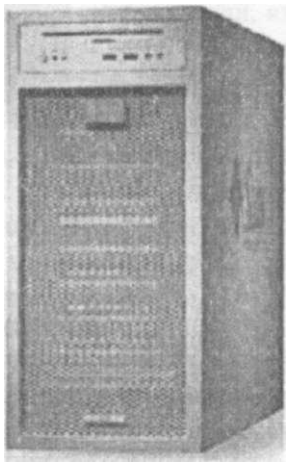


Рис. 1.15. Зовнішній вигляд робочої станції Sun Ultra 45 Workstation фірми Sun Microsystems

Виробники робочих станцій швидко відреагували на потребу в маловартісних моделях для ринку комерційних застосувань. Потреба в високій потужності на робочому столі навела компанії Sun Microsystems і Hewlett-Packard, які є лідерами на ринку робочих станцій, на ідею орієнтувати свою продукцію на комерційні застосування. І хоча значна частина систем цих компаній все ще орієнтована на технічні та наукові застосування, спостерігається безпрецедентне зростання продажу їх продукції для робіт комерційного застосування, що вимагають все більшої потужності для реалізації складних мережних прикладних систем, включаючи системи мультимедіа.

1.4.3. Багатотермінальні системи

ПК та робочі станції часто застосовуються в якості дорогих дисплеїв і в цьому випадку не повністю використовується їх обчислювальна потужність. Разом з тим, багато користувачів терміналів хотіли б покращити їхні графічні характеристики та мати можливість роботи в багатовіконній системі. Ці проблеми були вирішені шляхом створення багатотермінальних систем, які є набором стандартних терміналів, підключених до сервера. Як тільки стали доступними потужні графічні робочі станції, з'явилася тенденція застосування "підлеглих" терміналів, що використають робочу станцію в якості локального сервера.

На комп'ютерному ринку багатотермінальні системи займають проміжне положення між персональними комп'ютерами і робочими станціями. Постачальники терміналів заявляють, що їхні вироби ефективніші в вартісному вираженні, ніж робочі станції високого цінового класу, і пропонують збільшений рівень продуктивності у порівнянні з персональними комп'ютерами, що робить цю технологію доступною для широкого кола користувачів. Вартість терміналів складає біля половини вартості близького за конфігурацією ПК без зовнішньої пам'яті і приблизно чверть вартості повністю оснащеної робочої станції.

Типовий термінал включає наступні елементи: екран високої роздільної здатності; головний процесор, який підтримує двопроцесорну архітектуру; окремий графічний співпроцесор, що забезпечує швидше малювання на екрані і прокручування екрану; базові системні програми; програмне забезпечення сервера; локальну пам'ять для дисплею та мережного інтерфейсу, що підтримує протокол TCP/IP та інші мережні протоколи; порти для підключення клавіатури і миші.

Термінали відрізняються від ПК і робочих станцій не тільки тим, що не виконують функції звичайної локальної обробки. Робота терміналів залежить від головної системи, до якої вони підключені через мережу. Для того, щоб термінал міг працювати, користувачі повинні встановити програмне забезпечення багатовіконного сервера на головному процесорі, що виконує прикладну задачу. Локальна обчислювальна потужність терміналу зазвичай використовується для виконання програм обробки зображень, а не прикладних програм, які виконуються на головному процесорі. Термінал може відображати на одному і тому ж екрані декілька задач. Користувач може змінювати розміри вікон, їхнє місцезнаходження і маніпулювати ними в будь-якому місці екрана.

1.4.4. Сервери

Прикладні комерційні та бізнесові системи, розраховані на багато користувачів, включаючи системи керування базами даних і обробки транзакцій, великі видавничі системи, мережні системи і системи обслуговування комунікацій, системи розробки програмного забезпечення і обробки зображень, вимагають переходу до моделі обчислень "клієнт-сервер" і розподіленої обробки. В розподіленій моделі "клієнт-сервер" частину роботи виконує сервер, а частину - комп'ютер користувача (в загальному випадку частини сервера і користувача можуть виконуватись і на одному комп'ютері). Існує декілька типів серверів для різних застосувань: файловий сервер, сервер бази даних, принт-сервер, обчислювальний сервер, сервер застосувань. Таким чином, тип сервера визначається ресурсом, яким він володіє (файлова система, база даних, принтери, процесори або прикладні пакети програм).

З іншого боку існує класифікація серверів за масштабом мережі, в якій вони використовуються: сервер робочої групи, сервер відділу або сервер підприємства (корпоративний сервер). Ця класифікація надто умовна. Наприклад, розмір групи може змінюватися в діапазоні від декількох людей до декількох сотень людей, а сервер відділу може обслуговувати від 20 до 150 користувачів. Очевидно, залежно від числа користувачів і характеру вирішуваних ними завдань, вимоги до складу обладнання і програмного забезпечення сервера, до його надійності та продуктивності суттєво відрізняються.

Файлові сервери невеликих робочих груп (не більше 20-30 людей) простіше всього реалізуються на платформі персональних комп'ютерів і програмному забезпеченні Novell NetWare. Файл-сервер в даному випадку виконує роль центрального сховища даних. Типовими до складу невеликих файл-серверів входять процесор, основна та зовнішня пам'ять, а також адаптер Ethernet. До складу таких серверів часто включаються дисковод гнучких дисків і дисковод компакт-дисків. Графіка для більшості серверів несуттєва, тому достатньо мати звичайний монохромний монітор з невисокою роздільною здатністю. Бажано застосувати пристрій безперебійного живлення.

Для файл-серверів загального доступу, з якими водночас можуть працювати декілька десятків, а то і сотень людей, простої однопроцесорної платформи і програмного забезпечення Novell може виявитися недостатньо. В цьому випадку використовуються потужні багатопроцесорні сервери з можливостями нарощування основної пам'яті до декількох ГБ, дискового простору до сотень ГБ, швидкими інтерфейсами дискового обміну (типу Fast SCSI-2, Fast&Wide SCSI-2 і Fiber Channel) і декількома мережними інтерфейсами. Ці сервери використовують операційну систему UNIX, мережні протоколи TCP/IP і NFS. На базі багатопроцесорних UNIX-серверів зазвичай будуються також сервери баз даних великих інформаційних систем, бо на них покладається основне навантаження з обробки інформаційних запитів. Сервери подібного типу отримали назву суперсерверів.

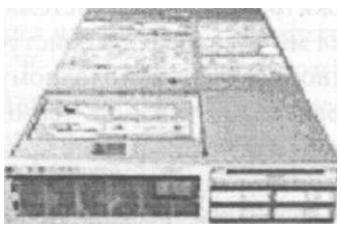


Рис. 1.16. Зовнішній вигляд сервера T2000 фірми Sun Microsystems

Розглянемо технічні характеристики одного з серверів фірми Sun, а саме сервера T2000, зовнішній вигляд якого наведено на рис. 1.16. Його ціна, залежно від комплектування, становить понад $8,419. До складу сервера

входять до восьми процесорів частотою 1.2 ГГц з архітектурою UltraSPARC TI, до 32 ГБ пам'ять, операційна система Solaris.

За загальносистемною продуктивністю, функціональними можливостями окремих компонентів, стійкістю до відмов, а також ступенем підтримки багатопроцесорної обробки, системного адміністрування і дискових масивів великої ємності суперсервери вийшли в нинішній час на один рівень з мейнфреймами. Сучасні суперсервери характеризуються наявністю двох або більше центральних процесорів, багатошинною структурою, мають достатні можливості нарощування дискового простору і обчислювальної потужності, засоби забезпечення надійності зберігання даних і захисту від несанкціонованого доступу.

1.4.5. Великі універсальні комп'ютерні системи

Великі універсальні комп'ютерні системи (мейнфрейми) і до сьогоднішнього дня залишаються найпотужнішими (не рахуючи суперкомп'ютерів) комп'ютерними системами загального призначення, що забезпечують безперервний цілодобовий режим експлуатації. Вони можуть включати один або декілька процесорів, кожний з яких, у свою чергу, може споряджатися векторними спеціалізованими процесорами (прискорювачами операцій з продуктивністю суперкомп'ютерів). Зазвичай мейнфрейми асоціюються з великими за габаритами комп'ютерами, що вимагають спеціально обладнаних приміщень із системами водяного охолоджування і кондиціювання. Однак прогрес в області елементної та конструкторської бази дозволив істотно зменшити габарити основних пристроїв. Поряд з надпотужними мейнфреймами, що вимагають організації двоконтурної водяної системи охолоджування, є менш потужні моделі, для охолоджування яких достатньо примусової повітряної вентиляції, та моделі, побудовані за блоково-модульним принципом, які не вимагають спеціальних приміщень і кондиціонерів.

Основними постачальниками мейнфреймів є відомі комп'ютерні компанії IBM, Amdahl (в даний час в складі концерну Fujitsu), ICL, Fujitsu Siemens Computers, Wincor Nixdorf AG, Siemens Business Services і деякі інші, але провідна роль належить безумовно компанії IBM. Саме архітектура системи IBM/360, випущеної в 1964 році, і її наступні покоління стали зразком для наслідування. Так, в СРСР протягом багатьох років випускалися машини ряду ЄС ЕОМ, що є аналогом цієї системи. В архітектурному плані мейнфрейми є багатопроцесорними системами, що містять один або декілька центральних і периферійних процесорів зі спільною пам'яттю, зв'язаних між собою високошвидкісними магістралями передачі даних. При цьому основне обчислювальне навантаження покладено на центральні процесори, а периферійні процесори (в термінології IBM - селекторні, блок-мультиплексні, мультиплексні канали і процесори телеопрацювання) забезпечують роботу з широкою номенклатурою периферійних пристроїв.

Спочатку мейнфрейми були орієнтовані на централізовану модель обчислень та працювали під керуванням патентованих операційних систем. Однак підвищений інтерес споживачів до відкритих систем, побудованих на базі міжнародних стандартів, змусив постачальників мейнфреймів істотно розширити можливості своїх операційних систем в напрямку сумісності.

Одними з найпотужніших на даний час є мейнфрейми сім'ї zSeries фірми IBM, зовнішній вигляд одного з блоків яких наведено на рис. 1.17. До основних особливостей

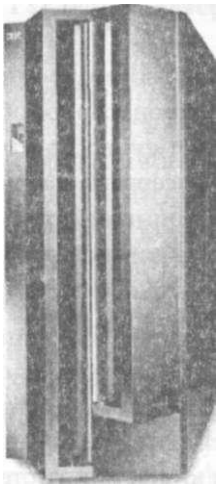


Рис. 1.17. Зовнішній вигляд одного блока мэйнфрейма сім'їzSeries фірми IBM

цих комп ютерів належать наступні: велика ємність зовнішньої пам'яті, адресація якої забезпечується 64-розрядним адресним словом, розміщення в кожному з 32 блоків до 54 центральних процесорів, причому кожен блок може бути рознесеним в просторі на віддаль до 100 кілометрів, які забезпечують продуктивність понад 18 мільярдів команд за секунду, підтримка роботи з операційними системами Linux, z/OS, z/VM, z/VSE, z/TPF, MUSIC/SP, потужні зовнішні інтерфейси.

Стрімкий ріст продуктивності персональних комп'ютерів, робочих станцій і серверів призвів до зниження потреби в мейнфреймдх. Однак цей процес останнім часом дещо уповільнився. Основною причиною відродження інтересу до мейнфреймів експерти вважають складність переходу до розподіленої архітектури клієнт-сервер, що виявилася вищою, ніж припускалося. Крім того, багато користувачів вважають, що розподілене середовище, на противагу мейнфреймам, не є достатньо надійним для найвідповідальніших застосувань.

Головним недоліком мейнфреймів у нинішній час залишається відносно низьке співвідношення продуктивність/вартість. Однак постачальники мейнфреймів докладають значних зусиль для поліпшення цього показника. Слід також пам'ятати, що в світі існує величезна інстальована база мейнфреймів, на якій працюють десятки тисяч прикладних програмних систем. Відмовитися від роками напрацьованого програмного забезпечення просто нерозумно. Тому в нинішній час очікується зростання об'ємів продажу мейнфреймів, які з одного боку, дозволять модернізувати існуючі системи, забезпечивши скорочення експлуатаційних видатків, а з іншого боку, створять нову базу для найвідповідальніших застосувань.

1.4.6. Кластерні комп'ютерні системи

Двома основними проблемами побудови комп'ютерних систем для критично важливих застосувань, зв'язаних з обробкою транзакцій, керуванням базами даних і обслуговуванням телекомунікацій, є забезпечення високої продуктивності та тривалого функціонування систем. Найефективнішим засобом для досягнення заданого рівня продуктивності є застосування паралельних архітектур, які піддаються масштабуванню. Завдання забезпечення тривалого функціонування системи має три складових: надійність, готовність і вартість обслуговування. Всі три складові передбачають, в першу чергу, боротьбу з несправностями системи, що породжуються відмовами і збоями в її роботі. Ця боротьба ведеться по всіх трьох напрямках, що взаємозв'язані і застосовуються спільно.

Підвищення надійності базується на принципі відвертання несправностей шляхом зниження інтенсивності відмов і збоїв за рахунок застосування електронних схем і компонентів з високим і надвисоким ступенем інтеграції, зниження рівня завад, полегшених режимів роботи схем, забезпечення теплових режимів їхньої роботи, а також за рахунок вдосконалення засобів монтажу апаратури. Підвищення рівня готовності передбачає

зниження в певних межах впливу відмов і збоїв на роботу системи з допомогою засобів контролю і корекції помилок, а також засобів автоматичного відновлення обчислювального процесу після прояву несправності, включаючи апаратну і програмну надлишковість, на основі якої реалізуються різноманітні варіанти стійкої до відмови архітектури. Підвищення готовності є засобом боротьби за зниження часу простою системи. Основні експлуатаційні характеристики системи істотно залежать від зручності її обслуговування, зокрема від ремонтопридатності, контролепридатності і т. д.

В останні роки в літературі з обчислювальної техніки все частіше вживається термін "системи високої готовності" (High Availability Systems). Всі типи систем високої готовності орієнтовані на мінімізацію часу простою. Є два типи часу простою комп'ютера: плановий і неплановий. Плановий час простою зазвичай включає час, прийнятий для проведення робіт по модернізації системи і для її обслуговування. Неплановий час простою є результатом відмови системи або її компоненти. Хоча системи високої готовності, можливо, більше асоціюються з мінімізацією непланових простоїв, вони виявляються також корисними для зменшення планового часу простою.

Існує декілька типів систем високої готовності, що відрізняються своїми функціональними можливостями і вартістю. Слід відзначити, що висока готовність не дається безкоштовно. Вартість систем високої готовності набагато перевищує вартість звичайних систем. Певно тому найбільше розповсюдження отримали кластерні системи завдяки їх здатності забезпечити достатньо високий рівень готовності при відносно низьких витратах. Термін "кластеризація" на сьогодні в комп'ютерній промисловості має багато різноманітних значень. Точне визначення могло б звучати так: "реалізація об'єднання машин, що представляється єдиним цілим для операційної системи, системного програмного забезпечення, прикладних програм і користувачів". Машини, кластеризовані таким способом, можуть при відмові одного процесора дуже швидко перерозподілити роботу на інші процесори всередині кластера. Це, можливо, найважливіше завдання багатьох постачальників систем високої готовності.

Першою концепцію кластерної системи анонсувала компанія DEC, визначивши її як групу об'єднаних між собою комп'ютерів, що представляють собою єдиний вузол обробки інформації. По суті кластер цієї компанії був слабко зв'язаною багатомашинною системою з спільною зовнішньою пам'яттю, що забезпечує єдиний механізм керування і адміністрування. На рис. 1.18 показано зовнішній вигляд такого кластера.

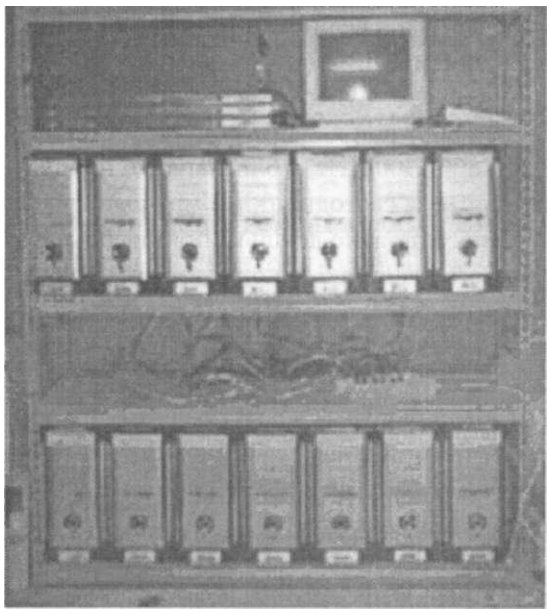


Рис. 1.18. *Зовнішній вигляд кластера*

Робота будь-якої кластерної системи визначається двома головними компонентами: високошвидкісним механізмом зв'язку процесорів між собою і системним програмним забезпеченням, що надає клієнтам прозорий доступ до системного сервісу.

В даний час широке розповсюдження отримала технологія паралельних баз даних. Ця технологія дозволяє великій кількості процесорів поділяти доступ до єдиної бази даних. Розподіл завдань між процесорними ресурсами і паралельне їх виконання дозволяє досягнути вищого рівня пропускної спроможності транзакцій, підтримувати більше число одночасно працюючих користувачів і прискорити виконання складних запитів. Для вирішення цих завдань використовується архітектура зі спільними (розподіленими) дисками. Це типовий випадок побудови кластерної системи. Ця архітектура підтримує єдину базу даних при роботі з декількома комп'ютерами, об'єднаними в кластер (зазвичай такі комп'ютери називаються вузлами кластера), кожний з яких працює під керуванням своєї копії операційної системи. В таких системах всі вузли поділяють доступ до загальних дисків, на яких власне і розміщується єдина база даних. Продуктивність таких систем може збільшуватися як шляхом нарощування числа процесорів і ємності основної пам'яті в кожному вузлі кластера, так і шляхом збільшення кількості самих вузлів. У випадку відмови одного з таких вузлів, вузли, що залишилися, можуть взяти на себе завдання, що виконувалися на вузлі, який відмовив, не зупиняючи загальний процес роботи з базою даних. Оскільки логічно в кожному вузлі системи є образ бази даних, доступ до неї буде забезпечуватися до тих пір, доки в системі є принаймні один справний вузол.

1.4.7. Суперкомп'ютери

До класу суперкомп'ютерів належать комп'ютери, що мають максимальну в даний час продуктивність, а також максимальну ємність основної та зовнішньої пам'яті. Вони асоціюються з великими розмірами, великими завданнями, гранично високими характеристиками. Швидкий розвиток комп'ютерної індустрії призводить до відносності даного поняття. Суперкомп'ютер десятирічної давності сьогодні під це визначення вже не потрапляє. Наприклад, продуктивність персональних комп'ютерів, що використовують Pentium-II/300MHz, є близькою до продуктивності суперкомп'ютерів середини 70-х років, проте за сьогоднішніми мірками суперкомп'ютерами не є ні ті, ні інші.

Нижче подано декілька прикладів, що показують основні характеристики комп'ютерів цього класу, які використовуються в даний час.

CRAY T932, векторно-конвеєрний комп'ютер фірми CRAY Research Inc. (на сьогодні це є підрозділ Silicon Graphics Inc.), уперше випущений у 1996 році. Максимальна продуктивність одного процесора дорівнює майже 2 млрд операцій за секунду, основна пам'ять нарощується до 8 ГБ, дисковий простір до 256000 ГБ (тобто 256Т6). Комп'ютер у максимальній конфігурації вміщує 32 процесори, що працюють із загальною пам'яттю, тому максимальна продуктивність всієї комп'ютерної системи складає більше 60 млрд операцій за секунду.

IBM SP2, матричний паралельний комп'ютер фірми IBM. Побудований на основі стандартних процесорів PowerPC 604e або POWER2 SC, сполучених між собою через високошвидкісний комутатор, причохму кожний має свою локальну основну пам'ять і

дискову підсистему. Характеристики цих процесорів відомі й особливого подиву не викликають, проте в рамках однієї системи SP2 їх може бути об'єднано дуже багато. Зокрема, максимальна система, встановлена в Pacific Northwest National Laboratory (Richland, USA), вміщує 512 процесорів. Виходячи з числа процесорів, можна уявити сумарну потужність всієї системи.

HP Exemplar, комп'ютер із кластерною архітектурою від Hewlett-Packard Inc. Зокрема, модель V2250 (клас V) побудована на основі мікропроцесора PA-8200, що працює з тактовою частотою 240 МГц. В рамках одного вузла зі спільною основною пам'яттю до 16 ГБ можна об'єднати до 16 процесорів. У свою чергу вузли в рамках однієї комп'ютерної системи з'єднуються між собою через високошвидкісні канали передачі даних.

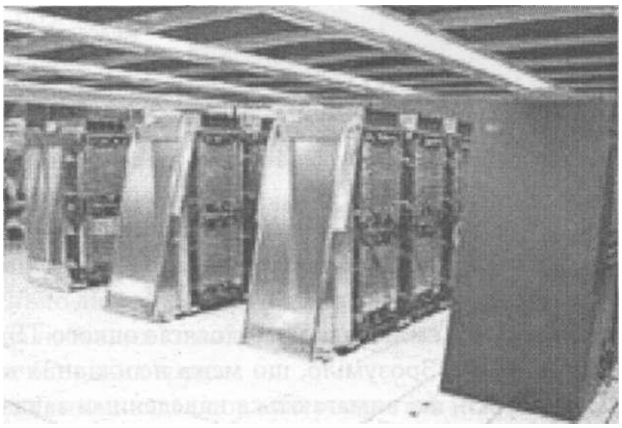


Рис. 1.19. Зовнішній вигляд суперкомп'ютера Blue Gene/L фірми IBM

Суперкомп'ютер ASCI RED, результат виконання програми Accelerated Strategic Computing Initiative. Побудований на замовлення Міністерства енергетики США, він об'єднує 9152 процесори Pentium Pro, має 600 ГБ сумарної основної пам'яті та загальну продуктивність 1800 мільярдів операцій за секунду.

Найпотужнішим на сьогодні комп'ютером є суперкомп'ютер фірми IBM Blue Gene/L (рис. 1.19), який має 131 072 процесорних вузлів та продуктивність 280.6 TFLOPS (10^{12} FLOPS). Кожен вузол містить процесор PowerPC 440 із 512 МБ локальної пам'яті.

В 2006 році був уведений в експлуатацію суперкомп'ютер MDGRAPE-3, який досяг продуктивності 1 PFLOPS (10^{15} FLOPS), однак його не відносять до універсальних суперкомп'ютерів, оскільки він є орієнтованим на виконання задач молекулярної динаміки.

Навіть спрощені конфігурації таких комп'ютерів коштують не один мільйон доларів США. Виникає ряд природних запитанньː

> які завдання настільки важливі, що потребують використання комп'ютерів вартістю декілька мільйонів доларів?

> які завдання настільки складні, що процесора Pentium IV недостатньо?

От лише невеличкий список областей людської діяльності, де необхідно використовувати суперкомп'ютери: автомобілебудування; нафто- і газовидобуток; фармакологія; прогноз погоди і моделювання зміни клімату; сейсморозвідка; проектування електронних пристроїв; синтез нових матеріалів, генні дослідження.

На рис. 1.20 подано завдання, для виконання яких необхідне застосування суперкомп'ютерів, а також потрібні для їх вирішення комп'ютерні ресурси.

Рис. 1.20. Завдання та комп'ютерні ресурси

Видно, що ємність пам'яті досягає одного ТБ за умови, що продуктивність має бути один TFLOPS. Зрозуміло, що межа необхідних комп'ютерних ресурсів є рухомою. Надати ресурси, які вимагаються наведеними завданнями, за допомогою стандартних однопроцесорних систем неможливо. Це спричинює використання багатопроцесорних комп'ютерних систем як магістрального напрямку досягнення високої продуктивності.

1.4.8. *Мікроконтролери*

Мікроконтролери - комп'ютери на кристалі, призначені для керування електронними пристроями, зокрема побутовими пристроями, виробничими лініями, вимірювальними пристроями і т. д. До складу мікроконтролера входять наступні вузли:

> центральний процесор, розрядністю від 4 до 64 бітів, залежно від потрібної точності обчислень;

> інтерфейси введення-виведення, в першу чергу послідовні порти;

> периферійні пристрої, такі як: таймери та схеми захисту, цифроаналогові та аналогоцифрові перетворювачі;

> пам'ять з довільним доступом для зберігання даних;

> постійна пам'ять типу ROM, EPROM, EEPROM чи Flash для зберігання програми;

> генератор тактів.

Така інтеграція названих пристроїв на кристалі дозволяє забезпечити малі габарити та споживання і сприяє широкому використанню мікроконтролерів у різного роду вбудованих системах. Наприклад, в сучасному автомобілі використовується понад 50 мікроконтролерів. Вони також використовуються в побутовій електроніці, мобільних телефонах, виробничих лініях тощо. На рис. 1.21 подано зовнішній вигляд мікроконтролера PIC 18F8720 фірми Microchip в корпусі TQFP з 80 виводами.

Розробники мікроконтролерів забезпечують спеціальний сервіс для користувачів, зокрема версії, які дозволяють перепрограмування програмної пам'яті ультрафіолето-

вим світлом, можливість підключення зовнішньої оперативної пам'яті в якості пам'яті програм, та інше. Сучасні мікроконтролери програмуються в коді мови С та мають внутрішні схеми відлагодження.

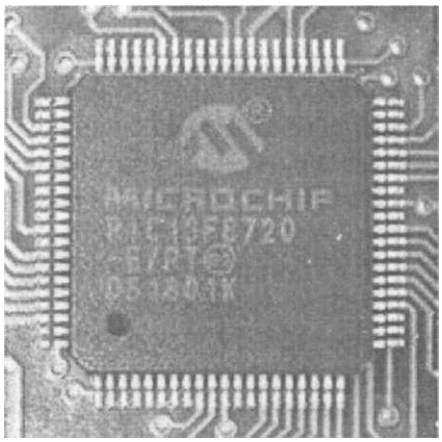


Рис. 1.21. Зовнішній вигляд мікроконтролера PIC 18F8720 фірми Microchip

1.4.9. Спеціалізовані комп'ютери

За допомогою універсальних комп'ютерів та комп'ютерних систем (УКС), які були розглянуті вище, можна вирішувати багато задач наукового, виробничо-технічного та іншого характеру. Однак існують надзвичайно важливі класи задач і окремі задачі, для розв'язку яких математичні та технікоекономічні якості УКС недостатні. Не варто доводити дієвість принципу спеціалізації інструментальних засобів взагалі, оскільки вся свідома технічна діяльність людства її підтверджує. Досить вказати, що цей принцип ефективно діє і в галузі інформатики. Загальний аналіз причин створення і використання спеціалізованих комп'ютерних систем (СКС) показує, що ці причини можна віднести до трьох основних груп.

Перша група об'єднує причини, що виникли внаслідок суперечностей між формальними математичними методами постановки і розв'язку задач, з одного боку, і загальними принципами організації та функціонування, а також технічними можливостями УКС, з іншого боку. Саме математична сутність задач часто обумовлює необхідність створення СКС для їх розв'язку. Як приклади тут можна навести нові нестандартні та неалгоритмічні методи, системи алгебраїчних, диференційних та інтегральних рівнянь великої розмірності, логічні та імовірнісно-статистичні задачі, дії над матрицями та векторами, задачі в багатовимірних просторах та багато інших.

До другої групи входять причини, які обумовлені змістовною стороною задач, вирішуваних СКС, та відображають специфіку відповідних предметних областей.

Третя група причин обумовлена особливими вимогами до якості реалізації комп'ютерних систем, які зазвичай полягають в екстремалізації (тобто в максимальному наближенні до теоретичних границь) деяких їх характеристик, наприклад, продуктивності, надійності (безвідмовності, живучості, відновлюваності, довговічності та ін.), вартості, точності, габаритів, маси і т.п. Сюди ж належать вимоги, що визначають такі якості комп'ютерних систем, як їх повна або часткова імплантація (конструктивне та функціональне суміщення) в інші системи, інформаційне поєднання з ними, пристосованість до умов експлуатації та кваліфікації обслуговуючого персоналу і т.д.

Слід мати на увазі, що реальні ситуації створення СКС найповніше характеризуються двома особливостями. Перша полягає в тому, що саме СКС є своєрідним засобом апробації нових методів автоматизації обробки інформації, що мають математичні корені. Наприклад, розпаралелювання та децентралізація обчислень, макрооперації та функціональні розширювачі, символьна обробка та розв'язок задач в багатовимірних числових системах та ін. пройшли спочатку дуже ретельну перевірку в СКС і тільки після цього з'явилися в УКС.

Друга особливість пов'язана з тшм, що реальні СКС є складними програмно-технічними комплексами, в яких на інженерному рівні необхідно задовольнити багато суперечливих вимог. Тому досягнення оптимальних і функціональних якостей СКС може бути проблематичним і доцільніше визначати ці якості як оптимізовані, тобто такі, що тією чи іншою мірою наближаються до оптимальних. Аналіз математичних методів оптимізації СКС показує, що вони дозволяють, певною мірою, виявляти недоліки таких систем, 'їхні "слабкі місця", простежити взаємозв'язок характеристик системи, визначити загальний напрямок підвищення їх ефективності та оцінити різні варіанти СКС. Однак ці методи не дають ніяких конструктивних рішень і шляхів удосконалення СКС, не визначають змістовної сторони різних варіантів їх організації та реалізації. Генезис таких варіантів формальними математичними методами неможливий. Тому процес створення оптимізованих СКС має характер багатоступеневої ітераційної процедури, де в різних відношеннях комбінуються формальні та конкретно-змістовні методи, що відіграють аналітичну (оціночну) та синтетичну (генеративну) ролі.

Таким чином, СКС - це комп'ютерні системи для розв'язку великого числа відносно вузьких класів задач, оптимізовані в певній критеріальній сукупності.

Для СКС характерні наступні риси, які відрізняють їх від універсальних комп'ютерних систем:

- орієнтація структури на вирішувані задачі;
- вузький, в основному постійний клас вирішуваних задач;
- особливі вимоги до точності, часто нестандартна довжина розрядної сітки;
- спеціальна система обміну, в тому числі наявність аналого-цифрових та цифро-аналогових каналів зв'язку;
- використання орієнтованих на область застосування мов програмування та широкі можливості їх апаратної інтерпретації;
- наявність спеціальних функцій і процедур в наборі операцій та команд;
- необхідність обробки вхідних даних в темпі їх поступлення та видачі результатів обчислень в темпі поступлення вхідних даних;
- суміщення в часі приймання, обробки та видачі даних;
- висока продуктивність;
- малі габарити;
- низька споживана потужність;
- орієнтація конструкції на конкретне застосування.

1.5. Предмет та порядок розгляду матеріалу даної книги

Комп'ютер є складною системою. Сучасні комп'ютери складаються з мільйонів або й мільярдів елементарних електронних компонент. Для зрозумілого опису такої складної системи використовується її ієрархічна природа, тобто комп'ютер розглядається як набір взаємозв'язаних підсистем, кожна з яких також є ієрархічною (рис. 1.22). Найнижчим рівнем є елементарна підсистема. На кожному рівні складна система включає набір відповідним чином з'єднаних компонент, які визначають її поведінку на даному рівні. Кожен рівень може бути описаним структурою та функціями компонент. Структура включає перелік компонент та організацію їх взаємозв'язків. Функції описують операції кожної індивідуальної компоненти.

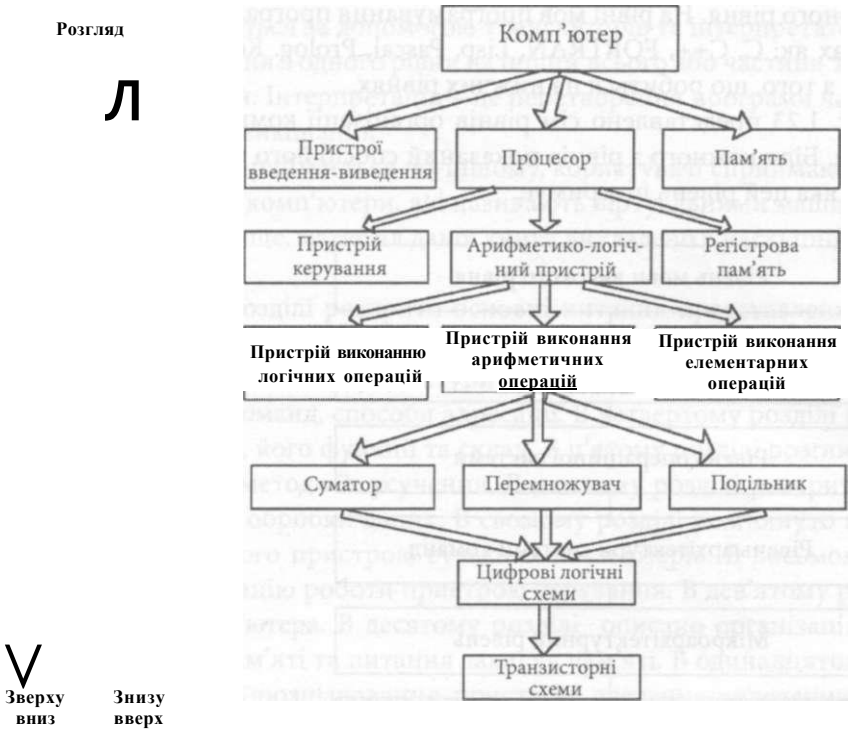


Рис. 1.22. Ієрархічна природа комп'ютера

Опис може виконуватись як знизу догори, починаючи з елементарних електронних компонентів і закінчуючи повним описом системи, так і згори донизу, шляхом поділу системи на складові частини. Перший підхід використовується при проектуванні комп'ютерів та їх функціональних вузлів, зокрема при їх ієрархічному описі мовами опису апаратних засобів типу УНОЬ та Уегііод. Для розгляду та вивчення комп'ютера другий підхід є ефективнішим, тому в подальшому будемо в основному користуватися цим підходом.

Комп'ютер має багаторівневу організацію. Найнижчим є рівень фізичних пристроїв, об'єктами якого є транзистори. Він є основою для цифрового логічного рівня, об'єктами якого є вентилі, побудовані на декількох транзисторах. Вентиль виконує прості логічні функції, такі як: І, АБО. На основі цифрового логічного рівня формується рівень міжрегістрових передач або мікроархітектурний рівень, об'єктами якого є регістри, мультиплексори, лічильники, стеки регістрів, блоки постійної та оперативної пам'яті, суматори, арифметико-логічні пристрої і т. д. Цей рівень можна поділити на підрівні, оскільки кожен з перерахованих елементів може бути представленим кількома рівнями простіших елементів. Наступним є архітектурний рівень, або рівень системи команд, який також має підрівні. Далі йде макроархітектурний рівень, або рівень операційної системи. Цей рівень є відповідальним за мультипрограмування, захист пам'яті, синхронізацію процесів та багато інших важливих функцій. І, нарешті, асемблерний рівень та рівні мов програмування. Кожна команда асемблерного рівня транслюється у відповідну їй команду

архітектурного рівня. На рівні мов програмування програміст працює з комп'ютером на таких мовах як: C, C++, FORTRAN, Lisp, Pascal, Prolog. Користувач цього рівня бачить дуже мало з того, що робиться на нижчих рівнях.

На рис. 1.23 представлено сім рівнів організації комп'ютера, запропоновані Е. Таненбаумом. Біля кожного з рівнів показаний спосіб його підтримки, а в дужках - назва програми, яка цей рівень підтримує.

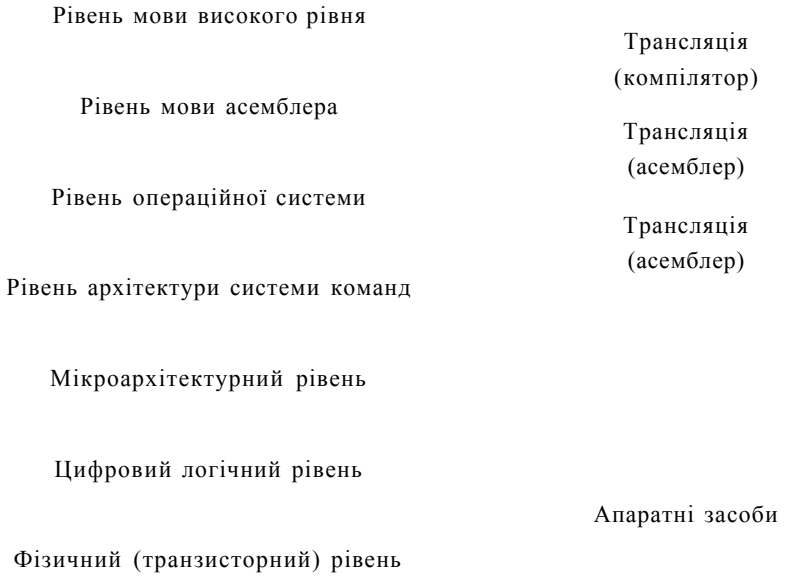


Рис. 1.23. Сім рівнів організації комп'ютера

Ми не будемо розглядати фізичний та цифровий логічний рівні, які є предметом розгляду комп'ютерної електроніки, прикладної теорії цифрових автоматів та комп'ютерної схемотехніки, та які читачеві потрібно знати, приступаючи до вивчення предмета архітектури комп'ютера. В даній книзі розглянемо рівень архітектури системи команд та частково знизу мікроархітектурний рівень, який з позиції функціонування та побудови елементів комп'ютера розглядається в курсі комп'ютерної схемотехніки, і зверху рівень операційної системи. З рівня архітектури системи команд програміст може працювати з комп'ютером, причому, на цьому рівні найбільше враховуються особливості побудови комп'ютера.

Разом з тим, для роботи з комп'ютером на рівні архітектури системи команд потрібно писати програму на машинній мові, тобто в командах, які є двійковими кодами, що є досить складним навіть для вирішення простих задач. Для спрощення написання машинних команд була введена асемблерна мова, коли команди представляються в мнемонічній формі. Але це не спростило взаємодію людини з комп'ютером. З метою скорочення розриву між мовою людини та машини були введені мови програмування високого рівня. Рівні мов програмування орієнтовані на специфіку використання комп'ютера. Оскільки комп'ютер розпізнає команди починаючи з архітектурного рівня, перехід з ви-

щих рівнів на нижчі здійснюється за допомогою трансляторів та інтерпретаторів. Трансляція передбачає перетворення з одного рівня на інший всього або частини тексту програми до початку її виконання. Інтерпретація - це перетворення програми частинами в реальному часі, тобто під час виконання.

Оскільки кожен з рівнів сприймається по-іншому, користувачі сприймають комп'ютери кожного рівня як окремі комп'ютери, які називають віртуальними машинами.

Виходячи з окресленого вище, матеріал даної книги викладено в наступній послідовності.

В наступному, другому, розділі розкрито основні питання представлення даних в комп'ютері, які є важливими для розуміння матеріалу наступних розділів. В третьому розділі розглянуто основні елементи архітектури комп'ютера, які включають організацію пам'яті, формати і типи команд, способи адресації. В четвертому розділі визначено місце процесора в комп'ютері, його функції та склад. В п'ятому розділі розглянуто конфлікти в конвеєрі команд та методи їх усунення. В шостому розділі розкрито основні питання виконання операцій обробки даних. В сьомому розділі розглянуто принципи побудови арифметико-логічного пристрою сучасних комп'ютерів. В восьмому розділі описано структуру та організацію роботи пристрою керування. В дев'ятому розділі подано структуру пам'яті комп'ютера. В десятому розділі описано організацію взаємодії між рівнями ієрархічної пам'яті та питання захисту пам'яті. В одинадцятому розділі наведено пояснення способів розпізнавання пристроїв введення-виведення, описано функції інтерфейсної схеми пристроїв введення-виведення та наведено методи керування введенням-виведенням. У дванадцятому розділі розглянуто питання подальшого підвищення продуктивності комп'ютерів шляхом створення паралельних комп'ютерних систем.

1.6. Підсумок розділу

У цьому розділі представлено короткий огляд організації й архітектури комп'ютерної системи. Пояснено деяку термінологію в контексті комп'ютерної реклами. Описано основні функціональні вузли комп'ютера, їх функції та характеристики, а також тенденції зміни цих характеристик. Багато чого з цієї термінології буде розширено в наступних розділах. Історично комп'ютери були досить простими, але з часом вони ускладнилися, що викликало необхідність проводити їх розгляд як ієрархію рівнів замість складної системи. Кожен рівень в цій ієрархії слугує певній меті, і всі рівні допомагають мінімізувати семантичний проміжок між мовою програмування високого рівня або застосуванням та вентилями і провідниками, які складають фізичні технічні засоби. Описано виконувані завдання та типи комп'ютерів. Можливо, найважливішим питанням цього розділу є введення поняття архітектури машини фон Неймана. Хоча є інші архітектурні моделі, архітектура фон Неймана є домінуючою в сьогоднішніх універсальних комп'ютерах.

1.7. Література для подальшого читання

Основні поняття, функції й основні функціональні вузли комп'ютера та їх взаємозв'язок, а також загальна організація роботи комп'ютера детально розглянуто в літературі [1-4]. Тут же, а також в роботах [5-11], проведено детальний огляд історії створення та

розвитку комп'ютерів. Окремо потрібно сказати про роботу [5], в якій подано історичний опис процесу створення комп'ютерів провідними науковими та промисловими підприємствами колишнього Радянського Союзу, в тому числі показано суттєві досягнення в цій галузі й українських науковців та інженерів. В роботі [12] наведено підтвердження дієвості закону Мура, а в роботі [13] описано багаторівневу організацію комп'ютера.

1.8. Література до розділу 1

1. Burks, A. W., H. H. Goldstine, and J. von Neumann [1946]. "Preliminary discussion of the logical design of an electronic computing instrument," Report to the U.S. Army.

2. Amdahl, G. M. [1967]. "Validity of the single processor approach to achieving large scale computing capabilities," Proc. AFIPS 1967 Spring Joint Computer Conf. 30 (April), Atlantic City, N.J., 483-485.

3. Wilkes, M. V., D. J. Wheeler, and S. Gill [1951]. The Preparation of Programs for an Electronic Digital Computer, Addison-Wesley, Cambridge, Mass.

4. Augarten, Stan. Bit by Bit: An Illustrated History of Computers. London: Unwin Paperbacks, 1985.

5. Малиновский Б. М. История вычислительной техники в лицах. KIT, Киев, 1995.

6. Blaauw, G., & Brooks, F. Computer Architecture: Concepts and Evolution. Reading, MA: Addison-Wesley, 1997.

7. Ceruzzi, Paul E. A History of Modern Computing. Cambridge, MA: MIT Press, 1998.

8. Cortada, J. W. Historical Dictionary of Data Processing, Volume 1: Biographies; Volume 2: Organization, Volume 3: Technology. Westport, CT: Greenwood Press, 1987.

9. McCartney, Scott. ENIAC: The Triumphs and Tragedies of the World's First Computer. New York: Walker and Company, 1999.

10. Möllenhoff, Clark R. Atanasoff: The Forgotten Father of the Computer. Ames, IA: Iowa State University Press, 1988.

11. Polachek, Harry. "Before the ENIAC." IEEE Annals of the History of Computing 19: 2 (June 1997), pp. 25-30.

12. Schaller, R. "Moore's Law: Past, Present, and Future." IEEE Spectrum, June 1997, pp. 52-59.

13. Tanenbaum, A. Structured Computer Organization, 4th ed. Upper Saddle River, NJ: Prentice Hall, 1999.

1.9. Питання до розділу 1

1. Що таке комп'ютер?
2. Дайте визначення алгоритму.
3. Дайте хронологію появи перших комп'ютерів.
4. Які основні функції комп'ютера? Які основні функціональні вузли комп'ютера? їх завдання та основні характеристики.
5. Назвіть три базових компоненти кожного комп'ютера.
6. Назвіть три типи пам'яті комп'ютера.
7. Який степінь 10 означає префікс Гіга? Якому степеню двійки він еквівалентний?
8. Який степінь 10 означає префікс мікро? Якому степеню двійки він еквівалентний?
9. Яка одиниця використовується для вимірювання тактової частоти комп'ютера?
10. Дайте пояснення суті закону Мура.
11. Як змінюється з часом тактова частота роботи процесора?
12. На скільки щороку зростає продуктивність комп'ютерів?

13. Як змінюється з часом ємність та час зчитування-запису динамічної напівпровідникової пам'яті?

14. Як змінюється з часом ємність зовнішньої дискової пам'яті?

15. Які одиниці використовуються для оцінки продуктивності комп'ютера?

16. Що дає використання тестових програм для оцінки продуктивності комп'ютера?

17. Які існують тестові програми для оцінки продуктивності комп'ютера?

18. Які є основні варіанти зв'язку між функціональними вузлами комп'ютера? Які їх відмінності?

19. Порівняйте двошинні структури комп'ютера з обміном через процесор та через пам'ять.

20. Як узгодити передачу даних між повільними і швидкими вузлами комп'ютера?

21. Поясніть роботу комп'ютера з одношинною структурою.

22. Що таке архітектура комп'ютера?

23. Що таке структура комп'ютера?

24. Що таке архітектура системи команд?

25. Яка різниця між архітектурою та структурою комп'ютера?

26. За якими ознаками здійснюється класифікація комп'ютерів?

27. Назвіть особливості архітектури Джона фон Неймана.

28. Які існують ненейманівські архітектури комп'ютерів? Які їх відмінні риси?

29. Назвіть два основних напрями використання комп'ютерів.

30. Що розуміється під паралельною обробкою?

31. Які є типи комп'ютерів?

32. Персональні комп'ютери - особливості та сфери застосування.

33. Як сконструйовано сучасний персональний комп'ютер?

34. Робочі станції - особливості та сфери застосування.

35. Багатотермінальні системи - особливості та сфери застосування.

36. Сервери - особливості та сфери застосування.

37. Великі універсальні комп'ютерні системи - особливості та сфери застосування.

38. Кластерні архітектури - особливості та сфери застосування.

39. Суперкомп'ютери - особливості та сфери застосування.

40. Мікроконтролери - особливості та сфери застосування.

41. Спеціалізовані комп'ютери - особливості та сфери застосування.

42. Поясніть ієрархічну природу комп'ютера.

43. Назвіть сім рівнів організації комп'ютера.

Розділ 2

\^ктт/ютері

У сучасних комп'ютерах використовують різні форми та формати представлення даних, якими є числа та закодовані символи. Це дозволяє вибирати ті із них, що найбільшою мірою відповідають вимогам розв'язуваних задач. Тип використовуваних форм та форматів представлення даних суттєво впливає на характеристики комп'ютера.

У цьому розділі висвітлені основні питання представлення даних у комп'ютері, які є важливими для розуміння матеріалу наступних розділів. Описані позиційні системи числення та принципи подання даних у двійковому, вісімковому і шістнадцятковому кодах. Подані правила переведення чисел із системи числення з довільною основою до десяткової, а також переведення чисел із десяткової до системи числення з іншою основою. Розглянуті засади подання чисел зі знаком у прямому, оберненому та доповняльному кодах. Проведено аналіз особливостей подання даних у форматах із фіксованою та з рухомою комою, включаючи подання даних за стандартом IEEE-754. Зважаючи на важливість, розглянуто питання кодування алфавітно-цифрової інформації кодами ASCII, EBCDIC та Unicode.

2.1. Позиційні системи числення

Система числення - це спосіб подання довільного числа за допомогою алфавіту символів, які називають цифрами. Є різні системи числення. Від їх особливостей залежить наочність відображення чисел та складність виконання операцій над числами. Прикладом системи числення з дуже складним способом запису чисел і громіздкими правилами виконання арифметичних операцій є римська система числення.

Якщо в послідовності цифр, які зображають число, має значення позиція цифри, то систему числення називають позиційною. Такі системи числення характеризуються наочністю відображення чисел та простим виконанням арифметичних операцій. У позиційних системах числення при безпосередньому представленні цифр число записується у вигляді:

$$X = x\,x_{з\ в-1}\,,\ldots x,x_{1\ (г\ -1}\,,\ldots x_{-т}\ .$$

Кома у цій послідовності відділяє цілу частину числа від дробової. Позиції цифр, які рахуються від коми, називають розрядами. Кількісний еквівалент, що виражається цим записом, визначається так:

$$X = к^{5}x+к^{5}{}''{}'x_{5}\ +\ldots+к^{1}x_{1},+к^{\circ}x_{0}+к'x_{-1}\ ,+\ldots+к''^{т}x_{-т}\ ,$$

Де:

- к - основа системи числення, тобто кількість різних цифр, які використовуються в позиційній системі числення,

- 8+1 - розрядність цілої частини числа,
- т - розрядність дробової частини числа,
- " х. - цифри і-го розряду запису числа (х. = 0, 1 , к - 1),
- к[1] - вага і-го розряду.

У цьому випадку вага і-го розряду в к разів більша за вагу (і-I)-го розряду. Такі системи числення називають системами з природним порядком ваги. До них належать двійкова, вісімкова, десяткова і шістнадцяткова системи числення.

У звичній для нас десятковій системі числення довільне число подається цифрами від 0 до 9; при цьому має значення позиція цифри. Число в десятковій системі записується у вигляді:

а значення числа обчислюється за таким виразом:

$$B = -IO''^{-1} + Ю_{,,} \cdot 10^{\wedge p} + \ldots + О_{I} \cdot 10 + 0\text{-}10^{\circ} + ГУ_{i} \cdot 10г^{1} + ТУ_{2} \cdot 10г^{2} + \ldots + ГУ_{M} \cdot 10^{n},$$

де:

- N - кількість цифр (розрядів) у цілій частині числа (зліва від коми),
- М - кількість розрядів у дробовій частині числа (справа від коми),
- Б. - значення і-го розряду (розряди цілої частини),
- Б'. - значення і-го розряду (розряди дробової частини),
- Б - значення числа.

Звичайно, що дробової або цілої частини числа може і не бути (Ы або М = 0).

2.2. *Двійкові, вісімкові та шістнадцяткові числа*

У зв'язку з тим, що елементи з двома станами використовуються як базові елементи комп'ютерної техніки, всі числа в комп'ютерах представляються у двійковій системі числення. Розглянемо особливості цієї системи.

Двійкова система числення будується за тим самим правилом, що і десяткова, але в ній використовуються лише дві цифри - 0 та 1. Число у двійковій системі числення записується у вигляді:

$$в = в_{,,\_,} в_{,,\_^{2}} \ldots в_{1} в^{\wedge} в'_{1} в'_{2} \ldots в'_{м},$$

а значення числа обчислюється за таким виразом:

$$B = B^{\wedge} \cdot 2''^{-1} + B_{н\_;} \cdot 2''^{\sim 2} + \ldots + B_{г} \cdot 2' + B_{0} \cdot 2^{\circ} + B\backslash \cdot 2''' + B^{\wedge} \cdot 2^{\sim 2} + \ldots + B'_{м} \cdot 2^{\sim м},$$

де:

N - кількість двійкових цифр (розрядів) у цілій частині числа,

М - кількість двійкових розрядів у дробовій частині числа,

В. - значення і-го розряду цілої частини числа,

В'. - значення і-го розряду дробової частини числа,

В - значення числа.

Приклади двійкових чисел:

$1011010{,}01_{2} = 1.2^{6} + 0 \cdot 2^{5} + 1 \cdot 2^{4} + 1 \cdot 2^{3} + 0\text{-}2^{2} + 1 \cdot 2^{1} + 0\text{-}2^{\circ} + 0 \cdot 2^{1} + 1 \cdot 2^{\sim 2} =$

$= 64 + 16 + 8 + 2 + 0{,}25 = 90{,}25_{10}$

$101{,}01101_{2} = 1 \cdot 2^{2} + 1 \cdot 2^{\circ} + 1 \cdot 2^{-2} + 1 \cdot 2''^{3} + 1 \cdot 2''^{5} = 5{,}040625_{10}$

$1100110{,}\ 11_{2} = 102{,}75_{ш}$

Часто у розробника, а то і в користувача комп'ютера, виникає потреба в перевірці коректності виконання операцій над двійковими числами комп'ютером або його вузлом.

А оскільки в комп'ютерах опрацьовуються багаторозрядні двійкові числа, і оперувати з такими довгими послідовностями нулів та одиниць (наприклад, рядок із 32 цифр) незручно, то набули поширення вісімкова та шістнадцяткова системи числення. У вісімковій системі числення використовують вісім цифр від 0 до 7, а у шістнадцятковій системі числення крім десяткових цифр від 0 до 9 використовують 6 літер латинського алфавіту (А, В, С, Б, Е, Б) для позначення цифр від 10 до 15. Значення числа обчислюється за таким виразом:

$$H = H_{N-1} \cdot 104 + H_{N-2} \cdot Iб^{*^{E}} + \ldots + H_{1} \cdot 16^{1} + H_{0} \cdot 16^{\circ} + H'_{1} \cdot 16^{-1} + H_{2} \cdot 16^{E} + \ldots + H'_{M} \cdot 16^{\prime},$$

де:

- N - кількість цифр (розрядів) у цілій частині числа (зліва від коми),
- M - кількість розрядів у дробовій частині числа (справа від коми),
- Н. - значення і-го розряду (розряди цілої частини),
- Н/ - значення і-го розряду (розряди дробової частини),
- Н - значення числа.

Особливістю цих систем є зручний перехід до двійкової системи та навпаки. Три двійкових розряди переводяться в один вісімковий, а чотири двійкових розряди - в один шістнадцятковий, як показано в табл. 2.1.

Таблиця 2.1

Двійкова	Шістнадцяткова	Двійкова	Шістнадцяткова
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
ООП	3	1011	B
0100	4	1100	C
0101	5	1101	
0110	6	1110	E
0111	7	1111	P

Наприклад, двійкове число 01101101 у шістнадцятковій системі записуватиметься як 6Б. Для переведення чисел із шістнадцяткової та вісімкової систем числення у двійкову необхідно кожну цифру числа, яке переводиться, замінити відповідно чотири- або трирозрядним двійковим еквівалентом - тетрадою або тріадою, а отримані двійкові цифри розташувати на місцях шістнадцяткових або вісімкових цифр.

У разі необхідності переведення чисел із десяткової системи числення у вісімкову, шістнадцяткову та двійкову переведення робиться лише в одну систему (вісімкову або шістнадцяткову). Подальше переведення виконується через двійкову систему, використовуючи тріади та тетради.

Приклад 1. Переведемо нисло 12345,67 з десяткової системи числення у двійкову, вісімкову, шістнадцяткову.

1. Переведення цілої частини числа у вісімкову систему:

12345 : 8 = 1543, залишок 1;

1543 : 8 = 192, залишок 7;

192 : 8 = 24, залишок 0;

24 : 8 = 3, залишок 0;

3:8 = 0, залишок 3.

Результат: 30071.

2. Переведення дробової частини числа у вісімкову систему:

0,67x8 = 5,36;

0,36x8 = 2,88;

0,88 x 8 = 7,04;

0,04x8 = 0,32.

Наближений результат: 0,5270....

3. Отримання повного результату шляхом об'єднання результатів, отриманих в п. 1 та п. 2. Результат: 30071,5270....

4. Переведення результату у двійкову та шістнадцяткову системи числення (табл. 2.2). Поділ двійкового числа на тріади та тетради починається від коми ліворуч і праворуч. Результат: 12345,6710=30071,52708=11000000111001,1010101112=3039,AB816.

Таблиця 2.2

3			0			0			7			1			>	5			2			7			0			8-кові цифри
0	1	1	0	0	0	0	0	0	1	1	1	0	0	1	>	1	0	1	0	1	0	1	1	1	0	0	0	2-кові цифри
3			0			3			9						>	A			B			8						16-кові цифри

2.3. Переведення чисел із системи числення з основою к у десяткову систему

Один із методів переведення чисел із системи числення з основою к у десяткову систему числення ґрунтується на використанні кількісного еквівалента числа. Для переведення необхідно записати число у його кількісному еквіваленті, замінивши цифри системи числення з основою к та основу к їхніми десятковими еквівалентами, а потім обчислити вираз за правилами десяткової арифметики.

Приклад 1. Переведемо двійкове число 1011,1001 у десяткову систему числення.

1011,1001 = Ь2³+0-2Ч1-2Ч1-2Ч1-2-'+0-2-²+0-2³+1-2'⁴ =

= 8 + 0 + 2+1+0,5 + 0 + 0 + 0,0625 = 11,5625;

Таким чином, $1011{,}1001_2 = 11{,}5625_{10}$.

Приклад 2. Переведемо вісімкове число 105,71 у десяткову систему числення.

105,71 = 1-8² + 0-8¹ + 5-8° + 7-8⁻¹ + 1-8⁻² = 64 + 0 + 5 + 0,875 + 0,015625 = 69,890625;

Результат: $105{,}71_8 = 69{,}890625_{10}$.

Приклад 3. Переведемо шістнадцяткове число 2ЕБ,0А до десяткової системи числення.

2ЕБ,0А = 2-16²+ и-Іб'+ІЗ-ІбЧО-Іб'+Ю-Іб-² = 512 + 224 +13+0+0,0390625 =

= 849,0390625;

Результат: $2ЕД0А_{16} = 849{,}0390625_{10}$.

2.4. Переведення чисел із десяткової системи у систему числення з основою к

Розглянемо переведення чисел із десяткової системи числення у іншу однорідну позиційну систему числення з основою *к,* коли дії виконуються в десятковій системі. У разі цього переведення окремо виконується переведення цілої частини числа й окремо - дробової; результати потім додаються.

Цілу частину десяткового числа X ділять на основу системи числення к за правилами десяткової арифметики до отримання залишку, який буде десятковим еквівалентом цифри молодшого розряду результату. Якщо частка від ділення не дорівнює 0, то вона стає діленим і процес ділення на к продовжується. Як тільки чергова частка стане рівною 0, процес ділення на к припиняється. Залишок, який отримали у результаті першого ділення на к, є цифрою розряду результату з вагою к°, залишок у результаті другого ділення - цифрою з вагою к' і т. д. Останній залишок є цифрою старшого розряду результату.

Дробова частина десяткового числа X множиться на к за правилами десяткової арифметики. В отриманому добутку від'єднується ціла частина, яка може дорівнювати 0, а дробова частина знову множиться на к із наступним від'єднанням цілої частини. Ця операція повторюється або до отримання нульової дробової частини добутку, або до отримання необхідної кількості розрядів числа $X_к$. Цифра старшого розряду результату переведення (тобто, перша після коми) збігається з першою від'єднаною цілою частиною, цифра другого розряду результату переведення - із другою від'єднаною цілою частиною і т.д. При цьому від'єднані цілі частини необхідно представити в системі числення з основою к.

Приклад. Переведемо десяткове число 11,5625 у двійкову систему числення з точністю до п'яти розрядів після коми.

Переведення цілої частини:

11:2 = 5, залишок 1 (молодший розряд результату),

5:2 = 2, залишок 1,

2:2= 1, залишок 0,

1:2 = 0, залишок 1 (старший розряд результату).

Результат: $П_{,0} = 1011_2$.

Процедура переведення дробової частини наведена у табл. 2.3.

Таблиця 2.3

Крок	Дріб	Результат множення на *к = 2*	Ціла частина результату множення, яка вилучається	Вага двійкового розряду
1	0.5625	1.125	1	Старший (перший після коми)
2	0.125	0.25	0	
3	0.25	0.5	0	
4	0.5	1.0	1	Молодший
5	0.0	0.0	0	

Результат: $0.5625_{10} = 0{,}1001 0_2$.

Повний результат: $11{,}5625_{,0} = 1011 + 0{,}10010 = 1011{,}10010_{,}$.

2.5. *Представлення чисел зі знаком*

Для позначення знаку числа в звичайній арифметиці використовують символи «-» та «+». Як зазначалося, у комп'ютерній техніці використовують елементи з двома станами, які можуть зберігати двійкову цифру (0 чи 1). Зрозуміло, що цю цифру доцільно використати і для позначення знаку числа, коли 0 відображає знак «+», а 1 - знак «-».

Для спрощення виконання арифметичних операцій додатні та від'ємні числа (тобто числа зі знаком) відображаються спеціальними кодами: прямим, оберненим та доповняльним.

2.5.1. Прямий код

У прямому коді лівий (його ще називають старшим) розряд позначає знак числа, а решта розрядів - саме число (рис. 2.1).

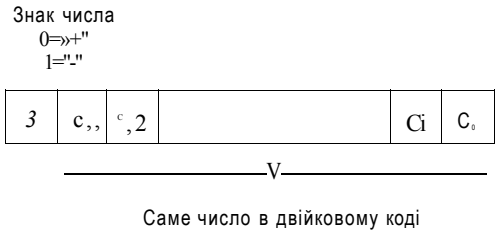


Рис. 2.1. Прямий код двійкового числа

Прямий код двійкового п-розрядного числа Є визначається як

$$[C \text{ при } в > 0;$$

$$I A + |C|, \text{ при } в < 0;$$

де А - величина, рівна вазі старшого разряду розрядної сітки (для дробових чисел А = 1, а для цілих чисел $A = 2^n$). Діапазон представлення чисел в прямому коді $0 < |Є| < A$. Додатні числа представляються кодами $0 < C_{пр} < A$, а від'ємні $0 < C_{пр} < 2A$

Ознакою представлення додатних або від'ємних чисел є наявність нуля або одиниці відповідно в старшому розряді, який називається знаковим. Цифрові розряди прямого коду представляють модуль числа, що забезпечує наочність представлення чисел в прямому коді

Наведемо кілька прикладів

$5_{10} = 00101_{прямий\ код}$ $\quad$ $5_{10} = 10101_{прямий\ код}$

$25_{10} = 011001_{прямий\ код}$ $\quad$ $-25_{10} = 111001_{прямий\ код}$

2.5.2. Обернений код

В оберненому коді, як і у прямому, старший розряд позначає знак числа (0 - додатне число, а 1 - від'ємне). Розряди додатного числа записуються у звичайному вигляді, а від'ємного - в інвертованому вигляді (замість 0 пишеться 1 і навпаки). На рис. 2.2 показано обернений код двійкового числа.

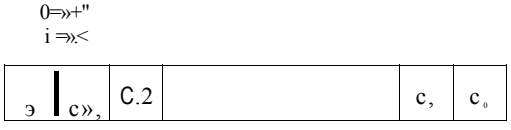


Рис. 2.2. Обернений код двійкового числа

Обернений код п-розрядного двійкового числа Є визначається як

$$\begin{cases} C \text{ при } б > 0; \\ B - |CI \text{ при } C<0; \end{cases}$$

де В - величина найбільшого числа без знаку, яке може бути розміщене в п- розрядній сітці (для дробових чисел $B = 2 - 2\text{-}(n\text{-}1)$, а для цілих чисел В = $2n\text{-}1)$. Діапазон зміни чисел в оберненому коді $0 < |C| < A$. Додатні числа представляються кодами в діапазоні $0 < C_{пр} < A$, а від'ємні - в діапазоні $A < C_{пр} < 2A$. За визначенням обернений код від'ємного числа є доповненням модуля вихідного числа до найбільшого числа без знаку, яке може бути розміщене в розрядній сітці. В зв'язку з цим отримання оберненого коду двійкового від'ємного числа зводиться до отримання інверсії п-розрядного коду модуля цього числа

Знову варто навести кілька прикладів

$5_{10} = 00101_{\text{обернений код}}$ $\quad -5_{10} = 10101_{\text{обернений код}}$.

$25_{10} = 011001_{\text{обернений код}}$, $\quad -25_{10} = 111001_{\text{обернений код}}$ -

2.5.3. *Доповняльний код*

Доповняльний код будується на основі оберненого. Якщо число додатне, то не проводиться жодних дій, якщо від'ємне - після інвертування до молодшого розряду числа додається одиниця (рис. 2.3).

Рис. 2.3. Доповняльний код двійкового числа

Доповняльний код п-розрядного двійкового числа Є визначається як

$$\begin{cases} C \text{ при } Є > 0; \\ C\text{-}|C|, \quad \text{при} C < 0; \end{cases}$$

де С - величина, рівна вазі розряду, який іде за старшим розрядом використаної розрядної сітки (для дробових чисел С = 2, а для цілих чисел $C = 2^{n-1}$). Діапазон зміни чисел у прямому коді $0 < |Є| < A$. Цифровими розрядами доповняльного коду додатного числа виражається модуль цього числа. Як уже було зазначено, доповняльний код від'ємного числа зручно отримувати через обернений код шляхом додавання 1 до молодшого розряду оберненого коду

Розглянемо приклади

$5_{10} = 00101_{\text{доповняльний код}}$ $\quad -5_{10} = 11011_{\text{доповняльний код}}$

$25_{10} = 011001_{\text{доповняльний код}}$. $\quad -25_{10} = 100111_{\text{доповняльний код}}$

$12_{10} = 01100_{\text{доповняльний код}}$ $\quad -12_{10} = 10100_{\text{доповняльний код}}$

Найчастіше серед розглянутих кодів в комп'ютерах використовується доповняльний код. Це зумовлено більшою зручністю проведення арифметичних операцій над числами, представленими в такому коді, оскільки при його застосуванні операція алгебраїчного додавання зводиться до додавання арифметичного.

2.6. *Формати даних*

2.6.1. Способи представлення чисел

Розряд двійкового числа представляється в комп'ютері деяким технічним пристроєм, наприклад тригером, двом різним станам якого приписують значення 0 та 1. Один двійковий розряд який може набувати ці два значення, є найменшою одиницею інформації, названої бітом. Набір відповідної кількості таких пристроїв слугує для представлення багаторозрядного двійкового числа (або в загальному випадку - двійкового коду слова). Розрядність слова може бути від 1 біта до довільної кількості п бітів. Слово із 8 бітів називають байтом. Як правило, коли йдеться про комп'ютерну техніку, всі виміри кількості розрядів наводяться в бітах або байтах Часто словом іще називають число із 32 бітів, а число із 16 бітів - півсловом.

Коли деяке число має 32 біти, то говорять, що воно представлене з одинарною точністю, якщо ж 64 біти - з подвійною точністю

Числові дані в комп'ютері зазвичай представляються трьома способами

- як цілі або дробові числа з фіксованою комою, які складаються із деякої кількості бітів;
- як числа з рухомою (ще деколи вживають "плаваючою") комою, кожне з яких має порядок та мантису
- як двійково кодовані десяткові, де байт (чи півбайта) представляє одну десяткову цифру, а послідовність байтів (чи півбайтів) представляє число.

Якщо певне число більше за максимальне, яке може бути представлене певною кількістю розрядів, то значення числа може втрачатися. Таку ситуацію називають переповненням Розробники комп'ютера або програми повинні передбачити, числа якої величини будуть використовуватись, і виділити для їх представлення таку кількість розрядів, щоб значення числа не втрачалось

Сучасні персональні комп'ютери використовують слова розрядністю від одного до 16 байтів. Спеціалізовані комп'ютери можуть використовувати слова й іншої розрядності, наприклад 128 байтів

Для кодування символів використовуються спеціальні коди, серед яких найпоширеніший у персональних комп'ютерах - американський стандартний код інформаційного обміну ASCII, а в мейнфреймах - розширений двійково-кодований десятковий код обміну EBCDIC Зазвичай для представлення одного символа використовується один байт

2.6.2. *Числа з фіксованою комою*

У разі використання чисел із фіксованою комою, представлення коми не виконується, але вважається, що вона є на певній наперед відомій позиції відносно розрядів числа. Найчастіше вважається, що кома стоїть після молодшого розряду числа (таким чином представляються цілі числа) або перед старшим розрядом числа (таким чином представляються дробові числа), хоча можливе застосування і змішаного варіанту. У такому форматі представляються числа з діапазону - $1 < число < 1$ (якщо є знаковий розряд) або $0 < число < 1$ (якщо знакового розряду немає).

На рис. 2.4 показано приклад розрядної сітки комп'ютера (формату даних) для представлення двійкових чисел із фіксованою комою в вигляді 32-розрядних слів для випад-

ків закріплення коми перед старшим і після молодшого розряду. Розряди пронумеровані зліва направо.

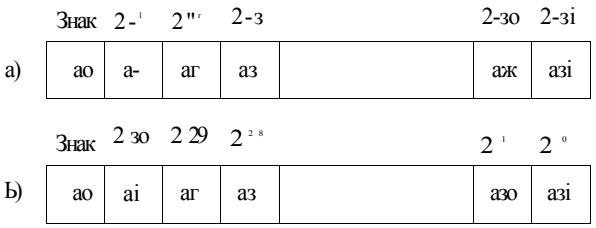


Рис. 2.4. Розрядна сітка при представленні двійкових чисел з фіксованою комою: а - кома фіксована перед старшим розрядом я,, *Ь - кома фіксована після молодшого розряду* $а_{31}$

Для кодування знаку числа використовується знаковий розряд ($а_0$ на рис. 2.4). У цьому розряді 0 відповідає знаку «+», а 1 - знаку «-». На розрядній сітці вказано вагу кожного розряду. Найбільше додатне число, яке може бути представлене в розрядній сітці, показаній на рис. 2.4 а, рівне 0,11...1=1-2"^{31}. Тут після коми розміщена 31 одиниця. А найменше додатне значуще число рівне 0,00... 01 =2~^{31}. Тут після коми розміщено 30 нулів.

В розрядній сітці (рис. 2.4 а) можуть бути представлені числа в діапазоні від -(1 -2"^{31}) до -2"^{31} і від +2"^{31} до +(1-2^{-31}), що відповідає діапазону абсолютних десяткових чисел приблизно від (1-Ю$^{-9}$) до 10"^{9}. Числа |x| < 2'^{31} не можуть бути представлені в розрядній сітці і приймаються рівними 0 (число виходить за розрядну сітку вправо). Всі числа |x| >=1 також не можуть бути представлені в прийнятій розрядній сітці.

Таке число виходить за межі сітки вліво (відбувається переповнення розрядної сітки), і його старші розряди (розряди зліва від коми) втрачаються, а результат обчислень виявляється неправильним. Тому, зазвичай, якщо при виконанні певної програми виникає переповнення, в арифметико-логічному пристрої формується сигнал, який фіксується в відповідному тригері та повідомляє операційну систему комп'ютера про наявність переповнення.

Якщо кома зафіксована праворуч від молодшого розряду, розрядна сітка (рис. 2.4 б) дозволяє представляти додатні та від'ємні цілі двійкові числа, модуль яких 1< = |x| < = 2^{31}-1, що відповідає діапазону абсолютних десяткових чисел приблизно від 1 до 10^{9}, а також 0.

Всі числа, модуль яких менший 1 або більший (2^{31}-1), не можуть бути представлені в цій розрядній сітці (число виходить за межі розрядної сітки).

При виконанні на комп'ютері обчислень необхідно, щоб всі вихідні та отримувані в процесі обчислень проміжні і кінцеві дані не виходили за діапазон чисел, які можуть бути представлені в цій розрядній сітці. В іншому випадку в обчисленнях можуть виникнути помилки. Для цього під час написання програм дані, що задіяні в обчисленнях, беруться з відповідними масштабними коефіцієнтами.

При виконанні науково-технічних розрахунків масштабування є простішим, якщо всі числа по модулю не перевищують 1, тобто кома зафіксована перед старшим розрядом числа.

Комп'ютери, які опрацьовують числа в форматі з фіксованою комою, є простішими (меншими за габаритами) та швидшими порівняно з комп'ютерами, які опрацьовують числа в форматі з рухомою комою, але в них можливе виникнення проблем через по-

требу передбачення переповнення. Перші комп'ютери опрацьовували дані з фіксованою комою, причому кома, як правило, фіксувалась перед старшим розрядом числа. Зараз представлення чисел з фіксованою комою використовується як єдине лише в порівняно невеликих за своїми обчислювальними можливостями комп'ютерах, які використовуються для управління технологічними процесами та опрацювання вимірювальної інформації в реальному часі.

В комп'ютерах, призначених для вирішення широкого кола обчислювальних задач, основним є представлення чисел з рухомою комою, яке не вимагає масштабування даних.

Однак у таких комп'ютерах, крім представлення чисел в цьому форматі, часто використовується представлення з фіксованою комою, оскільки на виконання операцій з такими числами витрачається менше часу. При цьому в більшості випадків формат чисел із фіксованою комою слугує для представлення цілих двійкових чисел (кома ставиться праворуч від молодшого розряду числа) та виконання операцій над ними, що, зокрема, необхідно для операцій над кодами адрес (операцій індексної арифметики).

Розглянемо основні формати чисел із фіксованою комою, що використовуються у сучасних комп'ютерах, та діапазони представлення в них чисел (табл. 2.4).

Таблиця 2.4

Формати без знакового розряду		
довжина	Min	Max
16 байт = 128 біт	0	3.40282366920938e+38
8 байт = 64 біт	0	18446744073709551615
4 байти = 32 біт	0	4294967295
2 байти = 16 біт	0	65535
1 байт = 8 біт	0	255
Формати зі знаковим розрядом (доповняльний код)		
довжина	Min	Max
16 байт = 128 біт	-1.70141183460469e+38	1.70141183460469e+38
8 байт = 64 біт	-9223372036854775808	9223372036854775807
4 байти = 32 біт	-2147483648	2147483647
2 байти =16 біт	-32768	32767
1 байт = 8 біт	-128	127

Інколи під час створення спеціалізованих комп'ютерів зручно ставити кому в іншу позицію (не лише праворуч від молодшого розряду числа або ліворуч від старшого розряду числа), і таким чином виокремлювати певну кількість розрядів для подання цілої частини числа, а також певну кількість розрядів для подання дробової частини числа.

2.6.3. *Числа із рухомою комою*

Не завжди діапазон представлення чисел у форматі з фіксованою комою є достатнім для проведення обчислень. В такому випадку використовується формат представлення чисел із рухомою комою.

У загальному випадку в форматі з рухомою комою число подається у вигляді $A = \pm т \cdot q^{\pm р}$, де т - мантиса числа, q - основа порядку, ± р - порядок числа. Попередній вираз можна записати як $A = \pm т_д \cdot \pm р_д$, де упущено основу порядку, оскільки в комп'ютерах вона незмінна. В більшості випадків основа порядку дорівнює основі системи числення, тобто 2.

Для однозначного і максимально точного відображення чисел число з рухомою комою представляють у нормалізованому вигляді. Якщо виконується нерівність q - 1 < = |т| <1, а у випадку двійкової системи числення 0.5 < = |пт| <1 (старший двійковий розряд мантиси дорівнює 1), то вважається, що число представлене в нормалізованому вигляді.

Таким чином, у двійкового нормалізованого числа у форматі з рухомою комою мантиса є правильним дробом і у старшому розряді мантиси завжди стоїть 1. Операцію приведення числа до нормалізованого вигляду називають нормалізацією. Нормалізація чисел у комп'ютері виконується або апаратно, або ж спеціальною програмою.

Для представлення двійкового числа у форматі з рухомою комою у розрядній сітці, наданій для цієї мети, виділяється:

- по одному розряду для представлення знаку числа Біті (поле знаку числа) і знаку порядку Бр;
- певне число розрядів для представлення значення порядку р;
- розряди для представлення значення модуля мантиси т (поле мантиси).

Наприклад, можливий такий варіант, коли формат числа складається з чотирьох полів (рис. 2.5) тобто, [А] = Б г>ₐ5ₜтₐ.

5р	*Рл*	*т*	$m_{л}$

Рис. 2.5. Число з рухомою комою

Зазвичай у форматі з рухомою комою замість порядку р використовують так звану характеристику ("зміщений порядок") г = ± р + 1, де 1 - надлишок (зсув), значення якого підбирається таким чином, щоб у разі зміни значення показника від деякого мінімального значення -|$р_{тіп}$| до максимального +|$р_{тах}$|» характеристика г змінювалася від 1 до $г_{тах}$. Отже, характеристика не змінює свого знаку. У цьому випадку відпадає необхідність у відображенні знаку порядку Б . Для цього приймається, що 1 = $2^{к-1}$, де к - число розрядів, виділених для представлення порядку числа у форматі з рухомою комою.

Тоді формат числа з рухомою комою можна подати так, як показано на рис. 2.6 (з використанням трьох полів), тобто, [А] = 5ₜр тₐ.

$5_{т}$ $р_{г}$ $m_{л}$

Рис. 2.6. Число з рухомою комою із зміщеним порядком

Одиниця старшого розряду нормалізованої мантиси зазвичай не відображається у форматі числа, тобто є уявною. Розряд слова, в якому повинна була бути відображена ця одиниця, використовується як молодший розряд характеристики, або старший розряд мантиси, що дозволяє збільшити діапазон представлення чисел у форматі з рухомою комою, або точність обчислень.

Таким чином, мантиса в такому варіанті відображається, починаючи з розряду, що йде після старшого. Це слід враховувати під час виконання будь-якої операції з мантисою числа, і перед початком операцій відновлювати старший розряд мантиси. Після завершення операцій формування нормалізованого результату у відведеній для нього розрядній сітці, старша одиниця мантиси знову відкидається. Порядок із к-розрядним полем може змінюватися в межах від -$2^{к-1}$ + 1 до + $2^{к-1}$ - 1 (табл. 2.5, к = 3).

Таблиця 2.5

Показник порядку	Прямий код показника	Характеристика (показник + 4)	Примітки
+3	011	111	3+4=7
+2	010	110	2 + 4 = 6
+ 1	001	101	1+4 = 5
0	000	100	0 + 4 = 4
-1	. 101	011	-1+4 = 3
-2	110	010	-2 + 4 = 2
-3	111	001	-3 + 4=1
		000	Ознаку нуля

Як зазначалося, характеристика г - це порядок р з надлишком 1 = $2^{к-1}$. Вона не змінює свого знаку і змінюється від 1 (при р = $-2^{к}$~') до $2^{к-1}$ (при р = $+2^{к}$~' - 1). Винятком є число 0, яке виражається нульовою характеристикою і нульовою мантисою (не обов'язково).

Основною перевагою представлення чисел у форматі з рухомою комою є великий діапазон машинних чисел і висока точність їхнього подання. Діапазон визначається довжиною розрядної сітки, виділеної для характеристики, а точність визначається довжиною розрядної сітки, виділеної для мантиси.

Особливості виконання операцій над числами з рухомою комою:

- збільшення мантиси у 2 рази здійснюється зсувом двійкового значення мантиси ліворуч (у бік старших розрядів);
- зменшення мантиси у 2 рази здійснюється зсувом двійкового значення мантиси праворуч (у бік молодших розрядів);
- величина числа не зміниться, якщо збільшити мантису в 2 рази і одночасно зменшити порядок на 1;
- величина числа не зміниться, якщо зменшити мантису в 2 рази і одночасно збільшити порядок на 1.

Тобто формат з рухомою комою має недолік, який полягає у відсутності унікального представлення для кожного числа. Усі числа, що наводяться на рис. 2.7, є еквівалентними. Слід зауважити, що цього недоліку не мають нормалізовані числа.

0 1 0 1 0 1 1 0 0 0 1 0 0 0

0 1 0 1 1 0 0 1 0 0 0 1 0 0

0 1 0 1 1 1 0 0 1 0 0 0 1 0

0 1 1 0 0 0 0 0 0 1 0 0 0 1

Рис. 2.7. Еквівалентні двійкові числа в форматі з рухомою комою

Під час арифметичних операцій над числами з рухомою комою виконуються дії як над порядком, так і над мантисою.

У деяких моделях комп'ютерів одержало поширення відображення чисел із рухомою комою з основою порядку, рівною цілому ступеню числа 2 (в *-2'*). При цьому порядок *р* відображається двійковим цілим числом, а мантиса *m* - числом, в якому групи по *г* двійкових розрядів зображають цифри мантиси з основою системи числення в.

Прикладами вживаних основ порядку є числа 8 та 16.

Використання для чисел з рухомою комою недвійкової основи порядку дещо зменшує точність обчислень (при заданому числі розрядів мантиси), але дозволяє збільшити діапазон чисел, що представляються в машині, і прискорити виконання деяких операцій, зокрема нормалізації, за рахунок того, що зсув проводиться відразу на кілька двійкових розрядів. Крім того, зменшується вірогідність появи ненормалізованих чисел в ході обчислень.

Наприклад, у разі використання шістнадцяткових чисел з рухомою комою число X вважається нормалізованим, якщо старша шістнадцяткова цифра X відмінна від 0. Тобто у нормалізованому числі три старші двійкові цифри можуть дорівнювати 0. Це дещо зменшує точність представлення чисел при фіксованому числі розрядів мантиси. Якщо г старших шістнадцяткових розрядів мантиси рівні 0, то нормалізація в цьому випадку полягає в зсуві вліво мантиси на г шістнадцяткових розрядів і відповідному зменшенні показника порядку на г одиниць. Зсув на один шістнадцятковий розряд виконується як зсув мантиси відразу на чотири двійкові розряди.

Розглянемо кілька прикладів.

Припустимо, що потрібно подати у форматі з рухомою комою число 17. Для десяткової системи $17 = 17.0 \times 10^{0} = 1.7 \times 10^{1} = 0.17 \times 10^{2}$. Аналогічно в двійковій системі $17 = 10001_2 \times 2^{0} = 1000.1_2 \times 2^{1} = 100.01_2 \times 2^{2} = 10.001_2 \times 2^{3} = 1.0001_2 \times 2^{4} = 0.10001_2 \times 2^{5}$. Якщо використати останній запис, то 8-розрядна мантиса числа буде рівною 10001000, а 5-розрядний порядок буде рівним 00101. Тоді число 17 в форматі з рухомою комою в двійковій системі має вигляд, показаний на рис. 2.8 а. Використовуючи формат з рухомою комою можна представляти числа в значно ширшому діапазоні, ніж використовуючи формат з фіксованою комою, при тих самих 14 розрядах. Так, на рис. 2.8 Ь) показано число $65536 = 0.1_2 \times 2^{17}$ у форматі з рухомою комою, для представлення якого у форматі з фіксованою комою потрібно було б і б розрядів.

а) 0 0 10 1 1 0 0 0 1 0 0 0

Ь) о 1 0 0 0 1 1 0 0 0 0 0 0 0

Рис. 2.8. Числа 17 а) та 65536 Ь) в двійковому форматі з рухомою комою

Як вже зазначалося, ідея зміщення порядку полягає в перетворенні його значень лише в додатні числа. Зміщення здійснюється шляхом додавання до кожного значення порядку фіксованого числа, рівного середньому значенню величини діапазону можливих чисел, яке вибирається для представлення нуля. В приведених вище прикладах як зміщення потрібно взяти число 16, тому що воно є середнім між 0 і 31 (порядок має 5 бітів, тому дозволяє представити $2^{5} = 32$ значень). Будь-яке число, більше ніж 16, в полі порядку буде представляти додатне значення, а менше - від'ємне. Зауважимо іще раз, що значення порядку з усіма нулями та одиницями зазвичай резервується для спеціальних випадків (таких як нуль та нескінченість).

Повернемося до попереднього прикладу. Ми обчислили 17 = $0.10001_2 \times 2^5$. Зміщення порядку рівне 16 + 5 = 21, і число має вигляд, показаний на рис. 2.9 а. Аналогічно для числа 0.25 = 1.0 x 2^{-2} будемо мати представлення, показане на рис. 2.9 Ь.

а) 0 10 10 1 1 0 0 0 1 0 0 0

Ь) 0 0 1 1 1 0 1 0 0 0 0 0 0 0

Рис. 2.9. Числа 17 а) та 0.25 Ь) в двійковому форматі з рухомою комою із зміщеним порядком

Розглянемо іще один приклад числа з рухомою комою, в даному випадку нормалізованого. Виразимо 0.03125_{10} в форматі з рухомою комою із зміщенням порядку на 16. Тоді $0.03125_{10} = 0.00001_2 \times 2° = 0.0001 \times 2'' = 0.001 \times 2^{2} = 0.01 \times 2^{1} = 0.1 \times 2^{4}$. Додавши до порядку зміщення отримаємо 16 - 4 = 12. Повний вигляд числа показано на рис. 2.10.

0 0 1 1 0 0 1 0 0 0 0 0 0 0 0

Рис. 2.10. Число 0.03125_{10} в нормалізованому двійковому форматі з рухомою комою із зміщеним порядком

На рис. 2.11 наведено два приклади використовуваного в комп'ютерах формату представлення чисел з рухомою комою. Зверху наведено формат, який був використаний в комп'ютерах CDC 6600, CDC 7000, та CYBER 170 фірми СКС, а знизу - формат, який був використаний в комп'ютерах системи IBM/370 фірми IBM, причому тут основою порядку є число 16, тому мантиса вважається нормалізованою, якщо є хоча б одна одиниця в перших її чотирьох розрядах.

Рис. 2.11. Формат з рухомою комою комп'ютерів фірм CDC та IBM

Існує велика кількість задач, коли обробці підлягають масиви чисел, які змінюються в вузькому діапазоні значень. В цьому випадку з метою більш ефективного використання розрядної сітки для представлення чисел використовують так звану поблоково-рухому кому, коли для всього масиву чисел є лише один порядок. В спеціалізованих комп'ютерах це дозволяє суттєво зменшити витрати обладнання на побудову арифметико-логічного пристрою.

При потребі іще більшого розширення діапазону представлення даних використовується так званий формат з рухомою-рухомою комою. Тут використовується два поля порядку, як це показано на рис. 2.12.

Sm	PI	P2	M

Рис. 2.12. Формат з рухомою-рухомою комою

Тут значення числа визначається з виразу A = 2^{e}, де e = P 2^{P2}.

2.6.4. *Стандарт IEEE-754*

Для представлення чисел з рухомою комою у більшості сучасних комп'ютерів використовується стандарт IEEE-754. В попередньому пункті ми розглянули можливі варіанти представлення даних в форматі з рухомою комою. До середини 80-х років в різних комп'ютерах використовувались різні варіанти цього представлення, що суттєво ускладнювало виконання на них тих самих програм. У 1985 році Інститут інженерів електротехніків і радіоелектроніків (IEEE) розробив стандарт для чисел з рухомою комою, який офіційно відомий як IEEE-754 (1985).

Стандарт IEEE-754 для чисел з одинарною точністю використовує зміщення 8-розрядного порядку на 127. Це ще один спосіб представлення чисел із знаком без використання знаку мінус. Мантиса має 23 біти. Із знаковим розрядом включно повна довжина слова складає 32 біти (рис. 2.13).

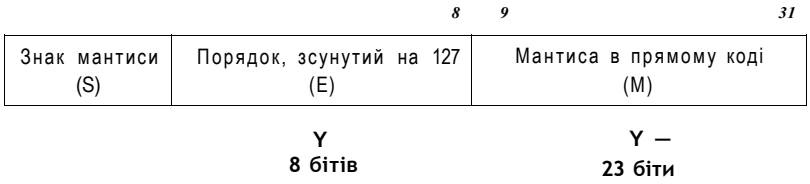


Рис. 2.13. IEEE-754 стандарт для чисел з одинарною точністю

Значення числа обчислюється за формулою:

$$число = (-1)^{s} \cdot 2^{E \sim 127} \cdot (1,M).$$

Мантиса представляється в прямому коді без знаку, знак мантиси представляється окремо. Суть нормалізації полягає в тому, що мантиса приводиться до вигляду I.xxxxx, тобто вона знаходиться в межах від 1,000...0 до 1,111...1. Слід зауважити, що оскільки кожна мантиса після нормалізації починається з 1, то нема сенсу зберігати цей розряд, тому він не зберігається разом з числом. Його необхідно просто враховувати під час операцій над числами.

Числа з подвійною точністю в стандарті IEEE-754 подаються 64-розрядним словом, яке має знаковий розряд, 11-розрядний порядок і 52-розрядну мантису (рис. 2.14). Зміщення порядку дорівнює 1023.

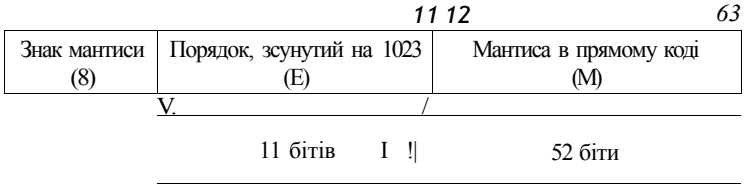


Рис. 2.14. IEEE-754 стандарт для чисел з подвійною точністю

Значення числа обчислюється за формулою:

$$число = (-1)^{s} \cdot 2^{E-1023} \cdot (1, M).$$

Діапазон чисел, які можуть бути представлені в цьому форматі, показаний на рис. 2.15.

Рис. 2.15. Діапазон чисел, які відображаються у форматі за стандартом IEEE-754 з подвійною точністю

У табл. 2.6 наведено характеристики форматів подання двійкових чисел в стандарті IEEE-754 з одинарною та подвійною точністю.

Таблиця 2.6

Характеристика	Формат з одинарною точністю	Формат з подвійною точністю
Довжина слова	32 біти	64 біти
Мантиса (зі знаком)	24 біти	53 біти
Порядок	8 бітів	11 бітів
Зміщення	127	1023
Наближений діапазон	$2^{128} = 3.8 \cdot 10^{38}$	$2^{1024} \sim 9 \cdot 10^{307}$
Найменше нормалізоване число	$2^{-126} \sim 10^{-38}$	$2^{-1022} \sim 10^{-308}$
Наближена точність представлення чисел	$2^{-23} = 10^{-7}$	$2^{-52} \sim 10^{-15}$

Як числа з одинарною точністю, так і числа з подвійною точністю в стандарті IEEE-754 мають для нуля два варіанти представлення. Коли порядок і мантиса рівні нулю - число є нулем. При цьому значення знаку є несуттєвим. На цю обставину потрібно звертати увагу при проведенні операції порівняння числа з рухомою комою на збіжність з нулем.

Стандарт IEEE-754 передбачає використання певної кількості значень мантиси та порядку для представлення нескінчених, невизначених та малих значень. Так мінус та плюс нескінченість подаються максимальним значенням порядку (377_8 для числа з оди-

нарною точністю та 3777_8 для числа з подвійною точністю) та нульовою мантисою. Для представлення невизначеного значення також використовується максимальне значення порядку та ненульова мантиса. Невизначене значення називають "не числом" - Not a Number (NaN). "Не число" використовується, щоб представити значення, яке не є дійсним числом і часто використовується як індикатор помилки, наприклад, коли відбулося ділення 0 на 0. Якщо число є дуже малим, то воно представляється нульовим порядком та ненульовою мантисою. У табл. 2.7 наведено приклади представлення різних величин в форматі за стандартом IEEE-754 для чисел з одинарною та подвійною точністю.

Таблиця 2.7

	32-розрядні числа			64-розрядні числа		
Приклади	Знак	Порядок	Мантиса	Знак	Порядок	Мантиса
NaN	?	377_8	Не нуль	0	3777_8	Не нуль
+ нескінченість	0	377_8	$.00000000_8$	0	3777_8	$.0000000000000000_8$
10,0	0	202_8	$(1).20000000_8$	0	2002_8	$(1).0400000000000000_8$
1,0	0	177_8	$(1).00000000_8$	0	1777_8	$(1).0000000000000000_8$
0,0	0	000_8	$.00000000_8$	0	0000_8	$.0000000000000000_8$
Ненормалізоване	0	000_8	Не нуль	0	0000_8	Не нуль
-0,0	1	000_8	$.00000000_8$	0	0000_8	$.0000000000000000_8$
-1,0	1	177_8	$(1).00000000_8$	1	1777_8	$(1).0000000000000000_8$
-10,0	1	202_8	$(1).20000000_8$	1	2002_8	$(1).0400000000000000_8$
- нескінченість	1	377_8	$.00000000_8$	1	3777_8	$.0000000000000000_8$

Тут знаком ? позначено несуттєве значення, а (1) - значення, яке не зафіксовується елементами пам'яті комп'ютера.

2.6.5. *Кодування алфавітно-цифрово) інформації*

2.6.5.1. Двійково-кодовані десяткові числа

Вище було показано представлення в комп'ютері даних у двійковій системі числення. Далі розглянемо, як ці внутрішні дані можуть бути перетворені у форму, яка піддається інтерпретації людиною.

Двійково-кодоване десяткове число - це десяткове число, кожна цифра якого представлена в двійковій формі. Одна з перших числова система кодування десяткових чисел двійковим кодом (Binary-coded decimal - BCD) була використана в великих і середнього розміру комп'ютерних системах фірми IBM. Система BCD кодує кожну цифру десяткового числа 4-розрядним двійковим кодом. Коли використовується 8-розрядне число, тобто байт, то старші 4 біти називають зоною, а молодші - цифрою. Ця домовленість прийшла з часів перфокарт, де кожна колонка карти могла мати "зональний отвір" в одній з двох верхніх стрічок і "цифровий отвір" в одній з десяти нижніх стрічок. Старші чотири розряди в байті BCD використовуються для представлення знаку, який може мати одне з трьох значень: число без знаку представляється кодом 1111; додатне число

представляється кодом 1100; від'ємне число представляється кодом 1101. Кодування для двійково-кодованих десяткових чисел показане в табл. 2.8.

Таблиця 2.8

Цифра	Код BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
Зони	
1111	Без знаку
1100	Додатне
1101	Від'ємне

Як видно з таблиці, шість можливих двійкових значень (від 1010 до 1111) не використовуються. Хоча втрачається приблизно 40 % можливих значень, але набуваються значні переваги в точності. Наприклад, десяткове число 0.3, перетворене в двійковий код та обмежене 8-розрядною сіткою, при зворотному перетворенні має значення 0.296875, тобто похибка складає приблизно 1.05 %. В коді BCD число запам'ятається безпосередньо як 1111 0011, не допускаючи жодної помилки.

Цифри в коді BCD займають лише чотири розряди, тому можна зберегти місце і спростити обчислення, розмістивши поряд числа з одним знаком. Цей процес називається пакуванням, а сформовані числа - пакованими десятковими числами.

Приклад:

Подамо число -1265 трьома байтами, використовуючи паковані цифри коду BCD.

Зонне десяткове кодування для числа 1265 є наступним:

1111 0001 1111 0010 1111 0110 1111 0101

Після пакування отримаємо:

0001 0010 0110 0101

Додавши знак після цифри молодшого розряду і заповнивши цифру старшого розряду одиницями до 3 байтів, отримаємо:

1111 0001 001001100101 1101.

Код BCD (або його іще називають кодом 8421) знайшов найбільше поширення в обчислювальній техніці. Цей код зручний для виконання перетворення з десяткової системи у двійкову і навпаки. Цей код адитивний, тобто сума представлення двох цифр є кодом їх суми.

Разом з тим, використання цього коду пов'язане з труднощами пошуку переносу в наступний десятковий розряд і важкістю переходу до оберненого та доповняльного коду для десяткових чисел. Це пояснюється тим, що код 8421 не є самодоповнюючим, тобто

інверсія його двійкових цифр не дає коду доповнення десяткової цифри до 9. В табл. 2.9 наведено інші широко вживані двійково-десяткові коди, а саме код з надлишком 3 та код 2 з 5. Можна побудувати й інші двійково-десяткові коди, наприклад 2421, 5121 тощо.

Таблиця 2.9

Десяткові цифри	Код з надлишком 3	Код 2 з 5
0	ООП	11 000
1	0100	00 011
2	0101	00 101
3	0110	00 П0
4	0111	01 001
5	1000	01 010
6	1001	01 100
7	1010	10 001
8	1011	01 001
9	1100	10 100

Код з надлишком 3 зручний при виконанні арифметичних операцій над десятковими цифрами, оскільки він є самодоповнюючим. Крім того, легко визначається перенос, так як сума двох доданків, кожне з яких береться з надлишком 3, вийде з надлишком 6, що виключає лишні кодові комбінації. Для отршмання правильного коду суми з отриманого результату відкидається 3. У деяких випадках для використання суттєво, що код нуля містить 1 і тому легко відрізнити наявність коду нуля від пропадання коду цифри. Код з надлишком 3 не дуже зручний для перетворення чисел з однієї системи числення в іншу. В коді 2 з 5 десяткові цифри зображаються п'ятьма розрядами, причому кожне значення містить дві 1. Ця надлишковість використовується для контролю правильності передачі цифри. Будь-яка помилка в одному розряді перетворює 0 в 1 або 1 в 0, в результаті вийде більше або менше двох 1, що вкаже на помилку. При одночасній появі двох помилок можливі випадки, коли їх можна не знайти (якщо 0 в одному розряді перетворюється в 1, а в іншому розряді 1 в 0).

2.6.4.2. Розширений двійково-кодований десятковий код обміну EBCDIC

Перед розробкою комп'ютерної системи IBM System/360, фірма IBM використала 6-розрядну версію двійково-кодованого десяткового числа для представлення символів і знаків. Цей код мав суттєві обмеження, пов'язані з малою розрядністю двійково-кодованого десяткового числа. Проектувальникам System/360 була потрібна більша інформаційна здатність коду так само як і узагальнений метод запам'ятовування і чисел, і символів. Для того, щоб підтримувати сумісність з попередніми комп'ютерами і периферійним устаткуванням, інженери IBM вирішили, що буде краще просто розширити код BCD від 6 бітів до 8 бітів. Відповідно, цей новий код було названо розширеним двійково-кодованим десятковим кодом обміну (EBCDIC). Фірма IBM і тепер продовжує використовувати код EBCDIC в мейнфреймах і обчислювальних системах середнього розміру. В табл. 2.10 код EBCDIC показано в зонально-цифровій формі.

Таблиця 2.10

Цифра

Зона	0000	(ККН	0010	ООП	0100	0101	0110	Olli	1000	1001	1010	1011	1100	1101	1110	F
(XXX)	N1 IL	son	STX	FTX	PH'	irr	*1C*	DEL			SMM	VT	FF	CR	SO	Sl
(XX) 1	ÞÍ K	IX'1	002	TM	RI:S	N1.	BS	Il	CAN	EM	er	*CA* ıı	ıı S	IGS	IRS	ins
0010	OS	SOS	FX		BYP	LF	етв	ESC			SM	cm		LNQ	ACK	BEL
ООП			SYN		PN	RS	uc:	ЕОТ				сиз	IX'4	NA к		Sl 'н
0100	SP												<	(	+	1
0101	&											S	*	)		-.
ОНО	—	/											•*i*		>	
Olli												#	©		=	"
1000		a	b	c	d	c	r	g	Ii	i						
1001		j	k	1	m	П	0	P	4	r						
1010			s	I	и	V	w	X	y	z						
1011										-						
1100		Л	B	C	1)	i:	]·	Ii	Il	I						
1101		J	К	L	M	N	0	P	У	R						
1110			S	T	U	V	w	X	Y	/,						
1111	0	1	1	3	4	5	6	7	8	9						

Знаки представлені шляхом додавання бітів до зонних бітів. Наприклад, знаку а відповідає код 1000 0001, а цифрі 3 - код 1111 0011. Зауважимо, що єдина різниця між верхніми і нижніми символами полягає в позиції розряду 2, що дозволяє зробити перетворення від верхніх до нижніх символів (або навпаки) шляхом переключення одного двійкового розряду. Зональні біти також роблять легшою для програміста перевірку правильності вхідних даних. Розшифрування абревіатур з табл. 2. 10 наведено в табл. 2.11.

Таблиця 2.11

NULL	Null	TM	Tape mark	ETB	End of transmission block
SOH	Start of heading	RES	Restore	ESC	Escape
STX	Start of text	NL	New line	SM	Set mode
ETX	End of test	BS	Backspace	CU2	Customer use 2
PF	Punch off	IL	Idle	ENQ	Enquiry
HT	Horizontal tab	CAN	Cancel	ACK	Acknowledge
LC	Lowercase	EM	End of medium	BEL	Ring the bell (beep)
DEL	Delete	cc	Cursor Control	SYN	Synchronous idle
RLF	Reverse if needed	cui	Customer use 1	PN	Punch on
SMM	Start manual message	IFS	Interchange file separator	RS	Record separator
VT	Vertical tab	IGS	Interchange group separator	UC	Uppercase
FF	Form feed	IRS	Interchange record separator	EOT	End of transmission
CR	Carriage return	IGS	Interchange unit separator	сиз	Customer use 3
SO	Shift out	DS	Digit select	DC4	Device control 4
Sl	Shift in	SOS	Start of significance	NAK	Negative acknowledgement
DLE	Data link escape	FS	Field separator	SUB	Substitute
DC1	Device control 1	BYP	Bypass	SP	Space
DC2	Device control 2	LF	Line feed		

2.6.4.3 *Американський стандартний код інформаційного обміну ASCII*

Американський стандартний код інформаційного обміну (American Standard Code for Information Interchange (ASCII)) з'явився завдяки зусиллям розробників покращити засоби передачі даних між системами. Міжнародна організація стандартизації ISO запропонувала цей 7-розрядний код взамін 5-розрядного коду, який використовувався в телетайпах.

Код ASCII визначає коди для 32 символів керування, 10 цифр, 52 букв, 32 спеціальних символів (таких як $ та #), а також для символу інтервалу (табл. 2.12). Старший восьмий біт було введено для забезпечення перевірки на парність. Цей біт дозволяє виявля-

ти однократні помилки при передачі даних. Тут значення записані в десятковій системі числення.

Таблиця 2.12

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
000	NUL	SOH	SIX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	EE	CR	SO	SI
001	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
010		!	"	#	$	%	&	'	(	)	*	+		—		/
011	0	1	2	3	4	5	6	7	8	9		;	<	=	>	?
100	@	A	B	c	D	E	F	G	H	I	J	K	L	M	N	O
101	P	Q	*R*	S	T	U	V	W	X	Y	z	I	\	1	A	
110	·	a	*Ь*	c	d	c	I'	o	h	i	j	k	1	m	n	0
111	Р	Ч	*x*	s	t	u	V	ow	X	у	z	1	ı	)	_	DEL

Розшифрування абревіатур з табл. 2.12 наведено в табл. 2.13.

Таблиця 2.13

Лбііісвіатура	РоНUН<>pvBaHHn aur.tificbKoio	Розшифрування українською
NUL	NULL	Пусто
SOU	START OF HEADING	Початок заголовка
STX	START OF TEXT	Початок текст\
ETX	END OF TEXT	Кінець ТСКСТУ
EOT	EN)) OF TRANSMISSION	Кінець передачі
1-M)	ENQUIRY	Хто там?
Лі 'К	ACKNOWLEDGE	Підівердження
BEL	BELL	Дзвінок
1ДS	BACKSPACE	Повернення на крок
TAB	HORIZONTAL TAB! ILATION	Горизонтальна табуляція
LI-	LINEFEED	Подача нової стрічки
VI'	VERTICAL TABULATION	Вертикальна табуляція
FF	FORM FEED	Подача нової форми
OR	CARRIAGE RETURN	Повернення карсгки
SO	SHIFT OUT	Вихід
SI	SHUT IN	В\іц
DLL	DATA LANK ESCAPE	Авторепстр 1
IX'1	DEVICE CONTROL ONE	Контроль пристрою 1
DC2	DEVICE CONTROL TWO	Контроль пристрою 2
ОС1	DEVICE CONTROL THREE	Контроль пристрою 3
IX "4	DEVICE CONTROL FOUR	Контроль присгрою 4
NAK.	NEC ATI VI-: AC 'KN( )WI .EDO! *1*	Ні
SYN	SYNCHRONOUS IDLE	Синхронізація
ETB	ENI) ()I- TRANSMISSK)N BLOCK	Кінець блока
CAN	CANOEI.	Анулювання
F.M	END OF MEDIl FM	Кінець носія
sun	SUBSTTrUTE	Заміна
ESC	ESCAPE	Авторегістр 2
FS	FILE SEPARATOR	Розділювач файлів
RS	RECORD SEPARATOR	Розділювач записів
1!S	UNO' SEPARATOR	Розділювач елементів
DEL)	DELETE	Видалення

З підвищенням надійності комп'ютерної техніки важливість біта парності знизилась, тому на початку 80-х років його стали використовувати для розширення набору кодованих символів в межах від 128_{10} до 255_{10}. Це можуть бути, наприклад, математичні символи, або символи іноземних мов.

2.6.4.4. Стандарт кодування символів Unicode

Коди EBCDIC та ASCII забезпечили кодування букв латинського алфавіту. З метою забезпечення кодування букв інших алфавітів та підтримки мов народів світу в 1991 році було запропоновано код під назвою Unicode.

Unicode - це 16-розрядний алфавіт, сумісний з ASCII та погоджений з міжнародним алфавітом ISO/IEC 10646-1. Оскільки 16-ма розрядами можна закодувати 64К символів, цього достатньо для кодування всіх букв алфавітів народів світу.

Кодовий простір коду Unicode вміщує 5 частин, як це показано в табл. 2.14.

Таблиця 2.14

Тип символу	Опис набору символів	Кількість символів	Шістнадцяткові значення символів
Алфавіти	Латинський, кирилиця, грецький і т.д.	8192	Від 0000 до 1FFF
Символи	Графічні мітки, математичні символи і т.д.	4096	Від 2000 до 2FFF
СІК	Китайські, японські і корейські фонетичні символи і пунктуації	4096	Від 3000 до 3FFF
Нап	Уніфіковані китайські, японські і корейські	40960	Від 4000 до DFFF
	Розширення чи надлишок від Нап	4096	Від Е000 до EFFF
Вказані користувачем		4096	Від F000 до FFFF

2.7. *Короткий зміст розділу*

У цьому розділі було показано, що в зв'язку з використанням у комп'ютерах елементів з двома станами, всі числа в них представляються у двійковій системі числення. Тому були висвітлені основні питання відображення чисел і символів у цій системі, а також її зв'язок з вісімковою, шістнадцятковою, та десятковою системами.

Було розглянуто особливості представлення чисел зі знаком спеціальними кодами: прямим, оберненим та доповняльним, які використовуються для спрощення виконання арифметичних операцій.

Оскільки числові дані в комп'ютері представляються у три способи: як цілі або дробові числа з фіксованою комою, як числа з рухомою комою та як двійково-кодовані десяткові числа, в цьому розділі було описано формати даних з фіксованою та з рухомою комою, включаючи стандарт IEEE-754, та було детально проаналізовано характеристики цих форматів.

2.8. Література для подальшого читання

Еволюція сисем числення показана в роботі [1]. У роботах [2-5] описано позиційні системи числення та представлення даних у двійковому, вісімковому та шістнадцятковому кодах, показано переведення чисел із системи числення з основою к до десяткової, та переведення чисел із десяткової до системи числення з основою к, а також описано представлення чисел із знаком в прямому, оберненому та доповняльному кодах. Особливості представлення чисел в форматі з рухомою комою, включаючи і стандарт IEEE-754,

подано в роботі [6]. Детальна інформація про Unicode може бути знайдена на сторінці Unicode Consortium www.unicode.org, так само як в описі стандарту *The Unicode Standard, Version 3.0* (2000). Сторінка International Standards Organization ISO може бути знайдена за адресою www. iso.ch. Багато інформації про стандарти є також на сторінці American National Standards Institute www.ansi.prg.

2.9. *Література до розділу 2*

1. Knuth, Donald E. The Art of Computer Programming,3rd ed. Reading, MA: Addison-Wesley, 1998.

2. Карцев М. А. Арифметика цифровых машин. - М.: Наука, 1969.

3. Каган Б. М. Электронные вычислительные машины и системы. - М.: Энергия, 1979.

4. Рабинович З. Л., Раманаускас В. А. Типовые операции в вычислительных машинах. - К.: Техніка, 1980. - 308 с.

5. Корнейчук В. И., Тарасенко В. П. Основы компьютерной арифметики. - К. Корнейчук, 2002. - 176 с.

6. Goldberg, David. "What Every Computer Scientist Should Know About Floating-Point Arithmetic." A C M Computing Surveys 23:1 March 1991. pp. 5-47.

2. ТО. *Питання до розділу 2*

1. Що таке система числення?
2. Що таке позиційна система числення?
3. Що таке основа системи числення?
4. Чому двійкові і десяткові системи числення названі позиційними?
5. Запишіть довільне число в позиційній системі числення та його кількісний еквівалент.
6. Поясніть зв'язок вісімкової та шістнадцяткової систем числення з двійковою системою числення.
7. Поясніть правило переведення чисел із системи числення з основою k до десяткової.
8. Поясніть правило переведення чисел із десяткової системи числення до системи числення з основою k.
9. Що означають слова біт, байт, слово?
10. Назвіть три способи представлення двійкових чисел із знаком в комп'ютерах, і поясніть їх відмінності.
11. Поясніть суть оберненого коду представлення двійкових чисел із знаком.
12. Поясніть суть доповняльного коду представлення двійкових чисел із знаком.
13. Поясніть суть прямого коду представлення двійкових чисел із знаком.
14. Яке саме з трьох цілочисельних представлень використовується частіше всього в комп'ютері?
15. Який діапазон представлення двійкових чисел із знаком в прямому, оберненому та доповняльному кодах?
16. Що таке переповнення, і як воно може бути виявлене?
17. Як переповнення для чисел без знаку відрізняється від переповнення для чисел із знаком?
18. Назвіть три способи представлення числових даних в комп'ютері.
19. Що означає представлення числа з одинарною та з подвійною точністю?
20. Де розміщується кома при представленні чисел з фіксованою комою?
21. На якій позиції розміщується кома при представленні цілих чисел?

22. На якій позиції розміщується кома при представленні дробових чисел?
23. Яке найбільше ціле додатне число може бути представлене в n-розрядній сітці?
24. Яке найбільше дробове додатне число може бути представлене в n-розрядній сітці?
25. Яке найменше ціле додатне число може бути представлене в n-розрядній сітці?
26. Яке найменше дробове додатне число може бути представлене в n-розрядній сітці?
27. Яке найбільше ціле від'ємне число може бути представлене в n-розрядній сітці?
28. Яке найбільше дробове від'ємне число може бути представлене в n-розрядній сітці?
29. Яке найменше ціле від'ємне число може бути представлене в n-розрядній сітці?
30. Яке найменше дробове від'ємне число може бути представлене в n-розрядній сітці?
31. Які є три складові частини чисел з рухомою комою?
32. Що таке зміщений порядок і яка мета його застосування?
33. Яка перевага використання зміщення взамін знакового біта в порядку?
34. Які найбільші та найменші додатні і від'ємні числа можуть бути представлені в форматі IEEE-754?
35. Чому використовується представлення чисел з рухомою комою в нормалізованій формі?
36. Що таке нормалізація?
37. Чому упускається одиниця старшого розряду нормалізованої мантиси при зберіганні числа з рухомою комою?
38. Скільки розрядів має число з рухомою комою в форматі IEEE-754 з одинарною точністю?
39. Скільки розрядів має число з рухомою комою в форматі IEEE-754 з подвійною точністю?
40. Які є переваги та недоліки використання відмінної від 2 основи порядку для чисел з рухомою комою?
41. Назвіть особливості виконання операцій над числами з рухомою комою.
42. Коли використовується представлення чисел з поблоково-рухомою комою?
43. В яких випадках використовується формат представлення чисел з рухомою-рухомою комою?
44. Поясніть суть кодування чисел кодом BCD.
45. Приведіть двійково-десятковий код з 2 з 5 та назвіть вигоди від його використання.
46. Приведіть двійково-десятковий код з надлишком 3 та назвіть вигоди від його використання.
47. Поясніть суть кодування чисел кодом EBCDIC.
48. Що таке код ASCII і чим він відрізняється від коду BCD?
49. Скільки розрядів використовується в коді Unicode для представлення символу?
50. Чому був запроваджений стандарт кодування символів Unicode?

2.11. Задачі до розділу 2

1. Запишіть довільне число в двійковій системі числення та його кількісний еквівалент.
2. Запишіть довільне число в вісімковій системі числення та його кількісний еквівалент.
3. Запишіть довільне число в шістнадцятковій системі числення та його кількісний еквівалент.
4. Виконати наступні перетворення, використовуючи віднімання або ділення з остачею:

a. 658_{10} =_____;

b. $477_{...}$ =_____,

c 518_{10} = _

d. 5401_{10} = .

5. Виконати наступні перетворення, використовуючи віднімання або ділення з остачею:

a. 488.. =

b. 5254_{10} =
c. 752_{10} =
d. 6104_{10} =

7
9

6. Перевести наступні десяткові дробові числа в двійкові, обмежившись шістьма розрядами справа після коми:
   a. 36.78125
   b. 294.03125
   c. 498.796875
   d. 26.1240234375
7. Перевести наступні десяткові дробові числа в двійкові, обмежившись шістьма розрядами справа після коми:
   a. 35.84375
   b. 47.55
   c. 50.90625
   d. 74.874023
8. Представити наступні десяткові числа 8-розрядними двійковими із знаком в оберненому і доповняльному кодах:
   a. 67
   b. 52
   c. 219
   d. 207
9. Використовуючи 3-розрядні слова, назвати всі можливі двійкові числа із знаком та їх десяткові еквіваленти при їх представленні в:
   a. Прямому коді
   b. Оберненому коді
   c. Доповняльному коді
10. Використовуючи 4-розрядні слова, назвати всі можливі двійкові числа із знаком та їх десяткові еквіваленти при їх представленні в:
   a. Прямому коді
   b. Оберненому коді
   c. Доповняльному коді
11. Визначити значення двійкового числа із знаком 1001, представленого в оберненому коді, в десятковій системі.
12. Для числа з рухомою комою, яке має 3-бітовий порядок і 5-бітову мантису із знаком знайти:
   a. Найбільше додатне та найменше від'ємне нормалізовані числа.
   b. Зміщення порядку, при якому всі значення порядку є від'ємними.
13. Використовуючи 14-розрядний формат числа, причому 5 розрядів - для представлення порядку, 8 розрядів - для представлення нормалізованої мантиси та один для її знаку:
   a. Показати, як в цьому форматі будуть представлені числа 100.0 та 0.25.
   b. Додати ці два числа в названому форматі.
14. Які найбільші та найменші додатні і від'ємні числа можуть бути представлені в форматі IEEE-754?
15. Записати в десятковій системі 32-розрядне число F0ABCD78, представлене в форматі IEEE з рухомою комою. Тут для запису числа використана шістнадцяткова система.
16. Показати, як в форматі IEEE-754 з рухомою комою можуть бути представлені наступні числа: +1;-1; $356*2^{33}$.
17. Записати число N = -1 в форматі IEEE-754.
18. Записати число N = -1.5 в форматі IEEE-754.

19. Описати діапазони представлення чисел при використанні форматів з фіксованою і рухомою комою

20. Вибрати формат представлення даних який забезпечує точність до 8 десяткової цифри та динамічний діапазон від 10^{25} до 10^{25}. Тут число в дужках показник степеня

21. Приведіть діапазони представлення чисел при використанні відмінної від 2 основи порядку для формату з рухомою комою

22. Показати як в форматі IEEE-754 з рухомою комою можуть бути представлені наступні числа:+1;-1; 356*2"

23. Вибрати формат представлення даних, який забезпечує точність до 12 десяткової цифри та динамічний діапазон від 10^{-22} до 10^{22}. Тут число в дужках показник степеня

24. Декодувати наступний лист в 7-розрядному АБСІІ коді без біта парності: 1001010 1001111 1001000 1001110 0100000 1000100 1000101

Розділ 3

і пртрам & комп'ютері

В цьому розділі розглядаються формати і типи команд, способи кодування та виконання команд в комп'ютері, включаючи конвеєризацію виконання команд - один з типів паралелізму на рівні команди, який може підвищити продуктивність виконання послідовності команд при відсутності конфліктів в конвеєрі. Наводиться класифікація архітектури комп'ютера за типом адресованої пам'яті: стекова, акумуляторна, та на основі регістрів загального призначення. Для кожної архітектури аналізуються переваги і недоліки, які розглядаються в контексті застосування запропонованої архітектури. Розглядаються різні способи адресації, включаючи безпосередню, пряму, непряму, базову, індексну, сторінкову і стекову. Наявність множини способів забезпечує гнучкість і зручність для програміста.

Архітектура комп'ютера розглядається на рівні системи команд, який видимий програмісту, що працює на мові асемблера та розробнику компіляторів, що дозволяє встановити межу між апаратним і програмним забезпеченням. Проводиться поділ комп'ютерів за складом системи команд на наступні типи: комп'ютери з складною, з простою, з доповненою та спеціалізованою системою команд.

3.1. Кодування та *виконання команд в комп'ютері*

Більшість сучасних комп'ютерів працюють за принципом програмного керування, згідно з яким над даними виконуються операції, тип яких вказується командами, які зберігаються в тій же пам'яті, що і дані. Послідовність команд, за яким виконується задача, називається програмою. Для того, щоб виконати на комп'ютері задачу, необхідно:

- забезпечити вибірку команд програми із його пам'яті в заданій послідовності, організувавши звернення до неї за відповідними адресами;
- забезпечити розпізнавання типів виконуваних операцій;
- організувати звернення до пам'яті за відповідними адресами для вибірки необхідних для виконання кожної команди даних;
- організувати виконання над даними операцій відповідно до вказівок команд;
- запам'ятати результати обчислень.

Розглянемо як це відбувається в комп'ютері детальніше.

3.1.1. Кодування команди та програми

Команда в комп'ютері зберігається в двійковій формі. Вона вказує тип операції, яка має бути виконаною, адреси операндів, над якими виконується операція, та адреси розміщення результатів виконання операції. Відповідно до цього команда складається з двох частин, як це показано на рис. 3.1: коду операції та адресної частини.

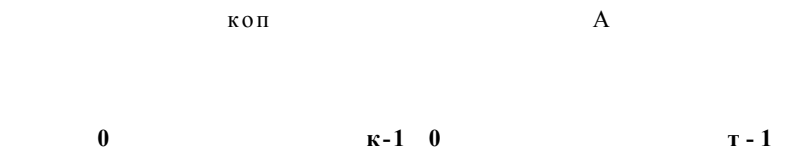


Рис. 3.1. Кодування команди

Поле коду операції (КОП) займає k розрядів. Ним може бути закодовано до $N = 2^k$ різних операцій. Кількість двійкових розрядів, які відводяться під код операції, вибирається таким чином, щоб ними можна було закодувати всі виконувані в даному комп'ютері операції. Якщо деякий комп'ютер може виконувати Nc різних операцій, то мінімальна розрядність поля коду операції k визначається наступним чином: $k = [\log Nc]$, де вираз в дужках означає заокруглення до більшого цілого.

Поле адреси (адресна частина) займає m розрядів. В ньому знаходяться адреси операндів. Кожна адреса займає пт. розрядів, де і - номер адреси (i=1,2,...1), 1 - кількість адресних полів. Кожною адресою можна адресувати пам'ять ємністю $2^{т_i}$ слів. Детальна інформація про зв'язок між ємністю пам'яті та розрядністю адреси наведена в розділі 9, в якому описана будова пам'яті.

Розмір команди k + m повинен бути узгодженим з розміром даних, тобто бути з ним однаковим або кратним цілому числу, що спрощує організацію роботи з пам'яттю. Як правило, розмір команди рівний 8, 16, 32 біти.

При написанні програми крім двійкової можуть використовуватись й інші форми представлення команд: вісімкова, шістнадцяткова, символьна (мнемонічна).

Використання вісімкового і шістнадцяткового кодування дозволяє скоротити записи і спростити роботу програміста. Як відомо 3 двійкових розряди (тріада) замінюються на 1 вісімковий, а 4 двійкових розряди (тетрада) - на 1 шістнадцятковий. Приклад: $(000011111111)_2 = (0377)_8 = (0FF)_{16}$

Мнемонічне кодування спрощує процес написання, читання і відлагодження програми. Основний принцип такого кодування - кожна команда представляється 3-х або 4-х буквеним символом, який показує назву команди. Деякі приклади мнемонічного кодування:

ADD - додати (add),

SUB - відняти (subtract),

MPY - перемножити (multiply),

DIV - поділити (divide),

LOAD - зчитати дані з пам'яті (load data from memory),

STORE - записати дані в пам'ять (store data to memory).

Операнди також представляються символічно. Наприклад команда ADD R, Y означає додавання вмісту комірки пам'яті Y до вмісту регістра R. Зауважимо, що операція виконується над вмістом, а не над адресою комірки пам'яті та регістра.

Таким чином, з'являється можливість написання машинних програм в символічній формі. Повний набір символічних назв і правила їх використання утворюють мову програмування, відому як асемблерна мова. Символічні імена називаються мнемонічними, а правила їх використання для створення команд і програм називаються синтаксисом мови.

Програма, яка переводить із мнемонічного коду асемблерної мови в машинний, називається асемблером. Команди, які використовуються для переводу вихідної програми в асемблерну, називаються командами асемблера. Ці команди вказують як інтерпретувати назви, де розмістити програму в пам'яті, яка кількість комірок пам'яті необхідна для зберігання даних.

Асемблерна мова є дуже далекою від мови людини і заставляє програміста думати виходячи з принципів побудови комп'ютера. Тому були створені мови високого рівня та компілятори, які переводять програми з цих мов на мову асемблера. Використання мов високого рівня має цілий ряд переваг в порівнянні з використанням асемблера. По-перше, програміст пише програми на мові, близькій до його мови спілкування. Більше того, мови високого рівня орієнтуються на класи вирішуваних задач. По-друге, скорочується час написання програм. І по-третє, мови високого рівня є незалежними від типу та архітектури комп'ютера, що дозволяє використовувати написані на цих мовах програми на всіх комп'ютерах, а програміста звільнити від знання їх структури та організації роботи.

Разом з тим, хоча більшість програм сьогодні пишуться на мовах високого рівня, асемблерна мова є корисним засобом для написання машинних команд завдяки близькості до машинної мови, наочності та компактності, і ми також будемо її для цього використовувати.

3.7.2. *Порядок виконання команд*

Команди зберігаються в основній пам'яті комп'ютера за відповідними адресами. Для того, щоб виконати команду та здійснити обробку даних, команду та дані потрібно зчитати з основної пам'яті та заслати до відповідних регістрів процесора. Комп'ютер виконує кожну команду як послідовність простих операцій:

1. Вибірка чергової команди із основної пам'яті.
2. Визначення типу вибраної команди, тобто її дешифрування.
3. Визначення адрес даних, необхідних для виконання цієї команди.
4. Виконання операцій пересилання даних (зчитування даних із пам'яті в регістри процесора).
5. Виконання операції відповідно до її коду в полі коду операції команди.
6. Визначення адрес, за якими запам'ятовуються результати.
7. Запам'ятовування результатів.
8. Підготовка до виконання наступної команди, тобто обчислення її адреси.

На рис. 3.2 показана діаграма циклу виконання команди, причому в нижній стрічці наведені операції, які виконуються всередині процесора, а в верхній стрічці - операції запису та вибірки із основної пам'яті. Операції 3 та 4 можуть повторюватись стільки

разів, скільки потрібно вибрати операндів з основної пам'яті. Така ситуація відбувається при виконанні багатомісних операцій. Аналогічно можуть повторюватись операції 6 та 7, якщо отримано кілька результатів. При обробці декількох даних за однією командою операції 3-7 повторюються відповідну кількість разів. Такі операції називають векторними.

Рис. 3.2. Діаграма циклу виконання команди

3.1.3. Виконання команд на рівні регістрів процесора

Для глибшого розуміння послідовності виконання команди розглянемо детальніше структуру регістрової (надоперативної) пам'яті процесора. Ця пам'ять (рис. 3.3) складається з регістрів з закріпленими операціями, та регістрів зального призначення. Тут РгА, РгК і РгД - відповідно регістри адреси, команд і даних. РгА зберігає адресу даного або команди при зверненні до основної пам'яті. РгД зберігає операнд при його запису або зчитуванні з основної пам'яті. В ролі операнда може бути дане, команда або адреса. РгК зберігає команду після її зчитування з основної пам'яті. ПЛ - програмний лічильник, який підраховує команди та зберігає адресу поточної команди. Комп'ютер з архітектурою Джона фон Неймана має один програмний лічильник.

РгА	РгД
ПЛ	РгО
	Рг і
РгК	Рг(п-І)
РгССП	п регістрів загального призначення

Рис. 3.3. Регістрова пам'ять процесора

Більшість комп'ютерів мають в складі процесора тригери для зберігання бітів стану процесора, або, як їх іще називають, прапорців. Кожен прапорець має спеціальне призначення. Частина прапорців вказує на результати арифметичних і логічних операцій: додатній результат (Р), від'ємний результат (N), нульовий результат (Z), перенос (С), арифметичне переповнення (V), і т. д. В системі команд комп'ютера є команди, які вказують процесору коли встановити чи скинути ці тригери. Інша частина прапорців вказує режими захисту пам'яті. Існують також прапорці, які вказують пріоритети виконуваних програм. В деяких процесорах додаткові тригери служать для зберігання кодів умов, формуючи регістр кодів умов. Взяті разом описані прапорці формують слово стану програми (ССП), а відповідні тригери - регістр ССП.

Регістри загального призначення (РЗП) є програмно доступними. Зазвичай їх називають регістровим файлом. Вони можуть використовуватись програмістом в якості регістрів для зберігання вхідних та вихідних даних, а також проміжних результатів обчислень, в якості адресних та індексних регістрів при виконанні операцій модифікації адрес. Наприклад, в процесорі UltraSPARC II є дві групи регістрових файлів: 32 64-розрядні регістри загального призначення та 32 регістри для даних з рухомою комою, які можуть зберігати або 32-розрядні дані одинарної точності, або 64-розрядні дані подвійної точності. В процесорі Pentium II є лише 8 32-розрядних та 6 16-розрядних регістрів загального призначення.

Зв'язки між вузлами процесора і основною пам'яттю показано на рис. 3.4. Як видно з рисунка, процесор взаємодіє з основною пам'яттю через регістри адрес та даних. Крім того, пристрій керування формує сигнали задання режимів роботи пам'яті.

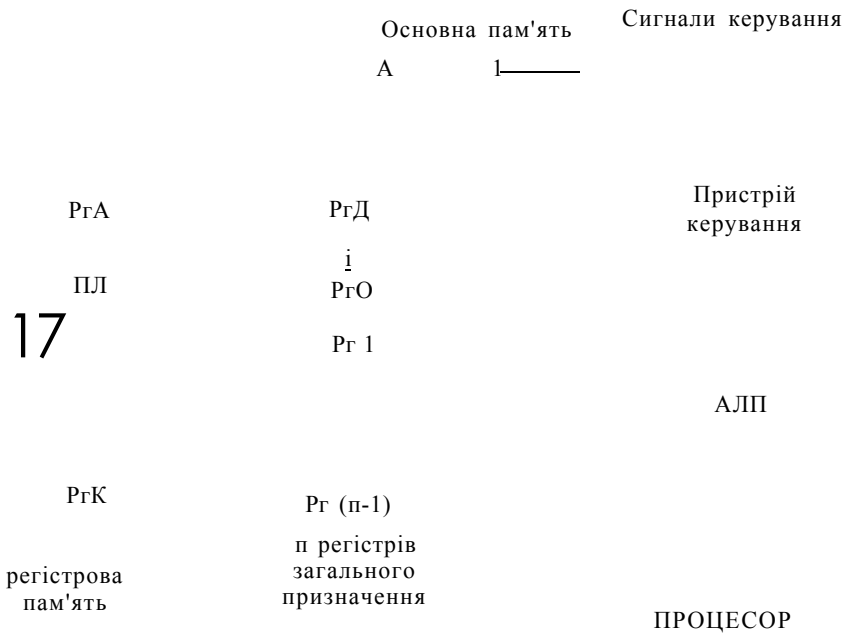


Рис. 3.4. Зв'язки між процесором і основною пам'яттю

Виходячи з наведеної вище інформації про вузли процесора та його зв'язки з основною пам'яттю, розглянемо детальніше виконання команд на прикладі виконання операції додавання слова, яке знаходиться в пам'яті за адресою 9, з вмістом регістра РгО регістрової пам'яті процесора, коли результат операції засилається в пам'ять за адресою 9. Програма обчислень знаходиться також в пам'яті. Операцію можна записати наступним чином:

[Комірка 9] := [Комірка 9] + [РгО].

Послідовність дій при виконанні цієї операції буде наступною:

1. Рг А :- ПЛ. Значення із програмного лічильника, тобто адреса команди, записується в регістр адреси РгА.

2. Зчитування із комірки [РгА] основної пам'яті команди додавання двох чисел в регістр даних РгД.

3. Рг К := Рг Д. Перезапис команди додавання двох чисел з регістра даних в регістр команди РгК.

4. РгА := [А РгК]. Запис адреси числа із регістра команди до регістра адреси (ця адреса рівна 9).

5. Зчитування із комірки [РгА] основної пам'яті даного і засилання його в регістр РгД.

6. Рг Д := [РгД] + [РгО]. Виконання в АЛП операції [РгД] + [РгО] і засилання результату в РгД.

7. Запис в комірку 9 основної пам'яті даного із регістра РгД.

8. ПЛ := ПЛ +1. Прирощення на одиницю вмісту програмного лічильника.

Подібним чином виконуються інші команди, включаючи команди взаємодії з пристроями введення-виведення.

3.2. Типи операцій та команд

*3.2.*1. *Класифікація команд за типами операцій*

Команди можуть бути класифіковані відповідно до ініційованих ними типів операцій.

Команди обробки даних ініціюють:

- арифметичні операції (додавання, віднімання, множення та ділення) над скалярними, тобто одиночними даними, та над векторами даних (деякою кількістю даних);
- логічні операції (логічне множення, додавання, інверсія, і т.д.) над окремими розрядами даного, скалярними даними та над векторами даних;
- операції зсуву (вправо, вліво) над скалярними та над векторними даними;
- операції перетворення даних (перетворення із формату з фіксованою в формат з рухомою комою і навпаки, і т. д.);
- операції над символами та стрічками символів.

Команди переміщення даних, включаючи команди звертання до пам'яті, ініціюють:

- операції переміщення даних в регістрах та стеках над скалярами та векторами;
- команди вибірки даних з пам'яті та запам'ятовування даних в пам'яті;
- " команди вибірки адрес з пам'яті та запам'ятовування адрес в пам'яті;
- команди вибірки команд з пам'яті та запам'ятовування команд в пам'яті.

Команди передачі керування змінюють логічний потік ходу програми. До них належать наступні команди: переходу; розгалуження, шляхом виконання операцій порівняння та перевірки; звернення до підпрограм.

Команди введення-виведення ініціюють операції введення та виведення даних та команд.

Розглянемо деякі типи команд детальніше.

3.2.2. *Команди обробки даних*

Як вже було зазначено, команди обробки даних в першу чергу ініціюють виконання арифметичних операцій. Це операції додавання, віднімання, множення та ділення над числами, представленими в форматах з фіксованою та рухомою комою. При цьому арифметичні операції можуть виконуватись як над скалярними даними, так і над векторами даних.

До арифметичних належать також наступні операції над одиночними операндами: взяття абсолютної величини від операнда (absolute); інверсія знаку операнда (negate); прирощення операнда на одиницю (increment); зменшення операнда на одиницю (decrement).

Команди обробки даних ініціюють також виконання логічних операцій. До їх числа входять операції логічного множення, логічного додавання, додавання за модулем два, інверсія, і т. д. Логічні операції виконуються над окремими розрядами даних, над одиночними даними і над векторами даних.

До складу команд обробки даних більшості комп'ютерів входить велика кількість операцій зсуву над скалярними даними та векторами даних, які будуть детально розглянуті в розділі 3.

Операції перетворення даних змінюють формат представлення даних. Це може бути перетворення із формату з фіксованою в формат з рухомою комою і навпаки, перетворення із двійкової в десяткову систему і т. д.

Операції над символами та стрічками символів передбачають обробку символьних даних і будуть розглянуті далі.

В табл. 3.1 наведено перелік та функції арифметичних та логічних команд комп'ютера DLX, запропонованого Д. Паттерсоном та Д. Хеннессі в якості навчальної моделі сучасного комп'ютера.

Таблиця 3.1

№ пп	Код команди	Функції команд комп'ютера DLX
Арифметичні та логічні		Операції над цілими та логічними даними в регістрах цілих чисел; знакова арифметика з фіксацією переповнення
1	ADD, ADDI, ADDU, ADDUI	Додати, додати безпосереднє дане (immediate), з знаком та без знаку
2	SUB, SUBI, SUBU, SUBUI	Відняти, відняти безпосереднє дане, з знаком та без знаку
3	MULT, MULTU, DIV, DIVU	Перемножити та поділити, з знаком та без знаку
4	AND, ANDI	Додати, додати безпосереднє дане

5	OR, ORI, XOR, XORI	Логічне АБО, логічне АБО над безпосереднім даним, виключне АБО, виключне АБО над безпосереднім даним
6	LHI	Зчитування старших 16 розрядів безпосереднього даного та 16 нулів молодших розрядів
7	SLL, SRL, SRA, SLLI, SRLI, SRAI	Зсуви направо та наліво логічні та арифметичні, включаючи безпосереднє дане
8	SEQ, SXE, SLT, SGT, SLE, SGE	Встановити за умови, якщо рівне нулю, не рівне нулю, менше ніж, більше ніж, менше ніж або рівне, більше ніж або рівне
Рухома кома		Операції з рухомою комою
9	ADDD, ADDF	Додати з подвійною точністю та з рухомою комою
10	SUBD, SUBF	Відняти з подвійною точністю та з рухомою комою
11	MULTD, MULTF	Помножити з подвійною точністю та з рухомою комою
12	DIVD, DIVF	Поділити з подвійною точністю та з рухомою комою
13	CVTD2F, CVTD2I, CVTF2D, CVTF2I, CVTI2D, CVTI2F	Перетворення з формату з подвійною точністю до формату з рухомою комою, до формату з одинарною точністю та навпаки
14	EQD, EQF, NED, NEF, LTD, LTF, GTD, GTF, LED, LEF, GED, GEF	Порівняння даних в форматах з одинарною та подвійною точністю та записати результат до регістру станів

3.2.3. Команди переміщення даних

Команди переміщення даних належать до базових команд комп'ютера. Вони здійснюють передачу даних з одного місця в інше. Ці команди вказують:

* місце розміщення операндів - основна пам'ять чи регістр;
- адреси розміщення операндів в основній пам'яті або в регістровому файлі;
- методи адресації кожного операнда;
- кількість даних, що підлягають переміщенню;
- розрядність даних, які мають бути передані.

В системах команд різних комп'ютерів це зроблено по різному. Наприклад, в деяких комп'ютерах місце розміщення операндів вказується в полі коду операції, в інших - в адресному полі. В табл. 3.2 як приклад наведені команди переміщення даних комп'ютера IBM S/370.

Таблиця 3.2

Мнемонічний код операції	Ім'я операції	Розрядність даних	Опис операції
L	Load	32	Передача із пам'яті в регістр
LH	Load Halfword	16	Передача із пам'яті в регістр
LR	Load	32	Передача із регістра в регістр
LES	Load (Short)	32	Передача із регістра з РК в регістр з РК
LTS	Load (Short)	32	Передача із пам'яті в регістр з РК
LDR	Load (Long)	64	Передача із регістра з РК в регістр з РК
LD	Load (Long)	64	Передача із пам'яті в регістр з РК
ST	Store	32	Передача із регістра в пам'ять
STH	Store Halfword	16	Передача із регістра в пам'ять
SOC	Store Character	8	Передача із регістра в пам'ять
STE	Store (Short)	32	Передача із регістра з РК в пам'ять
STD	Store (Long)	64	Передача із регістра з РК в пам'ять

Тут наведено два типи команд переміщення - зчитування даних (Load) та запам'ятовування даних (Store). Вказується також розмір даних: character - 8-розрядні дані, half-word - 16-розрядні дані, short - 32-розрядні дані, long - 64-розрядні дані. Для задання номера регістра в регістровому файлі процесора та номера комірки основної пам'яті, при написанні команди ці номери (адреси) записуються справа від мнемонічного коду операції, наприклад L105.R4 означає передачу даного з комірки пам'яті 105 до регістра R4. Потрібно зазначити, що в комп'ютері IBM S/370 є, крім регістрового файлу для цілочислових даних, також регістровий файл для даних з рухомою комою. Для позначення цих регістрів в табл. 3.2 використана абревіатура РК, що означає "рухома кома".

Для порівняння в табл. 3.3 як приклад наведені команди переміщення даних комп'ютера з спрощеною системою команд DLX. Видно, що тут додались команди обміну між регістрами з фіксованою та рухомою комою.

Таблиця 3.3

№ пп	Код команди	Функції команди DLX машини
Пересилання даних		Пересилання даних поміж регістрами та пам'яттю або поміж спеціальними регістрами та регістрами чисел з фіксованою та рухомою комою; тільки один метод адресації: 16-бітовий зсув + вміст регістра чисел з фіксованою комою
1	LB, LBU, SB	Вибірка байта, вибірка байта без знаку, запис байта
2	LH, LHU, SH	Вибірка півслова, вибірка півслова без знаку, запис півслова
3	LW, SW	Вибірка слова, запис слова
4	LF, LD, SF, SD	Вибірка даного з рухомою комою з одинарною точністю, вибірка даного з рухомою комою з подвійною точністю, запис даного з рухомою комою з одинарною точністю, запис даного з рухомою комою з подвійною точністю
5	MOVI2S, MOVS2I	Перемістити з/до регістра з фіксованою комою до/з спеціального регістра
	MOVF, MOVD	Перемістити з регістра з рухомою комою чи з подвійною точністю до іншого регістра чи пари регістрів
	MOVFP2I, MOVI2FP	Перемістити з регістра з рухомою комою чи з фіксованою комою до іншого регістра з фіксованою комою чи з рухомою комою

3.2.4. *Команди передачі керування*

Як уже зазначалося, кожна команда має дві фази виконання. Перша фаза - це вибірка команди. На цій фазі за вмістом ПЛ з пам'яті в РгК вибирається команда. На другій фазі, яка називається виконанням команди, дешифрується код операції і виконується команда, тобто із основної пахм'яті вибираються операнди, виконується арифметична чи логічна операція і запам'ятовуються результати обчислень. На другій фазі при послідовному виконанні команд вміст ПЛ збільшується на одиницю, та вказує адресу наступної команди. На рис. 3.5 показано виконання операції віднімання від вмісту комірки А основної пам'яті вмісту комірки В з запам'ятовуванням результату в комірці С, тобто (С) := (А) - (В), з використанням двоадресного формату, коли три команди розміщені в послідовних комірках пам'яті, і їх адреси зростають в порядку виконання команд, починаючи з деякої комірки основної пам'яті і.

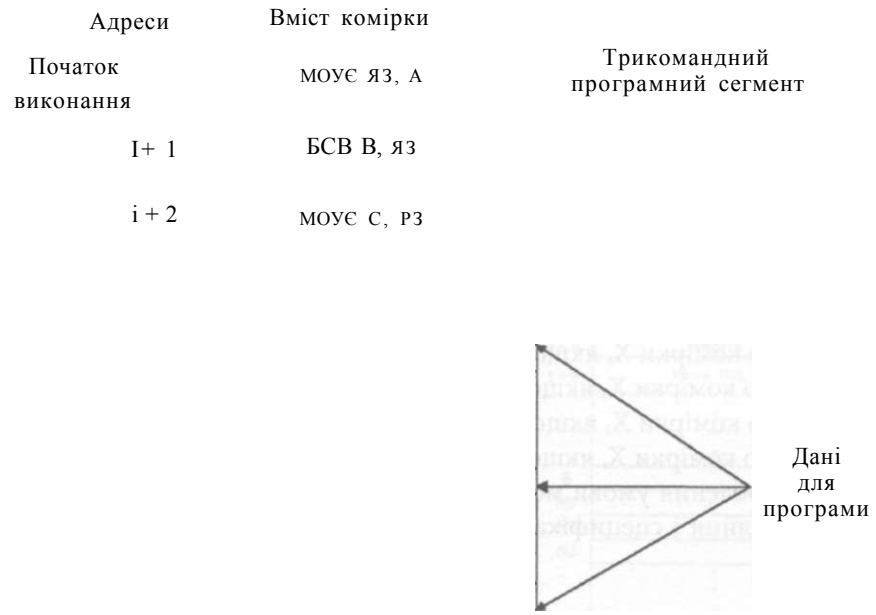


Рис. 3.5. Програма виконання операції (С) := (А) - (В)

Разом з тим, при виконанні більшості задач в комп'ютерах використовуються команди передачі керування, коли шляхом аналізу деяких умов вибирається один з набору можливих шляхів обчислень. При виконанні цих команд вміст програмного лічильника змінюється на адресу деякої комірки пам'яті.

Існує декілька причин необхідності виконання операцій передачі керування. До найважливіших з них належать наступні:

- Значна частина операцій в комп'ютері виконується багаторазово. Для вирішення деякої задачі може виникнути потреба в виконанні тисяч або й мільйонів команд. Писати та зберігати в пам'яті ті ж команди багаторазово є нелогічним та й збитковим. Тому, коли обробляється вектор даних чи список символів необхідно використати програмний цикл, коли виконується повторно та ж сама послідовність команд для обробки всіх операндів.
- Практично всі програми потребують прийняття деякого рішення - комп'ютер повинен виконати одні дії при наявності одних умов, та інші дії при наявності інших умов.
- Коректне компонування великих і навіть середнього розміру програм є достатньо важкою задачею. Наявність механізму її представлення малими кусками є дуже корисним.

Розглянемо організацію виконання основних команд передачі керування детальніше.

3.2.4.1. Команди переходу

Команди переходу мають в якості одного з операндів адресу команди, яка має бути виконана наступною. Є два типи команд переходу - умовного та безумовного. При безумовному переході відразу здійснюється перехід до вказаної в адресному полі команди. Найчастіше ці команди є командами умовного переходу, тобто перехід здійснюється за

адресою, вказаною операндом, тільки при виконанні заданих умов. В іншому випадку виконується наступна за даною команда, тобто, як зазвичай, значення програмного лічильника збільшується на одиницю.

Є два шляхи вироблення умови, яка буде перевірятись командою умовного переходу. Перший шлях вироблення умови передбачає формування коду умови, як результату виконання деяких операцій. Цей код може бути поміщений в доступний користувачеві регістр. Наприклад, арифметичні операції додавання, віднімання і т. д. можуть формувати дворозрядний код умови, який приймає одне з наступних чотирьох значень: нуль, додатне, від'ємне, переповнення. В такому комп'ютері буде чотири різних команди умовного переходу:

ВІДР X - перехід до комірки X, якщо результат додатній;

ВИХ X - перехід до комірки X, якщо результат від'ємний;

В±Ч2 X - перехід до комірки X, якщо результат рівний нулю;

ВІЮ X - перехід до комірки X, якщо відбулось переповнення.

Другий шлях вироблення умови може бути використаним в триадресному форматі команди, коли порівняння і специфікація переходу задається в самій команді. Наприклад команда

ВИЕ РчЗ, Я4, X

задає перехід до комірки X, якщо вміст регістра ЯЗ дорівнює вмісту регістра Я4.

На рис. 3.6 показано приклад програми, в якій використано безумовний перехід та умовні переходи з застосуванням обох описаних вище шляхів формування умови.

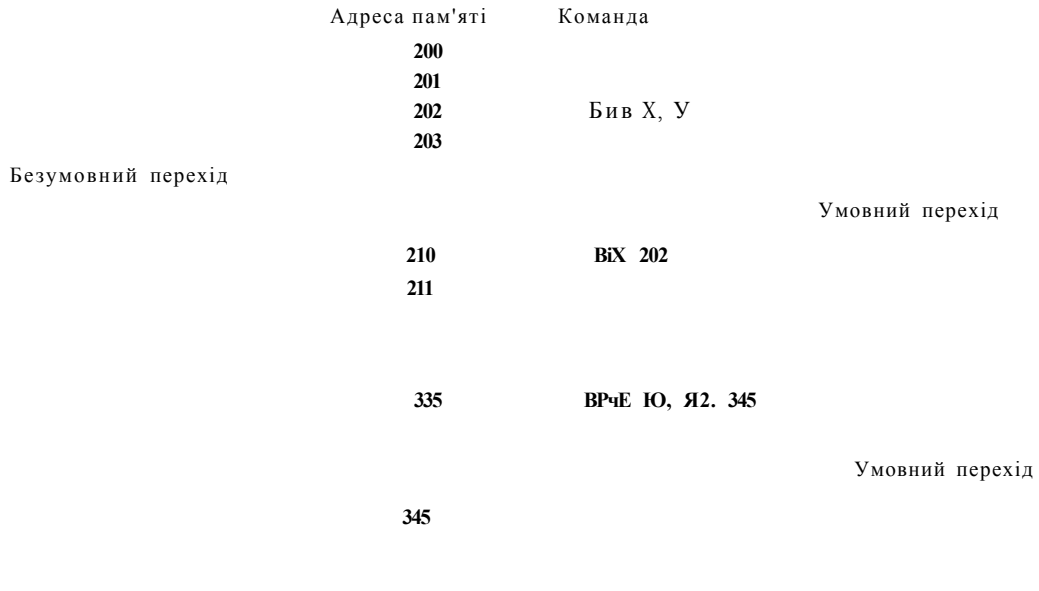


Рис. 3.6. Приклад операцій умовного та безумовного переходу

Зауважимо, що перехід може бути як в напрямку збільшення, так і в напрямку зменшення адреси. Приклад показує, як умовний та безумовний переходи можуть бути використані для створення повторюваних циклів команд. Команди від комірки 202 до комірки 210 будуть виконуватись повторно, аж поки результат віднімання У від X не стане рівним нулю.

Розглянемо приклад виконання задачі додавання п чисел, яку можна виконати шляхом послідовного виконання команд (рис. 3.7а), або з використанням команд переходів (рис. 3.7Ь). Тут М] *()* = 1,2,...,п) - адреса в пам'яті кожного із п чисел, і (i=1,2,...,п) - адреси розміщення команд, Б - адреса розміщення результуючої суми.

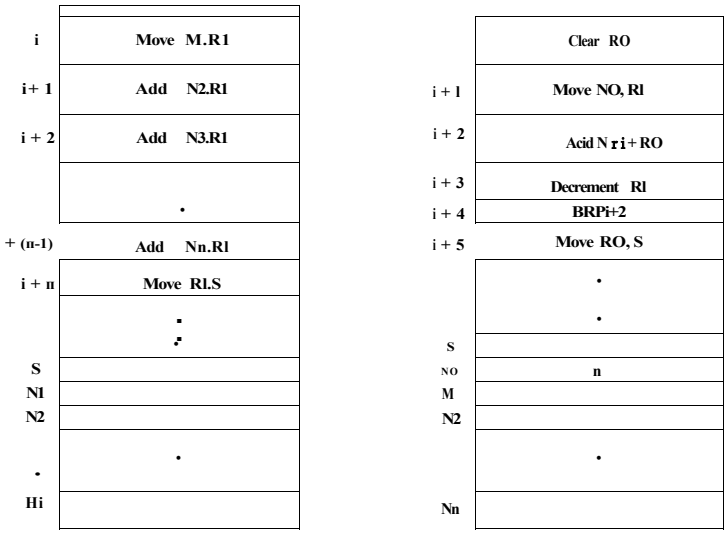


Рис. 3.7. Дві програми додавання п чисел шляхом послідовного виконання команд (а) та з використанням переходів (Ь)

Використання переходів вимагає введення додаткових команд очистки Clear RO, задання кількості п повторів виконання тіла програми, зменшення вмісту регістра R1, який використовується як лічильник, а також команди аналізу умови. Разом з тим, використання переходів зменшує об'єм програм, зокрема в наведеному прикладі взамін п+1 команд, де п - кількість чисел в масиві, використано лише 6 команд.

3.2.4.2. Команди пропуску

Команда пропуску включає передбачувану адресу. Типово пропуск передбачає, що одна команда буде пропущена. Ця команда не вимагає наявності адресного поля. Типовим її прикладом є команда increment-and-skip-if-zero (ISZ) - приріст на одиницю та перехід якщо нуль. Розглянемо наступний фрагмент програми (рис. 3.8).

	Адреса пам'яті	Команда	
	201		
Безумовний перехід			
	209	1SZR1	
	210	BR201	Пропуск
	211		

Рис. 3.8. Приклад використання команди пропуску

В цьому фрагменті використано команду пропуску для реалізації ітераційного циклу. В регістрі КЛ встановлюється від'ємне число, модуль якого рівний кількості необхідних ітерацій. В кінці циклу до вмісту регістра іЯІ додається одиниця. Якщо отримане значення не рівне нулю, програма переходить на початок циклу. В іншому випадку перехід пропускається і програма продовжується з наступної команди після кінця циклу.

3.2.4.3. Команди звернення до підпрограм

Одним з найбільших покращань в розробці мов програмування є підпрограми. Підпрограма є самостійною комп'ютерною програмою, яка, при необхідності, використовується іншою програмою. В якійсь точці програми підпрограма повинна бути викликаною та виконаною, після чого виконання програми має бути продовженим. Причиною використання підпрограм є економічна доцільність та потреба забезпечення поділу програми на незалежні модулі. Підпрограма дозволяє тій же частині коду бути використаною багаторазово. Це економить зусилля програміста та ємність пам'яті програм. Підпрограми також дозволяють розділити великі частини програми на малі блоки. Таке використання незалежних модулів суттєво спрощує написання та відлагодження програм.

Механізм підпрограм задіює дві базові команди: команду виклику, яка здійснює перехід з попередньої комірки до підпрограми, та команду повернення, яка здійснює повернення від підпрограми до місця, з якого вона була викликана. Обидві команди належать до команд переходу.

На рис. 3.9 показано як на основі підпрограм будувати програми.

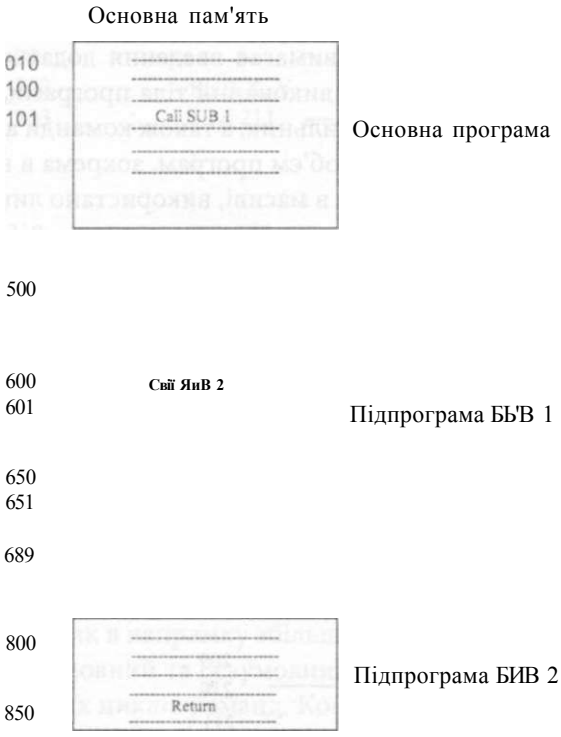


Рис. 3.9. Вкладені підпрограми

На цьому прикладі основна програма починається з комірки за адресою 010. Ця програма включає звернення до підпрограми SUBI, яка починається з комірки за адресою 500. Коли виконується ця команда виклику, процесор призупиняє виконання основної програми і починає виконання підпрограми SUBI шляхом вибору наступної команди з комірки 500. Всередині підпрограми SUBI є два звернення до підпрограми SUB2, яка починається з комірки за адресою 800. В обох цих випадках призупиняється виконання підпрограми SUBI і виконується підпрограма SUB2. Команда Return вказує процесору повернутись назад до програми SUBI та продовжити виконання команди, яка іде за відповідною командою виклику Call. Ці звернення між основною програмою та підпрограмами показані на рис. 3.10.

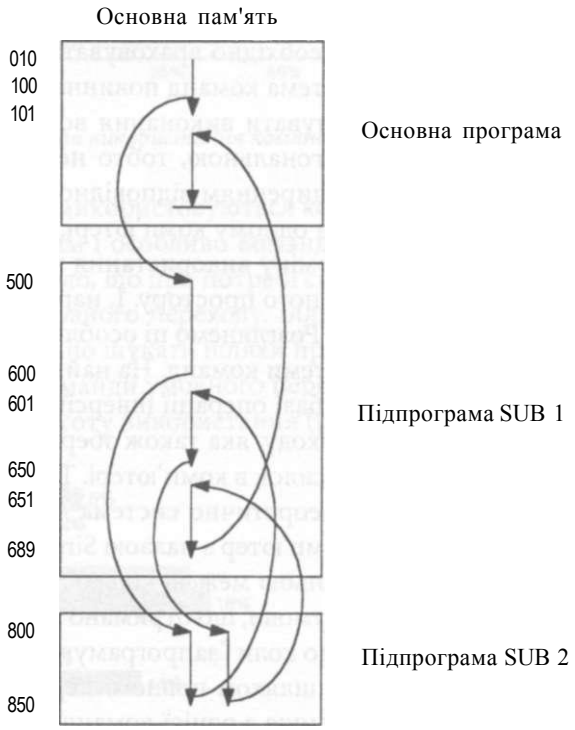


Рис. 3.10. Виконання послідовності вкладених підпрограм з рис. 3.16.

В табл. 3.4 як приклад наведені команди передачі керування комп'ютера з спрощеною системою команд БЬХ.

Таблиця 3.4

№ пп	Код команди	Функції команди DLX машини
Керування		Умовні та безумовні переходи; з використанням програмного лічильника або регістра
1	BEQZ, BNEZ	Перехід вміст регістра рівний/не рівний нулю; 16-розрядне зміщення до PC + 4
2	BFPF, BFPT	Тестове порівняння розряду в регістрі стану БР і перехід; 16-розрядне зміщення до PC + 4

3	J,JR	Jumps; 26-розрядне зміщення до PC + 4 (J) або цільового регістра (JR)
4	JAL, JALR	Jump and link; збереження PC+8 до R31, ціллю є 26-розрядне зміщення до PC + 4 (JAL) чи регістр (JALR)

3.2.5. *Команди введення-виведення*

Команди введення-виведення значно змінюються від одного комп'ютера до іншого. Базовими типами команд введення-виведення є програмоване введення-виведення, введення-виведення за перериваннями та прямий доступ до пам'яті. При розгляді цих способів введення-виведення будуть розглянуті і відповідні команди.

3.2.6. *Принципи формування системи команд комп'ютера*

При розробці комп'ютера необхідно враховувати багато особливостей вибору системи команд. З одного боку система команд повинна бути функціонально повною, тобто комп ютер повинен забезпечувати виконання всіх заданих функцій. З іншого боку система команд має бути ортогональною, тобто не повинна бути надлишковою. Для нового комп ютера, який є розширенням відповідної серії, першочерговою є сумісність програми, які виконуються на одному комп'ютері, повинні виконуватись і на іншому комп ютері. Для збільшення терміну використання важливим є розширення адресного діапазону та однорідності адресного простору. І, нарешті, важливим є розмір та організація самого формату команди. Розглянемо ці особливості детальніше

Є багато рівнів повноти системи команд. На найнижчому рівні всі комп'ютерні операції можуть бути виконані на базі операції інверсії логічного множення (NAND). Відповідно команда умовного переходу, яка також зберігає вміст програмного лічильника, забезпечує виконання всіх пересилок в комп'ютері. Тому можна створити повну систему команд на базі цих операцій. Теоретично система команд комп'ютера може включати лише одну команду. Відомий комп'ютер з назвою Single Instruction Computer (SIC), який є прикладом комп'ютера з нижньою межею кількості команд Єдиною командою SIC є команда "відняти та перейти за умови, що отримано негативний результат" (Subtract and Branch if Negative). Зрозуміло, що коли "запрограмувати" цією командою усі інші команди деякої реальної машини, то шляхом повного перебирання можна довести повноту системи команд, яку складено лише з однієї команди sbn. Недоліки SIC є зрозумілими, принаймні, на прикладі програмування однією командою операції множення (ділення тощо). Тобто програми на базі простих операцій є дуже складними.

Існує фундаментальний зв'язок між простотою процесора і складністю програми. Оскільки, як було показано вище, виконувати обробку даних з застосуванням однієї команди недоцільно, в реальних комп'ютерах застосовується система команд, до складу якої входить широкий спектр команд обробки даних, переміщення даних, передачі керування та введення-виведення

В ранніх комп'ютерах розробники старалися включити в систему команд максимальну їх кількість з тим, щоб забезпечити програмістам гнучкість та можливість вибору найдоцільнішої для конкретного випадку команди. Але з ростом досвіду та з проведених досліджень стало зрозумілим, що далеко не всі можливі команди доцільно включати в систему команд комп'ютера. Адже, з одного боку, це ускладнює формат команди, а в результаті і сам комп'ютер, а з іншого боку, багато команд дуже рідко використовуються

на практиці. Тому кращим є варіант реалізації таких команд на основі інших. Для прикладу на рис. 3.11 подано середню частоту використання команд передачі керування в різних задачах при обробці даних з фіксованою (темніший колір лінії) та з рухомою комою (світліший колір лінії).

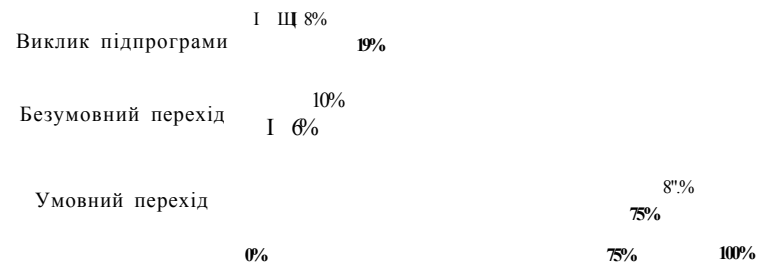


Рис. 3.11. Середня частота використання команд передачі керування

Як видно з рисунка, найчастіше використовуються команди умовного переходу, тоді як команди звертання до підпрограм і особливо команди безумовного переходу використовуються досить рідко. Зрозуміло, що при потребі скорочення кількості команд доцільно залишити саме команди умовного переходу. Більше того, зважаючи на частоту використання при реалізації, доцільно шукати шляхи прискорення виконання саме цих команд. Проаналізуємо далі самі команди умовного переходу, оскільки їх є декілька типів, і кожен тип також має свою частоту використання (рис. 3.12).

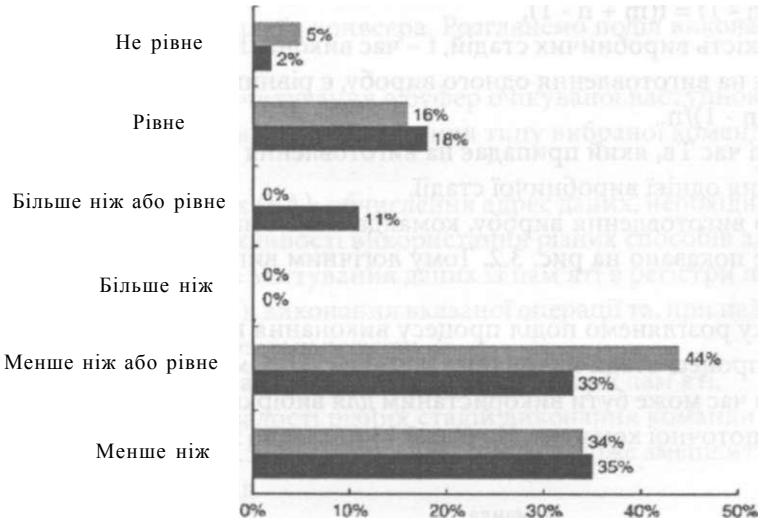


Рис. 3.12. Середня частота використання різних типів команд умовного переходу

Як видно з рисунка, деякі типи команд умовного переходу в вибраних тестових програмах не використовуються взагалі (команда „більше ніж" для чисел з фіксованою та рухомою комою, команда „більше ніж або рівне" для чисел з рухомою комою), або ви-

користовуються досить рідко. Оскільки ці команди взаємозамінні, немає сенсу реалізації їх всіх і доцільно обмежитись найуживанішими. Подібний аналіз проводиться і для інших типів команд з метою оптимізації системи команд комп'ютера.

Ортогональність характеризує незалежність команд. Набір команд є ортогональним, якщо існує тільки один простий шлях виконання набору операцій. Наприклад, для виконання арифметичних операцій можуть бути використані базові булеві операції, але час їх виконання в цьому випадку буде дуже великим. Це означає, що існування арифметичних операцій поряд з булевими не порушує ортогональності.

Існують спеціальні оптимізуючі компілятори, які слідкують за повнотою системи команд та їх ортогональністю.

Сумісність як і повнота та ортогональність має багато рівнів. Це може бути сумісність по вихідному коду, або по одному з об'єктних кодів. Близьким до сумісності є поняття взаємозамінності, тобто можливості виконання тієї ж програми на різних апаратних засобах.

3.2.7. Конвеєрне виконання команд

Конвеєрне виконання команд подібне до роботи конвеєра складальної лінії на заводі, наприклад автомобільному. На складальній лінії вироби проходять через однакові виробничі стадії. Одночасно на лінії знаходиться кількість виробів, рівна кількості виробничих стадій. Проходячи через всі виробничі стадії, виріб приймає кінцеві параметри. Час виготовлення одного виробу є рівним часу його проходження через всі виробничі стадії, але при виготовленні багатьох виробів, скажемо п, час, який припадає на виготовлення всіх виробів, є рівним:

Т = ітп + г(п - 1) = г(т + п - 1),

де т - кількість виробничих стадій, І - час виконання однієї виробничої стадії, а час, який припадає на виготовлення одного виробу, є рівним:

Тв = :(т + п - 1)/п.

При п >> т час Тв, який припадає на виготовлення одного виробу, наближається до часу 1 виконання однієї виробничої стадії.

Подібно до виготовлення виробу, команда також має кілька послідовних стадій виконання, як це показано на рис. 3.2. Тому логічним виглядає використання і тут принципу конвеєра.

Для початку розглянемо поділ процесу виконання команди на дві стадії: вибірку та виконання. В процесі стадії виконання команди є проміжки часу, коли немає звернень до пам'яті. Цей час може бути використаним для вибірки наступної команди паралельно з виконанням поточної команди. На рис. 3.13 показано цей підхід.

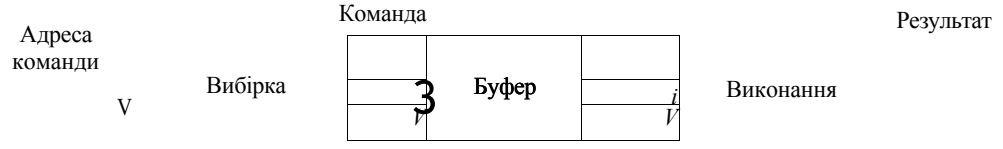


Рис. 3.13. Двоярусний конвеєр виконання команди

Конвеєр має два незалежних яруси. Перший ярус виконує операцію вибірки та буферизації (короткотермінового запам'ятовування) команди. Коли другий ярус звільняється від роботи, перший ярус передає йому буферизовану команду. Коли в другому ярусі виконується команда, в першому ярусі вибирається наступна команда. Така операція називається попередньою вибіркою команди (instruction prefetch) або суміщенням вибірки (fetch overlap).

Зрозуміло, що описаний процес прискорює виконання команди. Якби операції вибірки та виконання мали однаковий час виконання, то цикл виконання команди міг би бути зменшеним вдвоє. Однак це не зовсім так через наступні причини:

1. Стадія виконання значно довша стадії вибірки, оскільки вона вимагає виконання операцій зчитування та запису операндів та самої операції. Тому перший ярус повинен чекати деякий час, поки звільниться його буфер.

2. При появі команди умовного переходу адреса наступної команди до завершення поточної команди невідома. Тому перший ярус змушений чекати до завершення роботи другого ярусу. Після цього вже другий ярус повинен чекати на завершення роботи першим ярусом.

Час, який втрачається через другу причину, може бути зменшений шляхом використання механізму передбачення. Тут може бути використане наступне правило: коли команда умовного переходу поступає з ярусу вибірки на ярус виконання, в ярусі вибірки проводиться вибірка із пам'яті наступної команди після команди умовного переходу. Тоді в випадку відсутності умовного переходу втрат часу не буде. Коли ж буде умовний перехід, то вибрана команда повинна бути знехтувана і вибрана нова команда.

Хоча розглянуті дві причини знижують потенційну ефективність двоярусного конвеєра, в цілому виграш незаперечний. Для подальшого підвищення продуктивності потрібно збільшувати кількість ярусів конвеєра. Розглянемо поділ виконання команди на наступні стадії:

- Вибірка команди (ВК): зчитування в буфер очікуваної наступною команди.
- Дешифрування команди (ДК): визначення типу вибраної команди та специфікаторів операндів.
- Визначення адрес даних (ВА): обчислення адрес даних, необхідних для виконання команди з врахуванням можливості використання різних способів адресації.
- Вибірка операндів (ВО): зчитування даних із пам'яті в регістри процесора.
- Виконання команди (КВ): виконання вказаної операції та, при наявності, запам'ятовування результату в визначеному регістрі.
- Запис результату (ЗР): запам'ятовування результату в пам'яті.

При такому поділі час тривалості різних стадій виконання команди буде приблизно рівним. Тоді, як видно з табл. 3.5, шестиярусний конвеєр може зменшити час виконання 9 команд з 54 тактів до 14 тактів.

Таблиця 3.5

Номер такту	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Команда 1	ВК	ДК	ВА	ВО	КВ	ЗР								
Команда 2		ВК	ДК	ВА	ВО	КВ	ЗР							
Команда 3			ВК	ДК	ВА	ВО	КВ	ЗР						

Команда 4				ВК	дк	ВА	во	кв	ЗР					
Команда 5					ВК	ДК	ВА	во	кв	ЗР				
Команда 6						ВК	ДК	ВА	во	кв	ЗР			
Команда 7							ВК	ДК	ВА	во	кв	ЗР		
Команда 8								ВК	ДК	ВА	во	КВ	ЗР	
Команда 9									ВК	ДК	ВА	ВО	кв	ЗР

Часова діаграма в табл. 3.5 показує, що кожна команда виконується шляхом проходження через 6 ярусів конвеєра. Разом з тим, не для кожної команди це потрібно Наприклад, команда вибірки не вимагає виконання операції ЗО. Однак при її виконанні можна зробити шостий ярус конвеєра прозорим

Також на діаграмі показано, що всі стадії виконуються паралельно. В першу чергу тут прийнято, що не виникає конфліктів при зверненні до пам'яті. Наприклад, операції ВК, ВО та ЗО передбачають звернення до пам'яті. Діаграма допускає, що всі ці звернення можуть здійснюватись одночасно. Більшість систем пам'яті цього не допускають, тому звернення розносяться в часі. Іноді потрібне число може знаходитись в кеш пам'яті, а стадії ВО та ЗО відсутні. Тому конфлікти при зверненні до пам'яті не завжди сповільнюють конвеєр

Декілька інших факторів обмежують ріст продуктивності за рахунок використання конвеєра

- неоднаковість часу шести стадій виконання команди приводить до простою деяких з них, як це мало місце в двоярусному конвеєрі;
- поява команди умовного переходу може звести нанівець декілька вибірок команд
- подібною до команди умовного переходу є команда переривання

Табл. 3.6 відображає вплив умовного переходу при виконанні тих же операцій, що й в табл. 3.5.

Таблиця 3.6

Номер такту	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Команда 1	ВК	ДК	ВА	ВО	КВ	ЗР								
Команда 2		ВК	ДК	ВА	во	КВ	ЗР							
Команда 3			ВК	ДК	ВА	во	кв	ЗР						
Команда 4				ВК	дк	ВА	во							
Команда 5					ВК	ДК	ВА							
Команда 6						ВК	ДК							
Команда 7							ВК							
Команда 15								ВК	ДК	ВА	во	КВ	ЗР	
Команда 16									ВК	ДК	ВА	ВО	КВ	ЗР

Тут прийнято, що команда 3 є умовним переходом до команди 15. Поки команда 3 виконується, неможливо взнати, яка команда буде наступною. Конвеєр буде вибирати наступні команди (4,5,6,7) і виконувати їх. Наявність умовного переходу визначиться в кінці сьомого такту. Після цього конвеєр повинен звільнитись від непотрібних команд (очиститись). На 8 такті команда 15 поступить в конвеєр і дальше він почне заповнюватись знову. При цьому від 9 до 12 тактів не буде завершено виконання жодної команди. Це є розплата ефективністю за причини неможливості передбачити перехід.

На рис. 3.14 показано блок-схему виконання команди в шестиярусному конвеєрі з врахуванням переходів та переривань.

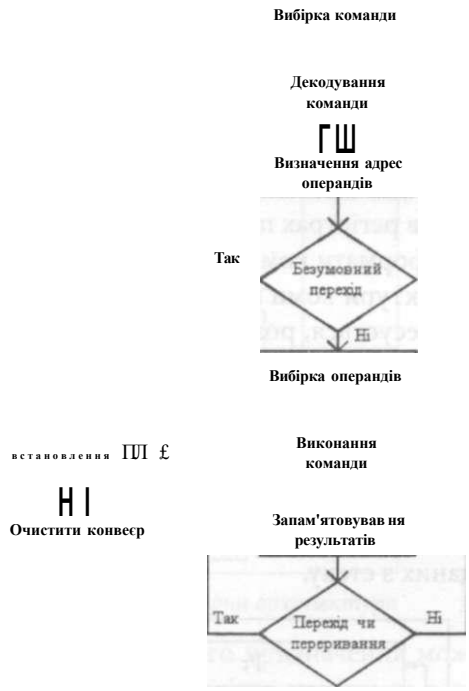


Рис. 3.14. Шестиярусний конвеєр команд

В шестиярусному конвеєрі команд появляється й інша проблема, якої не було в двоярусному конвеєрі. Стадія ВА може залежати від вмісту регістра, який змінюється попередньою командою, що знаходиться в конвеєрі. З'являється новий конфлікт, для усунення якого необхідна відповідна логіка.

Таким чином, конфлікти конвеєра можуть бути трьох типів:

- Конфлікт ресурсів, наприклад, одночасна потреба доступу до пам'яті. Цей конфлікт вирішується шляхом розділення доступу до ресурсу в часі, або шляхом введення додаткового ресурсу, наприклад, кількох блоків пам'яті.
- Залежність між даними, коли результат виконання деякої команди, яка іще не завершена, є операндом для наступної команди. Для подолання даного конфлікту існує декілька шляхів. Це може бути введення „пустої" операції пор (яка не виконує дій, але дозволяє ліквідувати конфлікт), введення зв'язків між ярусами конвеєра, що прискорює доступ до потрібного операнда, а також застосування компілятора, здатного передбачати такого типу конфлікти та перевпорядковувати команди програми.
- Наявність умовних переходів. В сучасних комп'ютерах для зменшення впливу на ефективність конвеєра цього конфлікту використовується спеціальна логіка передбачення переходу, яка дозволяє знайти вітку, якою програма піде після переходу. Інший підхід - виконання обох віток переходу до того часу, поки напрям переходу стане відомим.

В наступних розділах питання підвищення ефективності використання конвеєрної обробки буде розглянуто детальніше.

3.3. Формати команд комп'ютера

3.3.1. Класифікація архітектури комп'ютера за типом адресованої пам'яті

Як видно з рис. 3.1, крім коду операції до складу команди входить адресна частина. Цією частиною визначається місце знаходження даних, над якими виконується операція, задана кодом операції. Даних може бути декілька, і, крім того, вони можуть знаходитись в основній пам'яті, в регістрах процесора чи в запам'ятовуючих елементах інших вузлів комп'ютера. Тому і формати команд в цих випадках будуть різними. Можна здійснити класифікацію архітектури комп'ютера за типом адресованої пам'яті. Залежно від того, який тип пам'яті адресується, розрізняють наступні типи архітектур комп'ютера: стекова, акумуляторна, на основі регістрів загального призначення.

В стековій архітектурі (рис. 3.15) операнд завжди знаходиться в вершині стека - спеціальному регістрі пам'яті, з якого загружається в регістр процесора, або через нього результат операції загружається в пам'ять. Стек представляє собою пам'ять з детермінованою вибіркою, яка працює за принципом „останній прийшов - перший вийшов" (LIFO - Last In First Out). Стек виконує дві операції: push - вштовхування даних в стек, pop - виштовхування даних з стеку.

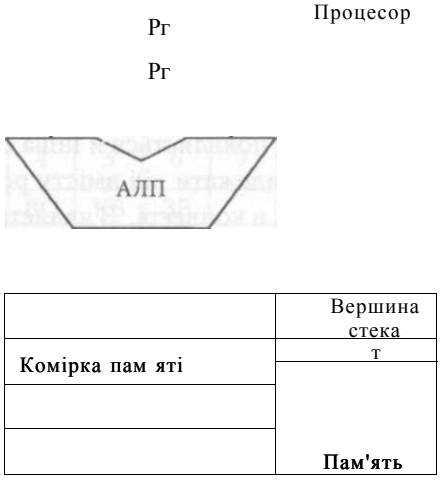


Рис. 3.15. Стекова архітектура

Інформація може бути занесеною в вершину стека з пам'яті або з регістра АЛП процесора. Перевага стекової архітектури - відсутність в команді адресної частини. З іншого боку, стекова архітектура не передбачає довільного доступу до комірок пам'яті, тому часто важко створити для неї ефективну програму. Крім того, стек не дозволяє підвищити продуктивність комп'ютера за рахунок розпаралелення, оскільки наявна лише одна вершина стека.

Стекова архітектура була реалізована в наступних комп'ютерах: B5500, B6500 фірми Burroughs, HP2116P, HP3 000/70 фірми Hewlett-Packard, JEM 1, JEM 2 фірми ajile Systems.

В акумуляторній архітектурі (рис. 3.16) операнд завжди знаходиться в акумуляторі - спеціальному регістрі процесора. В цей же регістр записується і результат операції. Оскільки адреса одного із операндів визначена, в команді достатньо вказати лише адресу другого операнда. Перевага даної архітектури - короткі команди. Вона була реалізована в комп'ютерах IBM 7090, DEC PDP-8 та інших.

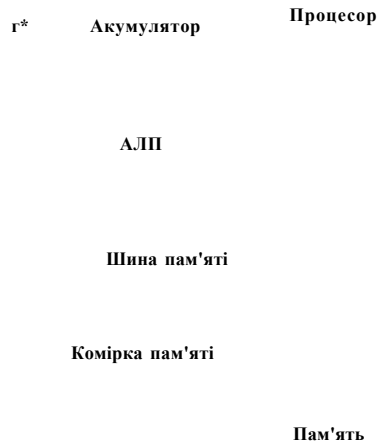


Рис. 3.16. Акумуляторна архітектура

Архітектура на основі регістрів загального призначення може мати такі різновидності як: архітектура типу пам'ять-пам'ять, регістр-пам'ять та регістр-регістр.

В архітектурі типу пам'ять-пам'ять (рис. 3.17) операнди поступають на вхідні регістри АЛП процесора прямо з пам'яті. Результат операції також записується прямо в пам'ять. Оскільки час звернення до пам'яті є більшим часу звернення до регістрів, ця архітектура характеризується низькою швидкодією. Прикладом таких комп'ютерів є сім'ї IBM System/370 та DEC VAX.

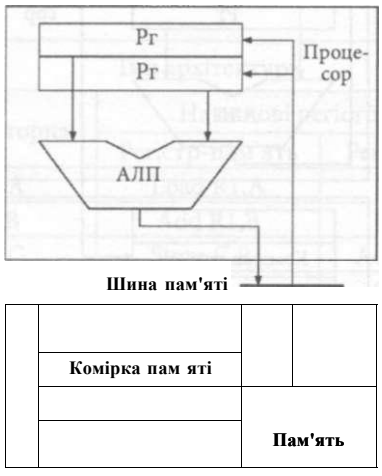


Рис. 3.17. Архітектура типу пам'ять-пам'ять

Архітектура типу регістр-пам'ять (рис. 3.18) передбачає вибірку та подачу в АЛП одного із операндів з пам'яті, а іншого - з регістра, тому характеризується вищою швидкодією ніж попередня. Тут в процесорі наявна регістрова пам'ять, причому регістри є програмно доступними.

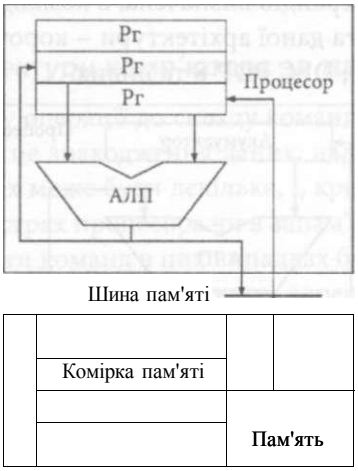


Рис. 3.18. Архітектура типу регістр-пам'ять

В архітектурі типу регістр-регістр (рис. 3.19) дані в АЛП поступають лише з регістрів процесора, результати виконання операцій також записуються в регістри, а обмін між цими регістрами і пам'яттю здійснюється паралельно з роботою АЛП. Ця архітектура характеризується високою швидкодією, оскільки операції виконуються в АЛП з їх читанням-записом до регістрів, які є значно швидшими пам'яті. Крім того, для цієї архітектури характерною є фіксована довжина команд та однакова кількість тактів для виконання всіх команд.

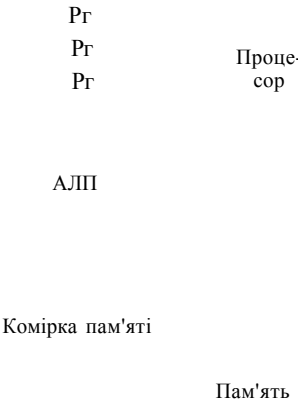


Рис. 3.19. Архітектура типу регістр-регістр

Будь-який із регістрів загального призначення може бути використаний в якості акумулятора, адресного регістра, індексного регістра, стекового регістра, а в деяких ма-

шинах навіть в якості програмного лічильника. Більшість сучасних комп'ютерів побудовані на основі описаної архітектури. Це, зокрема, комп'ютери-Pentium, SPARC, Power PC, A R M та інші.

Разом з тим, за регістрами можуть бути закріплені конкретні функції - один набір служить в якості індексних регістрів, інший призначений для зберігання арифметичних операндів і т. д. Таким чином організовані регістри в комп'ютерах сім'ї C D C 6000/7000.

3.3.2. Порівняльний аналіз форматів команд

Таким чином, як видно з попереднього пункту та прикладів із табл. 3.7, при роботі з основною пам'яттю в комп'ютерах використовуються нуль-, одно-,дво-, три- і чотири-адресні команди (рис. 3.20).

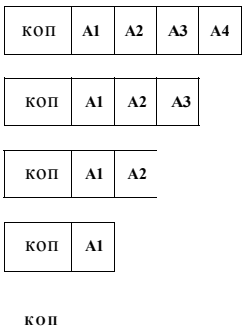


Рис. 3.20. Формати команд комп'ютерів

Розглянемо роботу комп'ютера з різними форматами команд на прикладі команди додавання двох чисел, яка найширше використовується. Нехай потрібно додати в АЛП число, яке знаходиться в пам'яті за адресою A, до числа, яке знаходиться в пам'яті за адресою B, і результат розмістити за адресою C:

$$C := [A] + [B].$$

В табл. 3.7 показана послідовність команд, які потрібно виконати при використанні трьох типів описаних архітектур для виконання операції додавання двох чисел.

Таблиця 3.7

Тип архітектури				
Стекова	Акумуляторна	На основі регістрів загального призначення		
		Регістр-пам'ять	Регістр-регістр	Пам'ять-пам'ять
Push A	Load A	Load R1,A	Load R1,A	Add A,B,C
Push B	Add B	Add R1,B	Load R2,B	
Add	Store C	Store C,R1	Add R3.R1.R2	
Pop C			Store C.R3	

Тут A, B і C - адреси комірок пам'яті, в яких знаходяться числа A, B, C; R1, R2 і R3 - адреси регістрів процесора.

При використанні триадресної команди та архітектури типу "пам'ять-пам'ять" вона символічно може бути записана наступним чином:

A D D A,B,C.

Якщо використовується двоадресна команда та архітектура типу "пам'ять-пам'ять", то для виконання поставленої задачі необхідно виконати дві команди:

ADD A,B, яка означає [A] := [A] + [B],
MOVE A,C, яка означає [C] := [A],

або дві команди:

MOVE B,C, яка означає [C]:= [B],
ADD A,C, яка означає [C] := [A] + [B].

Тут при виконанні двоадресної команди результат розміщується за адресою одного з операндів.

Одноадресна команда може бути реалізована з використанням акумуляторної архітектури. В цьому випадку операція C := [A] + [B] може бути виконана послідовністю із трьох команд:

LOAD A, яка означає АКК := [A],
ADD B, яка означає АКК := [АКК] + [B],
STORE C, яка означає C := [АКК].

При використанні архітектури на снові регістрів загального призначення, кожен з цих регістрів може бути використаний як акумулятор. Якщо таких регістрів r (зазвичай r рівне 8, 16, 32, 64), в полі команди необхідно добавити $\log_2 r$ бітів (3-6 бітів) для адресації регістрів, що беруть участь у виконанні операції. Це значно менше, ніж для адресації комірок пам'яті. Якщо Ri - і-й регістр регістрового файлу процесора, то команди з акумулятором можна замінити наступними:

LOAD A,Ri, яка означає Ргі := [A],
ADD B,Ri, яка означає Ргі := [Ргі] + [B],
STORE Ri,A, яка означає [A] := [Ргі].

Такого типу команди, коли одна частина адресного поля адресує основну пам'ять, а друга - регістровий файл, іноді називаються півтора адресними. Часто в комп'ютерах використовуються команди, в адресній частині яких вказуються тільки номери регістрів, наприклад:

ADD Ri,Rj, яка означає Rj := Ri + Rj

В цьому випадку команда є найкоротшою в порівнянні з іншими, та виконується в процесорі найшвидше.

Головні критерії вибору формату команд :

- чим коротша команда, тим менша ємність пам'яті потрібна для її зберігання і тим менша розрядність шини команд;

- чим більша адресність команди, тим менше відбувається звернень до пам'яті, тобто тим більша швидкодія комп'ютера.

Зазвичай шукається компромісне (оптимальне) рішення, і враховується, що розмір команди повинен бути узгодженим з розміром даних, оскільки вони зберігаються в одній пам'яті.

В перших комп'ютерах використовувалась архітектура на основі регістрів загального призначення типу "пам'ять-пам'ять". Але оскільки з розвитком елементної бази утворився великий розрив між швидкодією основної пам'яті та процесора, ця архітектура стала неефективною.

Після 1985 року більшість комп'ютерів будуються на основі архітектури типу регістр-пам'ять або регістр-регістр. Цьому є дві причини:

- перша - регістри процесора швидші пам'яті;
- друга - регістри простіше і ефективніше використовуються комп'ютером.

В цьому випадку найважливішим є те, що регістри можуть бути використані для зберігання змінних. Тоді зменшується об'єм обміну з пам'яттю і прискорюється виконання програми, оскільки регістри швидші пам'яті. Крім того, зменшується формат команди, оскільки адреса регістра коротша адреси пам'яті, тому що ємність регістрового файлу менша ємності ОП. Розробнику компілятора простіше працювати коли всі регістри регістрового файлу еквівалентні і загальнодоступні. В комп'ютерах перших поколінь регістри закріплювалися для конкретних цілей і тим самим число регістрів значно зменшувалось.

Яка кількість регістрів загального призначення є ефективною? Відповідь залежить від того, як вони використовуються компілятором. Більшість компіляторів використовують по декілька регістрів для роботи з формулами, зберігання параметрів та змінних.

Приклади комп'ютерів, в яких використовуються описані архітектури:

- регістр-регістр: SPARC, MIPS, Precision Architecture, Power PC, Alpha, Pentium IV, Athlon;
- регістр-пам'ять: Intel 80x86, Motorola 68000;
- пам'ять-пам'ять: VAX.

3.4. Способи адресації операндів

Як вже вказувалось, команда може включати від одного до чотирьох полів, які вказують адреси операндів. Варіанти інтерпретації бітів (розрядів) поля адреси з метою знаходження операнда називаються способами адресації. Коли команда вказує на операнд, він може знаходитись в самій команді, в основній або зовнішній пам'яті чи в регістровій пам'яті процесора. За роки існування комп'ютерів була створена своєрідна технологія адресації, яка передбачає реалізацію різних способів адресації, чому послужило ряд причин:

- забезпечення ефективного використання розрядної сітки команди;
- забезпечення ефективної апаратної підтримки роботи з масивами даних;
- забезпечення задання параметрів операндів;
- можливість генерації великих адрес на основі малих.

Існує велика кількість способів адресації. Розглянемо основні та найвживаніші способи адресації: безпосередню, пряму, непряму, відносну, базову, індексну, автоінкрементну

та автодекрементну, сторінкову, стекову. Практично у всіх існуючих комп'ютерах використовується один або декілька з цих способів адресації. Тому в команду потрібно вводити спеціальні ознаки з тим, щоб пристрій керування міг розпізнати використаний спосіб. Це можуть бути додаткові розряди в команді, або для різних типів команд закріплюватись різні способи адресації. Варіант формату команди з полями для розпізнавання типу адресації представлено на рис. 3.21, де ТАЇ - поле, яке вказує тип адресації, який використано в полі А1, а ТА2 - поле, яке вказує тип адресації, який використано в полі А2.

КОП	ТАЇ	А1	ТА 2	А2

Рис. 3.21. Формат команди з полями для розпізнавання типу адресації

Оскільки адреса підлягає обробці, розрізняють поточну Ат та виконавчу Ав адреси. Остання отримується як результат обробки (модифікації) поточної адреси Ат: Ав = Бі (Ат). Функція Бі задає спосіб модифікації поточної адреси залежно від способу адресації.

3.4.1. Безпосередня адресація

При безпосередній адресації операнд знаходиться безпосередньо в адресній частині команди (рис. 3.22), розрядність якої рівна розрядності операнда.

КОП Операнд

Рис. 3.22. Безпосередня адресація

Це найшвидший спосіб знаходження операнда, оскільки для його отримання потрібне лиш одне звернення до пам'яті. Він використовується для задания констант, наперед відомих чисел або початкових значень змінних. Недоліком є те, що розрядність операнда обмежується розрядністю поля адреси в команді, яке в більшості випадків є значно меншим розрядності даних.

3.4.2. Пряма адресація

При прямій (або абсолютній) адресації в адресному полі прямо вказується місце розміщення операнда, тобто виконавча адреса операнда. При цьому можливі два випадки: пряма адресація основної пам'яті ОП, та пряма адресація регістрів регістрового файлу процесора.

В першому випадку адреса А комірки основної пам'яті із адресної частини АЧ команди поступає на адресні входи основної пам'яті ОП (рис. 3.23 а) і у вказану комірку в режимі запису записується операнд із регістра даних РгД, а в режимі зчитування зчитується операнд в регістр даних РгД.

В другому випадку адреса К регістра регістрового файлу процесора із адресної частини АЧ команди поступає на адресні входи регістрового файлу процесора (рис. 3.23 Ь), і у вказаний регістр в режимі запису записується операнд із регістра даних РгД, а в режимі зчитування зчитується операнд в регістр даних РгД.

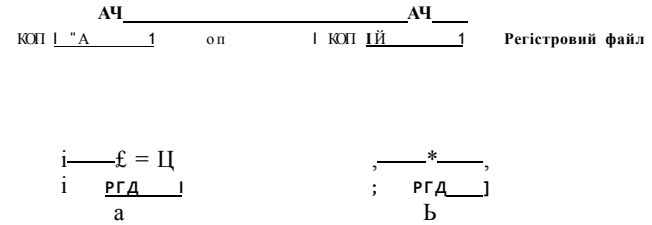


Рис. 3.23. Пряма адресація основної пам'яті а) та регістрового файлу процесора Ь)

Якщо основна пам'ять може зберігати М слів, то, використовуючи двійкове кодування, необхідно ш біт для представлення адрес всіх комірок пам'яті, де ш =]1од М[. Значення в дужках означає більше ціле. Якщо регістровий файл має N регістрів, то, використовуючи двійкове кодування, необхідно п біт для представлення адрес всіх регістрів, де п =]1од, КІ[. Оскільки число регістрів значно менше кількості комірок пам'яті, то і розрядність адреси для їх адресації буде значно меншою, а відповідно значно меншою буде і розрядність команди в цілому. Нехай для прикладу кількість виконуваних в комп'ютері команд рівна 256, тобто розрядність коду операції рівна 8 бітів, ємність основної пам'яті рівна 1ГБ, тобто розрядність адреси рівна ЗО бітів, а кількість регістрів регістрового файлу процесора рівна 64, тобто розрядність адреси рівна 6 бітів. На рис. 3.24 показано формати двоадресних команд при прямій адресації основної пам'яті та регістрового файлу процесора для наведеного прикладу.

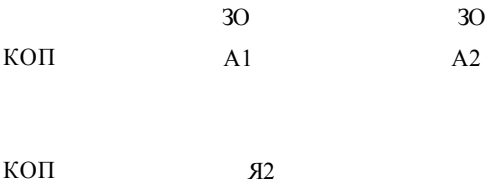


Рис. 3.24. Формати двоадресних команд при прямій адресації основної пам'яті та регістрового файлу процесора для наведеного прикладу

Як видно з рисунка, в першому випадку розрядність команди рівна 68 бітів, тоді як в другому випадку розрядність команди рівна 20 бітів, тобто менша в 3,4 рази.

Для того, щоб розпізнати який тип пам'яті адресується - основна пам'ять чи регістровий файл процесора, до команди вводиться спеціальне поле типу пам'яті ТП, як це показано на рис. 3.25.

Рис. 3.25. Команда з полем, яке вказує тип пам'яті

3.4.3. Непряма адресація

При непрямій адресації в адресному полі вказується місце розміщення адреси операнда, а виконавча адреса знаходиться наступним чином: А = [А'], де А' - адреса комірки пам'яті, в якій зберігається виконавча адреса. Адреса А із адресної частини АЧ команди поступає на адресні входи основної пам'яті ОП (рис. 3.26), з відповідної комірки осно-

п о

вної пам'яті ОП вибирається адреса операнда, по якій в відповідну комірку або регістр в режимі запису записується операнд із регістра даних РгД, а в режимі зчитування зчитується операнд в регістр даних РгД. Якщо пам'ять може зберігати М слів, то, використовуючи двійкове кодування, необхідно пт біт для представлення всіх адрес, де т =]к^ М[. Значення в дужках означає більше ціле. Для вибірки операнда необхідно здійснити два звернення до ОП.

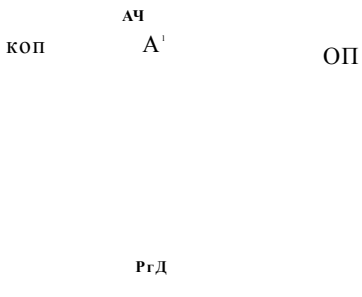


Рис. 3.26. Непряма адресація основної пам'яті, яка вимагає здійснення двох звернень

Для зберігання адрес операндів можна використати регістровий файл процесора (рис. 3.27). Якщо регістровий файл може зберігати N слів, то, використовуючи двійкове кодування, необхідно п біт для представлення непрямої адреси в адресній частині команди, де п =]к^2]М[, а розрядність регістрів буде рівною пт =]1од М[. Значення в дужках означає більше ціле. Для вибірки операнда необхідно здійснити одне звернення до регістрової пам'яті і одне звернення до основної пам'яті ОП. Такий підхід дозволяє при малій розрядності адресної частини команди п звертатися до пам'яті великої ємності маючи велику розрядність регістрів т.

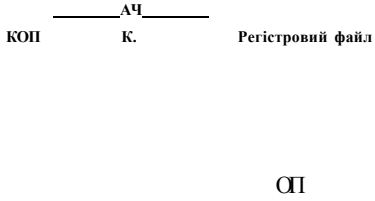


Рис. 3.27. Непряма адресація основної пам'яті з використанням регістрового файлу процесора

Потрібно відзначити, що використання регістрової пам'яті також дозволяє суттєво прискорити процес визначення ефективної адреси, оскільки час вибірки з неї значно менший порівняно з часом вибірки з основної пам'яті.

Можливе використання так званої багаторівневої або каскадної непрямої адресації, коли для знаходження ефективної адреси потрібно виконати кілька звернень до пам'яті. Кількість кроків звернення до пам'яті, необхідних при і-рівневій непрямій адресації, на-

зивається рангом гі. Розрізняють перший, другий і т. д. ранги. Пряма адресація - це адресація нульового рангу (гО).

Непряма адресація служить для зменшення довжини програми з великою кількістю змінних адрес.

3.4.4. Способи адресації операндів на основі операції зміщення

При використанні адресації на основі операції зміщення виконавча адреса формується шляхом додавання вмісту одного з адресних полів команди до вмісту одного або декількох регістрів процесора, або шляхом виконання операції конкатенації, тобто приєднання старшої частини адреси до молодшої. Нижче розглядаються способи адресації на основі операції зміщення.

3.4.4.1. Відносна адресація

При відносній адресації для отримання виконавчої адреси операнда вміст Б адресного поля команди додається до вмісту програмного лічильника ПЛ, як це показано на рис. 3.28. Тобто вміст адресного поля команди є зміщенням відносно адреси поточної команди. Даний тип адресації грунтується на тому, що при вибірці команд звернення відбувається до комірок пам'яті, розміщених поблизу одна від одної. Тим самим зменшується довжина адресної частини команди, оскільки довжина поля зміщення може бути досить малою. Більше того, при переміщенні програми в пам'яті значення зміщення не змінюється, оскільки взаємне розміщення в пам'яті команд програми при цьому не змінюється.

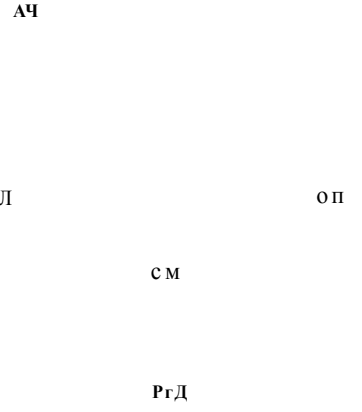


Рис. 3.28. Адресація основної пам'яті з використанням відносної адресації

Цей тип адресації іще називається відносною адресацією з перемінною базою, оскільки тут в якості регістра бази використаний програмний лічильник і модифікація базової адреси здійснюється автоматично.

3.4.4.2. *Базова адресація*

При використанні базової адресації (або базування) адресна частина команди вміщує два поля. В першому полі знаходиться адреса В регістра із регістрового файлу процесора, в якому зберігається база, до якого додається зміщення *В* із другого поля і тим

П 2

самим формується виконавча адреса операнда (рис. 3.29). Ця адреса поступає на адресні входи основної пам'яті ОП, у відповідну комірку якої в режимі запису записується операнд із регістра даних РгД, а в режимі зчитування зчитується операнд в регістр даних РгД.

Даний спосіб адресації дозволяє працювати з операндами із деякого сегмента пам'яті не змінюючи базу. Він ефективний при потребі обробки масиву даних. В якості бази тут виступає адреса першого елементу масиву, а всі інші його елементи вказуються шляхом додавання зміщення до адреси першого елементу масиву.

Якщо основна пам'ять може зберігати М слів, регістровий файл процесора може зберігати N слів, а сегмент має розмір Ь слів, то, використовуючи двійкове кодування, поле В буде займати п біт, де п =]lo§ 1M[, поле Б буде займати 1 біт, де 1 =]\o§ Ц, а розрядність регістрів буде рівною т =]log М[. Значення в дужках означає більше ціле.

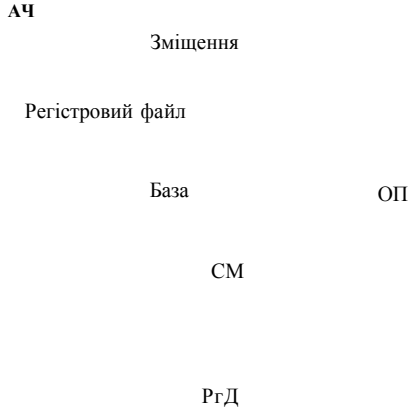


Рис. 3.29. Базова адресація ОП з використанням бази із регістрової пам'яті

Основна перевага відносної адресації - скорочення довжини команди за рахунок зменшення її адресної частини, а також спрощення розподілу пам'яті при написанні складних програм шляхом добавлення до кожного фрагменту програми відповідного значення базової адреси. Таким чином забезпечується переміщуваність фрагментів програми в полі пам'яті.

3.4.4.3. Індексна адресація

Індексна адресація використовується при виконанні циклів, коли потрібно збільшення або зменшення адреси на деяку величину. Цей спосіб адресації подібний до відносної адресації, при якій адреса може автоматично змінюватися в процесі виконання програми. Індексація є засобом для багатократного виконання одних і тих же відрізків програми над різними наборами (масивами) вхідних даних. Тим самим забезпечується мінімальна залежність довжини програми від кількості повторюваних відрізків програми. При цьому коди команд програми залишаються без змін. Для отримання виконавчої адреси адресна частина команди додається до вмісту спеціального регістра, в якому зберігається номер оброблюваного масиву чисел. Ці регістри називають індексними, а їх вміст - індексною величиною, або індексом (рис. 3.30).

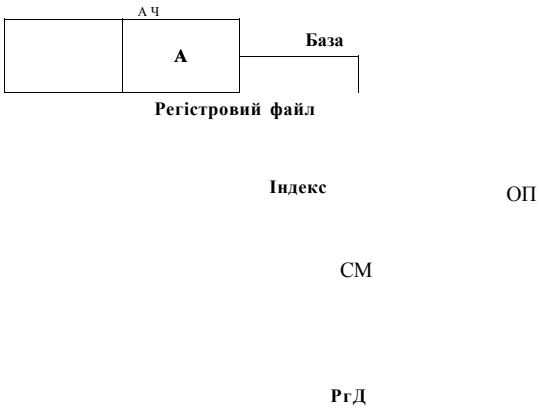


Рис. 3.30. Формування адреси при використанні індексної адресації

Вказівка про індексну адресацію вміщується в полі типу адресації. При наявності кількох індексних регістрів в цих розрядах команди вказується номер того індексного регістра, в якому зберігається значення індексу оброблюваного в даний час масиву інформації. Індекси можуть зберігатися як в спеціальних індексних регістрах, так і в регістровій пам'яті процесора. Вміст індексних регістрів змінюється після закінчення деякого циклу обробки. При цьому до попереднього значення індексу добавляється приріст, значення якого залежить від розміщення операндів в пам'яті.

Різновидністю індексної адресації є автоіндексація, при якій значення індексу є відомим наперед. Найчастіше операнди розміщуються в пам'яті послідовно і тому це значення рівне + 1 (так звана автоінкрементна адресація) або -1 (так звана автодекрементна адресація). Порядок формування адреси при використанні автоінкрементної та автодекрементної адресації показано на рис. 3.31, де для забезпечення переміщення по комірках пам'яті використовується лічильник.

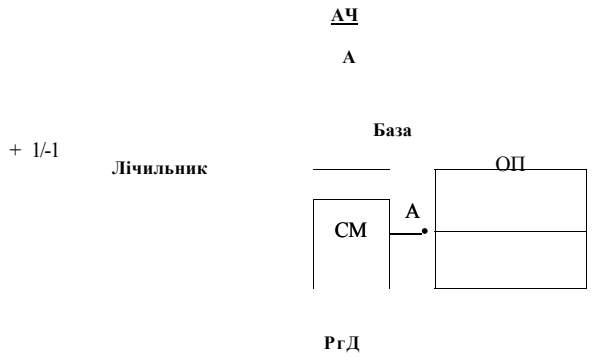


Рис. 3.31. Формування адреси при використанні автоінкрементної та автодекрементної адресації

Операції індексної арифметики виконуються в спеціальному індексному арифметичному пристрої, або в арифметико-логічному пристрої процесора. Команди індексної арифметики входять до складу команд керування. Вони забезпечують зміну значення

індексу шляхом добавлення до нього приросту, перевірку закінчення індексного циклу та засилання початкових значень індексу. Значення індексів, приростів та інформаційні біти циклів формують керуюче слово, яке розміщується в одній або декількох командах керування. Послідовність таких управляючих слів забезпечує роботу з масивами даних.

3.4.5. *Сторінкова адресація*

При сторінковій адресації адресний простір пам'яті розбивається на сторінки рівного розміру. Сторінка має початкову адресу, яка використовується в якості бази та зберігається в спеціальному регістрі, який називається регістром адреси сторінки. В адресній частині команди вказується зміщення всередині сторінки, яке є молодшою частиною виконавчої адреси. Тобто виконавча адреса формується шляхом приєднання (конкатенації) зміщення з адресної частини команди до початкової адреси. База може зберігатися в одному з регістрів загального призначення, як це показано на рис. 3.32. Число з цього регістра береться в якості старших розрядів срА адреси, а зміщення з адресної частини команди - в якості молодших розрядів мрА адреси.

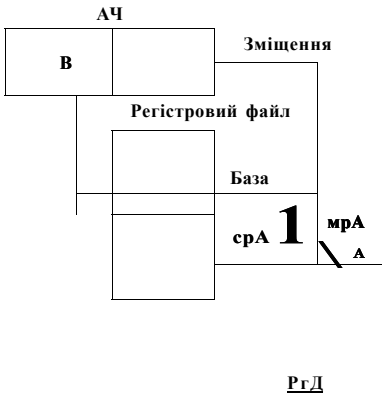


Рис. 3.32. Формування адреси при використанні сторінкової адресації

3.4.6. Неявна адресація

Існують способи адресації, при яких код адреси операнда в явному вигляді в команді відсутній. Так, використання одноадресного формату команди привело до того, що в команді адресується лише один з операндів, інші ж при цьому визначаються самим кодом операції. Наприклад, при виконанні арифметичних операцій адреса одного з операндів вказується в адресній частині команди, а інший операнд знаходиться в акумуляторі. Адреса останнього в команді не вказується, а є відомою наперед.

Неявна адресація дозволяє скоротити довжину команди, тому знайшла широке використання.

3.4.7. Стекова адресація

Зменшення довжини команди скорочує час виконання і економить пам'ять. Межею зменшення є безадресні команди, які можливі при використанні стекової адресації. Стекова адресація використовується в безадресних командах при роботі з масивами даних.

Широко використовується в мікропроцесорах і мікрокомп'ютерах. Принципи організації стекової адресації ілюструє рис. 3.33.

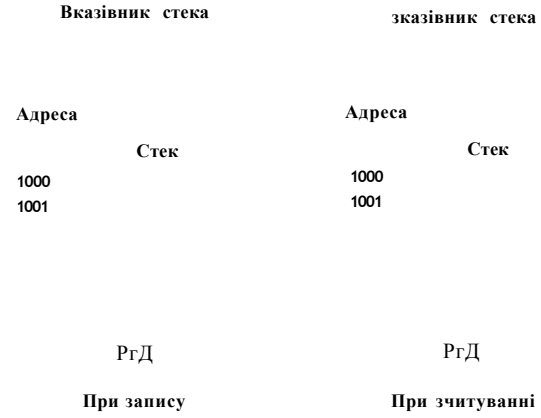


Рис. 3.33. Стекова адресація

Стек - це, набір комірок пам'яті або регістрів, в яких дані масиву розміщуються в заданому порядку, а саме відповідно до правила організації пам'яті з послідовним доступом типу FILO. Місце розміщення першого даного масиву називається дном стека, а останнього - вершиною стека. Для запису та читання даних передбачено дві операції: push (вштовхування даних в стек) та рор (виштовхування даних зі стеку). Операції зовнішнього запису та читання можливі тільки з вершиною стека. На її номер вказує вміст вказівника стека. При запису всі дані в стеку зміщуються на одну позицію вниз, а при зчитуванні зміщуються на одну позицію вверх. На рисунку 3.34 показано функціонування стека при запису та зчитуванні даних при виконанні двомісної операції множення числа 50 на число 10 в арифметико-логічному пристрої із записом результату в стек.

Рис. 3.34. Виконання двомісної операції з використанням стека

3.4.8. Використання стекової адресації

Зазвичай в математиці прийнято записувати знак операції між операндами, наприклад а+Ь, с:сі і т. д. Такий запис називають інфіксним. При використанні такого запису для обчислення складного виразу необхідно задавати пріоритети операцій. Наприклад, в алгебраїчному виразі

а + Ьс/сі - {

пріоритети можуть бути наступні: х, /, +, -.

Інший підхід, який забезпечує правильне виконання інфіксного запису - використання дужок, причому, з тим, що обчислення проводиться від внутрішніх дужок до зовнішніх. В дужковій формі наведений вираз прийме вигляд:

((a + ((Ьс)/а)) - **0.**

Замість аналізу пріоритету знаків операцій тут необхідно визначити дужки з найбільшою глибиною вкладення.

Для реалізації наведеного виразу в комп'ютері можна використати всі раніше описані способи адресації.

Польський математик Ян Лукашевич показав, що якщо знаки арифметичних операцій записувати перед операндами (префіксна форма) чи після операндів (постфіксна форма, або обернений польський запис), то для визначення порядку виконання операцій дужки стають непотрібними.

Так, для наведеного вище виразу префіксна форма має вигляд:

- + а / х Ьс **ci { ,**

а постфіксна форма має вигляд:

аЬсх *o\ / + {-*.

Ця форма - обернений польський запис.

Обернений польський запис прекрасно підходить для проведення обчислень на комп'ютері зі стеком.

Якщо вираз складається із N символів, то алгоритм його обчислення на стеку можна представити у вигляді блок-схеми, показаної на рис. 3.35.

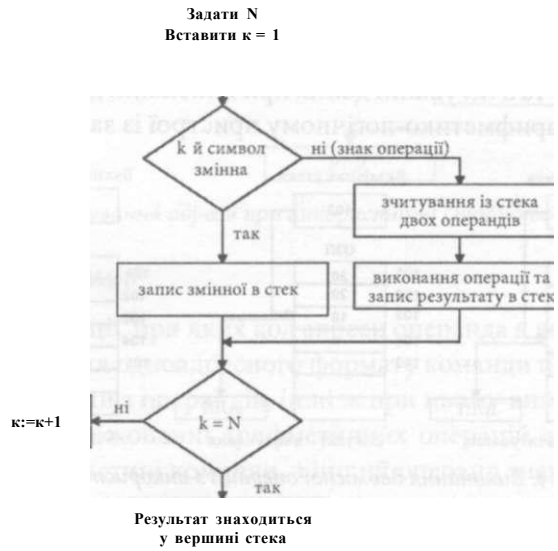


Рис. 3.35. Блок-схема обчислення на стеку виразу, представленого в постфіксній формі

Розглянемо потактовий стан комірок стеку при реалізації раніше розглянутого виразу, записаного в постфіксній формі (рис. 3.36).

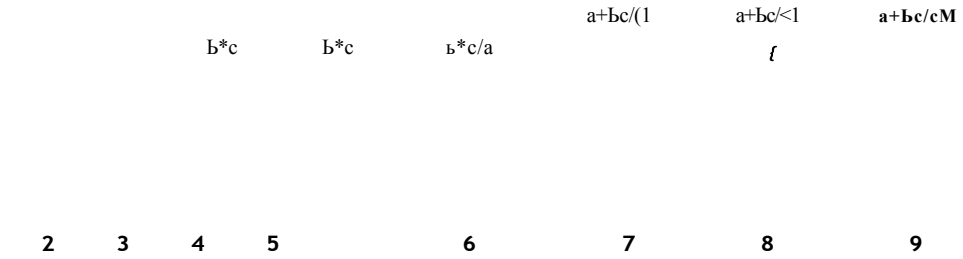


Рис. 3.36. Використання стека при розрахунку по формулі в оберненому польському запису

Для реалізації стека можуть бути використані регістри. В цьому випадку схема обчислень з використанням стека має вигляд, показаний на рис. 3.37.

Рис. 3.37. Схема обчислень з використанням регістрового стека

Алгоритм функціонування такий же, як вище описано функціонування на базі основної пам'яті ОП. Операція виконується над вмістом Ргі і результат розміщується в Ргі, а вміст нижніх регістрів зміщується на один крок вверх.

3.4.9. Вибір способів адресації операндів

В табл. 3.8 наведено перелік та короткий опис різних способів адресації, зроблений на основі вище наведеного розгляду. Кожний спосіб має свої переваги та сферу доцільного застосування.

Таблиця 3.8

Основні способи адресації	
Спосіб адресації	Місце розміщення операнда
Безпосередня	Значення операнда знаходиться в полі адресної частини команди
Пряма	Адреса операнда знаходиться в полі адресної частини команди
Непряма	Адресна частина команди вказує адресу розміщення операнда
Базова	Виконавча адреса формується шляхом додавання до вмісту регістра бази зміщення із поля адресної частини команди
Відносна	Виконавча адреса формується шляхом додавання вмісту адресного поля команди до вмісту програмного лічильника
Індексна	Виконавча адреса формується шляхом додавання вмісту адресного поля команди до вмісту спеціального регістра
Неявна	Адреса операнда в явному вигляді в команді відсутня
Сторінкова	Виконавча адреса формується шляхом конкатенації зміщення з адресної поля команди до початкової адреси
Стекова	Операнд розміщений у вершині стека

Частота використання різних способів адресації в комп'ютерах залежить від багатьох факторів: типу архітектури, вимог до параметрів комп'ютера, характеру вирішуваних задач. За час існування обчислювальної техніки була створена велика кількість комп'ютерів, у яких застосовано різноманітні способи адресації та їх поєднання. Тому дати однозначну відповідь стосовно вибору найкращого набору способів адресації практично неможливо. Але такому вибору може допомогти аналіз частоти застосування різних способів адресації при вирішенні на комп'ютерах складних тестових задач, як наприклад, програм TeX, gcc та spice, як це показано на рис. 3.38, які тестувалися на комп'ютері V A X фірми DEC.

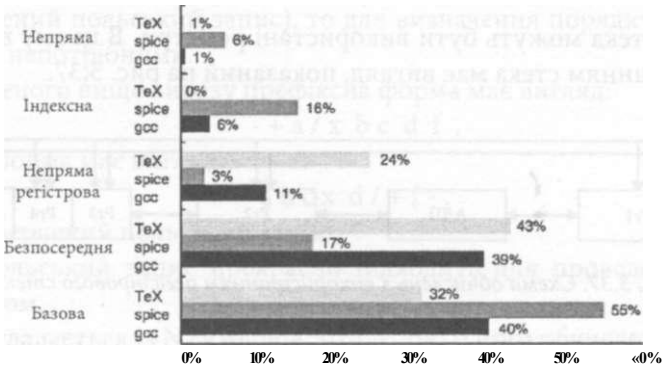


Рис. 3.38. Частота використання способів адресації при виконанні програм TeX, spice та gcc на комп'ютері VAX фірми DEC

Видно, що з використовуваних способів адресації найчастіше застосовується безпосередня, базова та непряма регістрова (з використанням регістрового файла процесора) адресація. Виходячи з подібного типу аналізу, в новітніх комп'ютерах проведено суттєве скорочення використовуваних способів адресації, що дозволило спростити комп'ютери та за рахунок цього підвищити їх продуктивність.

3.5. Приклади форматів команд

При розгляді систем команд будемо використовувати розповсюджені скорочені позначення типів команд: RR - регістр-регістр, RI - регістр-безпосередній операнд, RS - регістр-пам'ять, RX - регістр-індексована пам'ять, SI - пам'ять-безпосередній операнд, SS - пам'ять-пам'ять, а також позначення: КОП - код операції, R-регістр, S - пам'ять, X - індекс, B - базова адреса, D - зміщення, A - адреса пам'яті, L - довжина, C -номер символу в складному слові.

Формати команд комп'ютерів різних типів детально розглянуті в літературі та в технічній документації на ці комп'ютери. На рис. 3.39 показано три групи узагальнених форматів команд, які використовувалися та використовуються зараз в комп'ютерах: a - змінний, b - фіксований та c - гібридний формати.

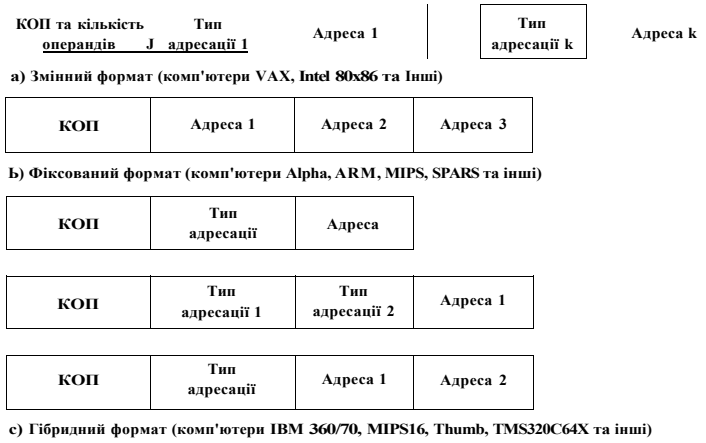


Рис. 3.39. Використовувані в комп'ютерах формати команд

Для наочності використання різних типів описаних вище форматів команд, аналізу *їх* переваг та недоліків розглянемо кілька характерних прикладів форматів команд, які використовувалися раніше в різних поколіннях комп'ютерів та які використовуються в сучасних комп'ютерах.

3.5.1. Формати команд комп'ютерної системи IBM 370

В системі IBM 370 використовувалися три варіанти довжини команди: двобайтова, чотирибайтова, шестибайтова. Також використовувалися два варіанти довжини коду операції: однобайтовий та двобайтовий. В сумі це складає десять різних форматів команд. На рис. 3.40 наведено формати команд комп'ютерної системи IBM 370.

Тип формату команди вказується першими двома розрядами коду операції КОП: 00 - RR; 01 - RX; 10 - RRE, RS, RX, S, SI; 11 - SS, SSE. Коротко опишемо кожен тип команди.

Команда регістр-регістр RR. Цей формат команди є двобайтовим. В адресній частині звернення відбувається до регістрів. Оскільки багато операцій виконується з використанням регістрів, такий формат при своїй компактності є досить ефективним.

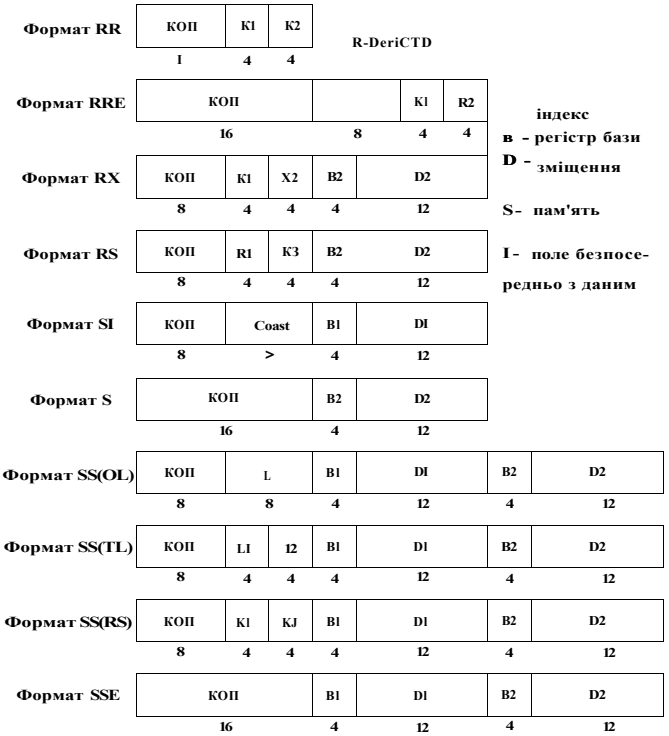


Рис. 3.40. Формати команд комп'ютерної системи IBM 370

Розширена команда регістр-регістр RRE (E - Extended). Цей формат використовується для декількох спеціальних привілейованих команд операційної системи. Розширений код операції дозволяє виконання додаткових операцій. Поле після коду операції не використовується.

Команда регістр-індексована пам'ять RX. За цією командою перший операнд знаходиться в регістрі, а другий операнд обчислюється шляхом додавання 12-розрядного зміщення D2 до вмісту регістра бази B2 та індексного регістра X2. Обидва регістри належать до регістрів загального призначення.

Команда регістр-пам'ять RS. Ця команда має триадресний формат. Тут також є три звернення до регістрів, але вони вказують на три різних операнди. Третій регістр використовується як регістр бази, до якого додається зміщення.

Команда пам'ять-безпосередній операнд SI (I - Immediate). Тут адреса першого операнда вираховується шляхом додавання зміщення до бази, а другий операнд знаходиться безпосередньо в 8-розрядному полі адреси.

Команда пам'ять S. Це привілейована команда, яка використовується для введення-виведення або системою контролю функцій. Тут використовується розширений 16-розрядний код операції. Адреса другого операнда вираховується шляхом додавання зміщення до бази. Адреса ж першого операнда, якщо він є, вказується кодом операції.

Команда пам'ять-пам'ять SS. Ця команда займає 6 байт і вказує на два операнди, розміщені в пам'яті. Наступні після коду операції 8 біт можуть бути використані трьома варіантами: в форматі одиночної довжини (OL) поле L вказує кількість байт, які будуть оброблені; в форматі подвійної довжини (EL) поле L вказує довжину двох операндів у байтах (цей формат використовується для команд десяткової арифметики; третій варіант (RS) використовується в декількох привілейованих командах, в яких другий байт вказує на два регістри загального призначення. Ці регістри вміщують покажчики або іншу керуючу інформацію.

Розширена команда пам'ять-пам'ять SSE. Цей формат також використовується в декількох привілейованих командах з розширеним кодом операції. Адреса першого та другого операндів вираховується шляхом додавання зміщення до бази.

3.5.2. Формати команд комп'ютера Cyber-70

Розглянемо іще одну систему команд, яка використовувалась в комп'ютері Cyber-70 (рис. 3.41). її відмінністю від інших є нестандартна довжина розрядної сітки, кратна трьом.

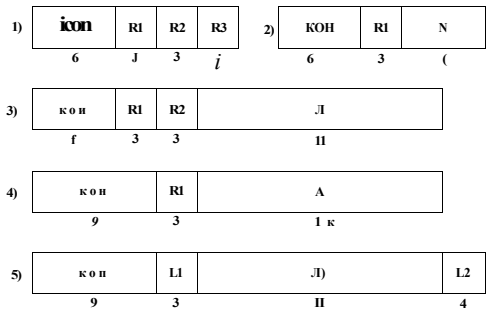


Рис. 3.41. Формат команди комп'ютера Cyber-70

Тут А - адреса пам'яті, К - адреса регістра, Ь - довжина операнда, С - номер символу в складному слові, N - число. Також використовуються дві довжини коду операції: шести- та дев'ятирозрядний. В сумі використовується п'ять різних форматів команди. Перший дозволяє одночасно адресувати три регістри, другий адресує один регістр та вміщує число, що підлягає обробці, третій адресує два регістри та пам'ять і є в два рази довший, так само, як і четвертий формат, який адресує один регістр і пам'ять, але має довший код операції. Іще в два рази довшим є п'ятий формат команди, який вказує дві адреси пам'яті та параметри відповідних чисел - довжину і номер в складному слові.

3.5.3. *Формати команд сучасного комп'ютера*

Формати команд сучасного комп'ютера на прикладі комп'ютера ПЬХ, який є узагальненням цілого спектра сучасних комп'ютерів, подано на рис. 3.42.

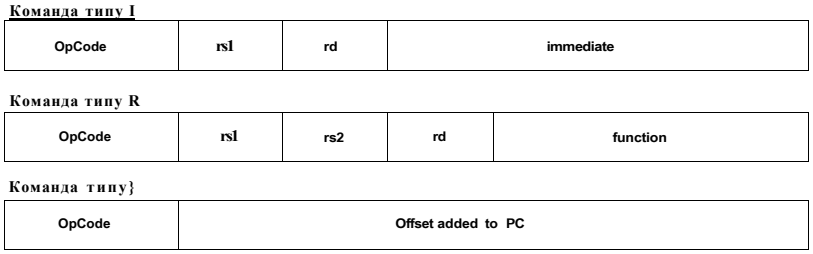


Рис. 3.42. Формати команд комп'ютера DLX

Дамо коротке пояснення щодо наведених на рисунку форматів команд.

• Команда типу I опрацьовує безпосередній операнд (Immediate).

• Команда типу R отримує пару операндів із джерельних регістрів (Registers) регістрового файла процесора та повертає результат знов таки до регістра призначення цього файла.

• Команда типу} є командою безумовного переходу (Jump).

• Opcode є полем коду операції КОП, довжина якого становить 6 бітів.

• rs1, rs2 є полями з довжиною 5 бітів, що визначають номери регістрів-джерел операндів (register of source) та програмно вибираються серед регістрів R0...R31 регістрового файла.

• rd є п'ятибітовим полем номера регістра призначення, приймача результату дії (register of destination). Регістр призначення також вибирають із множини R0...R31 регістрового файла.

• Immediate - це 16-бітове поле, що містить безпосередній операнд. При цьому лівий розряд immediate розглядають як знаковий. При використанні безпосередній операнд розширюють вліво за правилами доповняльного коду до 32-х бітів.

• Function - це поле, що визначає функцію, яка розширює на 2^a - 1 = 2047 комбінацій обмежену кількість дозволених кодів операції.

• Offset added to PC - це 26-бітова константа, яку додають до вмісту регістра наступної адреси при виконанні команди безумовного переходу.

Особливості форматів команд комп'ютера DLX:

• Довжина усіх форматів - 32 біти.

• Реалізовано тип архітектури регістр-регістр.

• Реалізовано фіксовану систему поділу форматів на поля.

• Усі команди з погляду їхньої обробки поділено на три групи: операції АЛП, операції зчитування/запису, операції керування виконанням програми.

Формати команд АЛП є триадресними, а саме, OP RX,RY,RZ. Вони є майже збіжними з форматами команд мікропроцесора M88X00 фірми Motorola. Останній, разом із мікропроцесорами IBM801 та AMD29000 у середині 80-х років склав історично першу трійку серійних комп'ютерів з архітектурою RISC.

3.6. Вплив технології компілювання на систему команд комп'ютера

В перших поколіннях комп'ютерів програми писались на асемблерній мові. Тому архітектура системи команд часто будувалась виходячи з потреби спрощення програмування на асемблері. Сьогодні переважна частина програм пишеться на мовах високого рівня. Для їх перекладу в машинні команди використовуються спеціальні програми, які називаються компіляторами. Перші компілятори створювались для вже існуючих комп'ютерів з конкретною системою команд. Оскільки від компілятора значною мірою залежить продуктивність комп'ютера, розуміння технології компілювання сьогодні є вкрай необхідним для проектування ефективної системи команд. Тому в подальшому розглянемо питання вибору складу системи команд комп'ютера з точки зору ефективного компілювання та виконання на ньому програм мовами високого рівня.

Структура сучасного компілятора показана на рис. 3.43. Перша процедура передбачає перехід з мови програмування високого рівня до деякої простої проміжної мови і є залежною від мови програмування та незалежною від апаратних засобів комп'ютера.

Під час виконання другої процедури здійснюється оптимізація коду представленого проміжною мовою, зокрема розкриваються звернення до підпрограм та циклів. Ця процедура також залежить від типу проміжної мови та не залежить від апаратних засобів комп'ютера.

На третій процедурі виконується глобальна оптимізація з врахуванням базових архітектурних принципів побудови комп'ютера. Тут здійснюється прив'язка до конкретних типів комп'ютерних пристроїв. На останній процедурі здійснюється детальний вибір команд та машинно-залежна оптимізація.

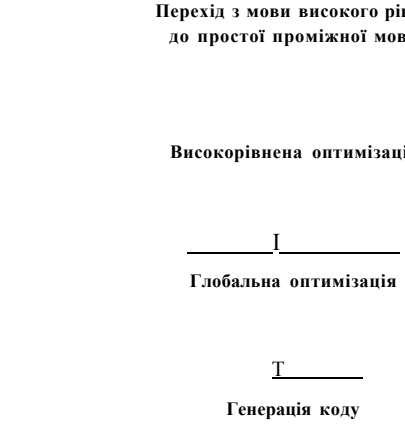


Рис. 3.43. Структура компілятора

3.7. Архітектура системи команд комп'ютера

3.7.1. Класифікація архітектури комп'ютера за складом системи команд

Як вже було відмічено, розгляд архітектури комп'ютера на рівні системи команд встановлює межу між апаратурою і програмним забезпеченням і дозволяє побачити комп'ютер на рівні, який видимий програмісту, що працює на мові асемблера, або розробнику компіляторів.

За складом системи команд комп'ютери можуть бути поділені на наступні типи:

• комп'ютери із складною (комплексною) системою команд (КССК, Complex Instruction Set Computers - CISC);

• комп'ютери з простою (спрощеною) системою команд (КПСК, Reduced Instruction Set Computers - RISC);

• комп'ютери з доповненою системою команд (КДСК, Supplemented Instruction Set Computers - SISC). В таких комп'ютерах складна або проста система команд доповнюється командами, орієнтованими на конкретну область використання. До таких комп'ютерів, зокрема, відносяться програмовані процесори обробки сигналів (Programmable Digital Signal Processors);

• комп'ютери з орієнтованою (спеціалізованою) системою команд (КОСК, Application Specific Instruction Set Computers - ASISC).

Далі ці архітектури будуть розглянуті детально.

3.7.2. Комп'ютери із складною та з простою системами команд

Двома основними архітектурами, які розрізняються за складом системи команд, а відповідно і за іншими елементами, що буде показано в подальшому, та які найширше використовуються комп'ютерною промисловістю на сучасному етапі, є архітектури КССК та КПСК (CISC і RISC). Основоположником архітектури КССК можна вважати компанію IBM з її базовою архітектурою IBM/360, ядро якої використовується з 1964 року і дійшло до наших днів, наприклад, в таких сучасних мейнфреймах, як IBM ES/9000.

Лідером в розробці мікропроцесорів на основі архітектури КССК є компанія Intel зі своїми серіями x86 і Pentium. Ця архітектура де-факто є стандартом для ринку мікрокомп'ютерів. Для архітектури КССК характерне порівняно невелике число регістрів загального призначення; велика кількість машинних команд, деякі з яких семантично навантажені аналогічно операторам мов програмування високого рівня і виконуються за багато тактів; велика кількість методів адресації; велика кількість форматів команд різної розрядності; переважання двоадресного формату команд; наявність команд типу регістр-пам'ять.

Основою архітектури сучасних робочих станцій і серверів є архітектура комп'ютера з простою системою команд КПСК. Зачатки цієї архітектури йдуть своїм корінням до комп'ютерів CDC 6600, розробники яких (Торнтон, Крей та ін.) усвідомили важливість спрощення набору команд для побудови швидких обчислювальних машин. Цю традицію спрощення архітектури С. Крей з успіхом застосував при створенні широко відомої серії суперкомп'ютерів компанії Cray Research. Проте остаточно поняття комп'ютера з простою системою команд в сучасному його розумінні сформувалося на базі трьох дослідницьких проектів комп'ютерів: процесора 801 компанії IBM, процесора RISC університету Берклі та процесора MIPS Стенфордського університету.

Розробка експериментального проекту компанії IBM почалася ще в кінці 70-х років, але його результати ніколи не публікувалися і комп'ютер на його основі в промислових масштабах не виготовлявся. У 1980 році Д. Паттерсон із колегами з університету Берклі почали свій проект і виготовили дві машини, які одержали назви RISC-I і RISC-II. Головними ідеями цих машин було відділення повільної пам'яті від високошвидкісних регістрів і використання регістрових вікон. У 1981 році Дж. Хеннессі із своїми колегами опублікував опис стенфордської машини MIPS, основним аспектом розробки якої була ефективна реалізація конвеєрної обробки за допомогою ретельного планування компілятором його завантаження.

Ці три машини мали багато спільного. Всі вони дотримувалися архітектури, що відокремлює команди обробки від команд роботи з пам'яттю, і робили акцент на ефективну конвеєрну обробку. Система команд розроблялася так, щоб виконання будь-якої команди займало невелику кількість машинних тактів (переважно один машинний такт). Сама логіка виконання команд з метою підвищення продуктивності орієнтувалася на апаратну, а не на мікропрограмну реалізацію. Щоб спростити логіку декодування команд, використовувалися команди фіксованої довжини і фіксованого формату.

Серед інших особливостей архітектури КПСК слід зазначити наявність досить великого регістрового фажта. тжсдал гр^яітоаур^ ¥5\Сл\ реалізуються *У1* або більша кількість регістрів у порівнянні з 8-16 регістрами в архітектурі КССК), що дозволяє більшому об'єму даних зберігатися в регістрах на кристалі процесора більший час і спрощує роботу компілятора при розподілі регістрів під змінні. Для обробки, як правило, використовуються триадресні команди, що, крім спрощення дешифрування, дає можливість зберігати більшу кількість змінних в регістрах без їх подальшого перезавантаження.

До часу завершення університетських проектів (1983-1984 рр.) відбувся також прорив у технології виготовлення надвеликих інтегральних схем. Простота архітектури та її ефективність, підтверджена цими проектами, викликали великий інтерес в комп'ютерній індустрії, і з 1986 року почалася активна промислова реалізація архітектури КПСК. До теперішнього часу ця архітектура міцно займає лідируючі позиції на світовому комп'ютерному ринку робочих станцій і серверів.

Розвиток архітектури КПСК значною мірою визначався прогресом у області створення оптимізуючих компіляторів. Саме сучасна технологія компіляції дозволяє ефективно використовувати переваги більшого регістрового файла, конвеєрної організації та більшої швидкості виконання команд. Сучасні компілятори використовують також переваги іншої технології оптимізації для підвищення продуктивності, зазвичай вживаної в комп'ютерах КПСК: реалізацію затриманих переходів і суперскалярної обробки, що дозволяє в один і той же момент часу видавати на виконання декілька команд.

Слід зазначити, що в останніх розробках компанії Intel (маються на увазі Pentium P54C і процесор наступного покоління P6), а також її послідовників-конкурентів (AMD R5, Cyrix M1, NexGen Nx586 та ін.) широко використовуються ідеї, реалізовані в архітектурах КССК, так що багато відмінностей між КССК і КПСК стираються.

3.7.3. Особливості архітектури комп'ютера з простою системою команд

Відомі ще з початку 80-х років принципи реалізації КПСК є наступними:

• Довільна комп'ютерна команда, незалежно від її типу, має виконуватися за один такт (чи однотактовий цикл).

• Система команд має містити мінімальну кількість спрощених команд, що статистично переважають у програмах

• Команди обробки даних реалізуються лише у формі «регістр регістр». Обміни з пам'яттю даних (гарвардська архітектура), з метою модифікації змінних у пам'яті виконуються лише за допомогою команд читання/запису (архітектура ІоааУзгоге).

• Програми, що модифікують власні коди (раніше це було розповсюджено у різних комп'ютерах, зокрема РОР-11/УАХ-11) є забороненими. Виходить, що згідно з новою концепцією команди обробки не можуть адресувати нічого, за винятком регістрів процесора

• Дешифрування команд із спрощеними форматами має виконуватися лише апаратно, аби збільшити швидкодію

• У системі команд відносно небагато операцій та режимів адресування операндів (способів адресації).

• Високий рівень конвеєризації виконання команд

• Велика кількість регістрів

• Застосовується багато рівнів ієрархії пам'яті

• Склад системи команд має задовольняти вимоги «зручної» компіляції операторів мов високого рівня

Виходячи з наведених вище принципів, можна прийти до висновку, що проектування КПСК вимагає вирішення наступних завдань

• Аналіз області використання з метою визначення найпоширеніших операцій та формування на основі цього списку команд (наприклад, обробки сигналів або створення документа).

• Оптимізація структури процесора, що проектується, з метою забезпечення найшвидшого виконання обраних команд

• Додавання до отриманого списку інших команд, якщо вони не ускладнюють процесора

• Застосування цих же принципів у розробленні інших пристроїв комп ютерної системи.

• Перенесення більшої частини дій з апаратури на програмну частину (компілятор).

Потрібно відзначити, що з часом тлумачення окремих принципів змінилося. Наприклад, вимогу виконання команди за один такт почали розуміти в той спосіб, що результати усіх операцій мають формуватися з темпом «одне слово за такт». Іншими словами, усі процесори обов язково містять конвеєризовані арифметичні пристрої. Сучасні технології елементної бази дозволили реалізувати замість первісних десятків команд більше сотні (до 150-200). Проте основний принцип архітектури КПСК виконувати операції тільки у межах регістрової структури процесора, аби виключити звертання до пам'яті даних, залишився незмінним

3.7.4. Архітектура комп'ютера з доповненою системою команд

Основною вимогою до КДСК є забезпечення високої продуктивності при реалізації алгоритмів з великим обсягом обчислень в реальному масштабі часу. В КДСК ця вимога задовольняється завдяки використанню таких основних принципів:

розділенню шин даних і команд із забезпеченням інформаційного обміну між ними, тобто використанню гарвардської архітектури;

- широкому використанню конвеєрного принципу обробки даних;

- використанню, крім традиційного АЛП, спеціалізованого операційного пристрою, який структурно орієнтований на виконання найуживаніших операцій

- використанню широкорозрядних блоків оперативної та постійної пам'яті великої ємності з можливістю її секціювання із забезпеченням незалежного доступу до секцій;

- використанню взамін адресного регістра спеціального процесора формування адрес та його структурній орієнтації на роботу з масивами даних та на виконувані алгоритми адресації пам'яті

- введенню в пристрої вибірки команд стеків індексних регістрів, регістрів загального призначення, регістрів попередньої вибірки та інших апаратних засобів для маніпуляцій з командами і даними;

- введенню в систему команд спеціальних команд для виконання найуживаніших алгоритмів;

- використанню швидкодіючих послідовних і паралельних інтерфейсів, які забезпечують створення багатопроцесорних систем

- короткому командному циклу

Дані принципи, покладені в основу проектування КДСК, забезпечують їм високі технічні параметри, завдяки яким КДСК обробляють великі масиви інформації за складними алгоритмами і виконують більшість операцій за один командний цикл

Типова структура комп'ютера з доповненою системою команд наведена на рис. 3.44. Видно, що тут використовується гарвардська архітектура, яка передбачає розподіл пам яті на пам ять даних і пам ять програм, що дає змогу суміщати в часі вибірку і виконання команд. Такий розподіл шин дає змогу створити конвеєр виконання команд і підвищити продуктивність КДСК. В багатьох комп'ютерах з доповненою системою команд, зокрема сім'ї ТМБ320, використовується модифікована гарвардська архітектура, яка допускає обмін інформацією між пам'яттю даних і пам яттю команд створенням зв'язку між шиною даних і шиною команд (міст зв'язку шин на рис. 3.44).

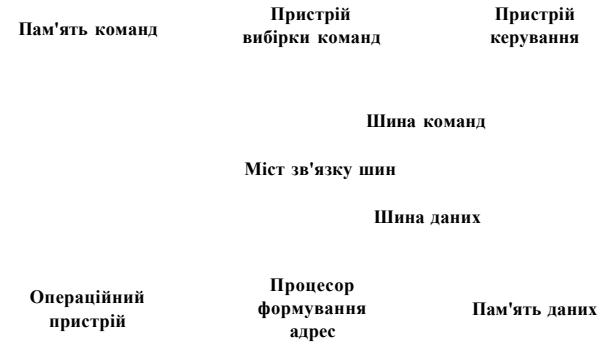


Рис. 3.51. Типова структура комп'ютера з доповненою системою команд

3.7.5. Комп'ютери зі спеціалізованою системою команд

Сьогодні комп ютерні засоби широко впроваджуються в нових сферах, таких як системи мобільного та персонального зв'язку, засоби мультимедіа, які формують інтенсивно зростаючий сектор електронної індустрії. На базі розповсюджених засобів зв'язку, таких як супутниковий зв язок, коміркове радіо, швидкісні оптичні мережі, з'являються

нові засоби для домашнього і ділового використання. Успішне використання цих засобів головним чином пов'язане з можливістю реалізації складних функцій цифрової обробки сигналів у вбудованих комп'ютерах. При розробці цих комп'ютерів ставляться жорсткі вимоги до їх продуктивності та споживаної потужності. Крім того, існуючі економічні умови створюють необхідність скорочення часу їх випуску на ринок, а також забезпечення можливості внесення нових функцій. Це вимагає нового, гнучкішого підходу до проектування, який забезпечив би внесення змін на останній стадії циклу проектування.

Одним із таких підходів є створення комп'ютерів із спеціалізованою (орієнтованою) системою команд (КОСК, англійський термін - ASISC - application-specific instruction-set computer). КОСК орієнтовані на використання в складних системах, які реалізуються у вигляді НВІС. Використання КОСК як компоненти системи на кристалі забезпечує її програмованість (бажану гнучкість процесу проектування) із збереженням основних переваг спеціалізованих архітектур - можливості підвищення продуктивності та зменшення споживаної потужності. Таким чином, КОСК поєднують в собі переваги двох різних класів архітектур: універсальних та спеціалізованих. КОСК - це комп'ютери, система команд яких орієнтована на конкретне використання, наприклад, коміркові телефони, стиск зображень і т. д. Порівняно з КДСК, КОСК мають вищий рівень спеціалізації.

Зазвичай КОСК мають малий набір команд, який включає:

• набір стандартних арифметичних команд, команди роботи з пам'яттю та команди керування програмним потоком, які необхідні для конкретного використання;

• декілька спеціальних команд, наприклад, виконання операцій множення-накопичення даних, цифрової фільтрації, декодування за алгоритмом Вітербі і т. д.

Таким чином, критичні за швидкодією команди можуть бути виконані за мінімальну кількість машинних циклів (можливо за 1 простий цикл) без зберігання у пам'яті проміжних результатів.

Охарактеризувати КОСК можна за допомогою шести основних параметрів. Цими параметрами є: формат даних, формат команд, структура пам'яті, структура регістрової пам'яті, кодування операцій, а також структурні особливості процесора. КОСК зазвичай орієнтовані на обробку даних з фіксованою комою. Розрядна сітка функціональних блоків, шин та пам'яті вибирається, виходячи із потреб задачі. Вони базуються на архітектурі типу регістр-регістр. Регістри регістрового файла процесора є програмно доступними. В КОСК можуть використовуватись два базових типи кодування операцій: мікрокодування та макрокодування.

При мікрокодуванні всі команди виконуються за один машинний цикл. Кожна команда вибирається з пам'яті програм, визначаються всі тракти даних і звертання до пам'яті, які будуть виконані протягом даного машинного циклу. При макрокодуванні деякі команди можуть виконуватися протягом багатьох циклів. Кожна команда визначає всі тракти даних і звертання до пам'яті, які відповідають специфічній трансформації даних, навіть якщо ці дії мають місце в різних машинних циклах. У випадку мікрокодування легше організується конвеєр команд, оскільки машинний код доступний програмісту, а при макрокодуванні ці дії виконує блок керування, тому можлива поява ефектів впливу затримки конвеєра на продуктивність. Цей ефект може бути усунений шляхом введення додаткової апаратури яка його передбачає, але відповідно вимагає ускладнення розробки.

Можливі два варіанти формату команд - ортогональний та кодований. Ортогональний формат складається з фіксованих незалежних керуючих полів. У випадку кодованого формату інтерпретація бітів команди як керуючих полів залежить від значення бітів формату. У випадку ортогонального формату команди розряди всередині керуючого поля можуть бути також закодовані для зменшення його розрядності.

Зазвичай КОСК мають 16- або 32-розрядний кодований формат команди. Прикладна програма розміщується в пам'яті кристала. Формат команди вибирають узгодженим з розміром паралельного порту даних та стандартних блоків пам'яті, які використовуються для зберігання прикладної програми.

КОСК мають ефективну структуру основної та регістрової пам'яті, які забезпечують високу швидкість обміну між різними блоками тракту даних і між трактом даних та пам'яттю. Такі комп'ютери мають один або два блоки пам'яті даних. Для збільшення продуктивності роботи таких комп'ютерів при їх побудові використовується гарвардська архітектура (розділення пам'яті даних та пам'яті програм).

Більшість КОСК використовують гетерогенну структуру регістрової пам'яті, що дозволяє зменшити кількість розрядів команди, збільшуючи швидкість обміну даними. Це означає, що спеціально організовані регістри та регістрові файли прямо під'єднані до спеціальних портів функціональних блоків. Функціональні блоки можуть вибирати операнди або записувати результати тільки в конкретні регістри. Між елементами комп'ютера є тільки ті зв'язки, які є корисними для ефективної реалізації заданого алгоритму.

Архітектура КОСК зазвичай має ряд особливостей, які відділяють її від інших архітектур і не були відображені в попередніх розділах. Далі наведено перелік особливостей існуючих КОСК:

- Як зазначалось вище, система команд КОСК включає спеціалізовані команди, які дозволяють виконувати критичні ділянки заданих алгоритмів за мінімальну кількість машинних циклів та без засилання в пам'ять проміжних результатів. Структура трактів даних таких процесорів спеціально проектується для конкретних алгоритмів.
- Пам'ять даних зазвичай працює з одним або двома процесорами формування адрес, які підтримують безпосередній, прямий та непрямий способи адресації. Також підтримуються спеціальні адресні операції, такі як обчислення модуля при реалізації циркулярних буферів при виконанні фільтрації та підрахунок з інверсним розповсюдженням переносу при виконанні швидкого перетворення Фур'є.
- КОСК обробки мовних, відео- та аудіосигналів часто включають блоки бітових маніпуляцій. Більше того, ці комп'ютери підтримують кілька типів даних з фіксованою комою (наприклад: 2 різних типи 16-розрядних даних двох доповняльних кодів з різною інтерпретацією бінарних значень, 32-розрядний тип двох доповняльних кодів у акумуляторі та тип 8-розрядного цілого в процесорах формування адрес). Перетворення типів даних підтримуються апаратурою тракту даних.
- Більшість КОСК підтримують стандартні команди керування, такі як умовні розгалуження, що базуються на значеннях розрядів в регістрі кодів умов. При цьому в них враховано можливості появи затримок при виконанні операцій переходу, які пов'язані з наявністю конвеєра, реалізовано можливість виконання арифметичних команд та команд пересилання при наявності певної умови, що дозволяє реалізувати умовні ал-

горитми без необхідності умовного зчитування програмного лічильника, а також можливість виконання деяких арифметичних команд резидентно, коли поведінка команди залежить від значення розряду в керуючому регістрі, який може бути записаний іншими операціями.

3.8. Короткий зміст розділу

В даному розділі були розглянуті основні елементи архітектури комп'ютера, які включають організацію пам яті, формати і типи команд, способи адресації. Показано, як кодуються та виконуються команди в комп'ютері. Проведена класифікація команд відповідно до ініційованих ними типів операцій та детально розглянуті команди обробки даних, переміщення даних, передачі керування, введення виведення. Багато рішень має бути прийнято, коли проектується система команд. Наявність значної кількості команд в системі команд комп'ютера призводить до збільшення довжини команди, до збільшення часу вибірки та декодування. Команди фіксованої довжини легше декодувати, але втрачається гнучкість. Реалізація одних команд на основі інших є компромісом між потребою у великих системах команд і бажанні мати короткі команди

Конвеєризація виконання команд один із прикладів паралелізму на рівні команди. Це загальна але складна технологія, яка може підвищити продуктивність виконання послідовності команд. Проте рівень паралелізму може бути обмежений конфліктами в конвеєрі

Є три типи архітектур комп'ютера за типом адресованої пам'яті: стекова, акумуляторна, та на основі регістрів загального призначення. Кожна має свої переваги і недоліки, які потрібно розглядати в контексті застосування запропонованої архітектури

Досягнення в технології пам'яті, привівши до великих її ємностей, викликали потребу в альтернативних способах адресації. Були введені різні способи адресації, включаючи безпосередню, пряму, непряму, базову, індексну, сторінкову і стекову. Наявність цих різних способів забезпечує гнучкість і зручність для програміста без заміни фундаментальних операцій процесора

Розгляд архітектури комп'ютера на рівні системи команд встановлює межу між апаратурою та програмним забезпеченням і дозволяє побачити комп'ютер на рівні, який видимий програмісту, що працює на мові асемблера, або розробнику компіляторів. За складом системи команд комп'ютери можуть бути поділені на наступні типи: комп'ютери із складною, з простою, з доповненою та спеціалізованою системою команд. В розділі ці архітектури розглянуті детально

3.9. Література для подальшого читання

Класифікація різних типів пам'яті залежно від способу доступу до даних наведена в [5]. В роботі [6] було запропоновано новий тип пам'яті, яка за способом доступу до даних належить до пам'яті з впорядкованим доступом. Принципи побудови та роботи пам яті з довільним доступом детально розглянуті в літературі [8, 9]. Асоціативна пам'ять описана в [1].

Порядок виконання команд та формат команди описані в роботах [2, 3]. Системи команд, адресація та формати команди розкриті детально в майже кожній книзі, яка стосується архітектури комп'ютерної системи. Книги [21,22] найповніше охоплюють ці питан-

ня. Багато книг, як наприклад [10, 11, 15, 18], присвячені архітектурі процесора x86 фірми Intel. Для тих, хто зацікавлений у вивченні серії 68000, пропонується література [20, 31].

Належне обговорення конвеєрної обробки команд надає [24]. В [16] подано цікавий короткий огляд конфліктів у конвеєрі. Історія конвеєрної обробки розкрита в [23]. Щоб одержати знання обмежень і проблем з конвеєрною обробкою, варто прочитати [ЗО].

Подивіться [11] для довершеного і зручного введення в сім'ю процесорів Intel. В [14] зроблено опис безадресної машини фірми Burroughs. [28] дає достатнє уявлення про серію IBM 360. В [12] наведено деталі про систему VAX, в якій об'єднана двоадресна архітектура з відпрацьованою системою команд. В [25] надано короткий огляд архітектури SPARC. В [19] зроблено цікавий огляд віртуальної машини Java.

Порівняння архітектур комп'ютерів із складною та спрощеною системою команд зроблено в [21, 22]. Тут же запропоновано архітектуру комп'ютера із спрощеною системою команд DLX. Архітектура комп'ютерів з доповненою системою команд розглянута в [4].

3. ТО. *Література до розділу 3*

1. Искусственный интеллект: В 3-х книгах. Кн. 3. Программные и аппаратные средства: Справочник/ Под ред. В. Н. Захарова, В Ф. Хорошевского. - М.: Радио и связь, 1990. - 191 с

2. Каган Б М. Электронные вычислительные машины и системы. - М.: Энергия, 1979. - 528 с

3. Каган Б М., Каневский М М. Цифровые вычислительные машины и системы М.: Энергия, 1974. - 680 с

4. Мельник А О. Програмовані процесори обробки сигналів. - Львів: Вид-тво Національного університету "Львівська політехніка", 2000. 55 с

5. Угрюмов Е П. Цифровая схемотехника. - СПб.: БХВ - Санкт Петерберг, 2000. - 528 с

6. Мельник А О. Принципи побудови буферної сортувальної пам'яті. Вісник Державного університету "Львівська політехніка". - "Комп'ютерна інженерія та інформаційні технології", № 307, 1996.-С. 65-71.

7. Справочник по цифровой вычислительной технике. Б Н Малиновский и др - К.: Техніка, 1980. - 320 с

8. Шигин А Г., Дерюгин А А. Цифровые вычислительные машины (память ЦВМ). - М.: Энергия, 1975. - 536 с

9. Метлицький Б А., Каверзнев В В. Системы параллельной памяти. Теория, проектирование, применение. Под. ред. В И. Тихонова. - Л., 1989.

10. Abel, Peter. *IBM PC Assembly Language and Programming,* 5th ed., Upper Saddle River, NJ: Prentice Hall, 2001.

11. Brey, B. *Intel Microprocessors 8086/8088, 80186/80188, 80286, 80386, 80486 Pentium, and Pentium Pro Processor, Pentium II, Pentium III, and Pentium IV: Architecture, Programming, and Interfacing,* 6th ed., Englewood Cliffs, NJ: Prentice Hall, 2003.

12. Brunner, R.A. *VAX Architecture Reference Manual,* 2nd ed., Herndon, VA: Digital Press, 1991.

13. Burks, Alice, 8c Burks, Arthur. *Иее First Electronic Computer: The Atanasoff Story.* Ann Arbor, MI: University of Michigan Press, 1988.

14. Hauck, E. A., 8c Dent, B. A. "Burroughs B6500/B7500 Stack Mechanism", *Proceedings ofAFIPS SJCC* (1968), Vol. 32, pp. 245-251.

15. Jones, William. *Assembly Language Programming for the IBM PC Family,* 3rd ed., El Granada, CA: Scott/Jones Publishing, 2001.

16. Kaeli, D., 8c Emma, P. "Branch History Table Prediction of Moving Target Branches Due to Subroutine Returns". *Proceedings of the 18th Annual International Symposium on Computer Architecture,* May 1991.

17. Lindholm, Tim, & Yellin, Frank. *The Java Virtual Machine Specification.* Online atjava.sun.com/docs/books/vmspec/html/VMSpecTOC.cod.html.

18. Messmer, H. *The Indispensable PC Hardware Book.* Reading, MA: Addison-Wesley, 1993.

19. Meyer, J., & Downing, T. *Java Virtual Machine.* Sebastopol, CA: O'Reilly & Associates, 1991.

20. Miller, M. A. *The 6800 Family, Architecture Programming and Applications,* 2nd ed., Columbus, OH: Charles E. Merrill, 1992.

21. D. Patterson, J. Hennessy. Computer Architecture. A Quantitative Approach. Morgan Kaufmann Publishers, Inc. 1996.

22. Patterson, D. A., & Hennessy, J. L. *Computer Organization and Design, Tlie Hardware/Software Interface,* 2nd ed , San Mateo, CA: Morgan Kaufmann, 1997.

23. Rau, B. Ramakrishna, & Fisher, Joseph A. "Instruction-Level Parallel Processing: History, Overview and Perspective". *Journal of Supercomputing 7* (1), Jan. 1993, pp. 9-50.

24. Sohi, G. "Instruction Issue Logic for High-Performance Interruptible, Multiple Functional Unit, Pipelined Computers". *IEEE Transactions on Computers,* March 1990.

25. SPARC International, Inc., *The SPARC Architecture Manual: Version 9,* Upper Saddle River, NJ: Prentice Hall, 1994.

26. Stallings, W. *Computer Organization and Architecture,* 5th ed., New York, NY: Macmillan Publishing Company, 2000.

27. Stern, Nancy. *From ENIAC to UNIVAC: An Appraisal of the Eckert-Mauchly Computers.* Herndon, VA: Digital Press, 1981.

28. Struble, G. W. *Assembler Language Programming: The IBM System/360 and 370,* 2nd ed., Reading, MA: Addison-Wesley, 1975.

29. Tanenbaum, Andrew. *Structured Computer Organization,* 4th ed., Upper Saddle River, NJ: Prentice Hall, 1999.

30. Wall, David W. *Limits of Instruction-Level Parallelism.* DEC-WRL Research Report 93/6, Nov. 1993.

31. Wray, W. C, 8c Greenfield, J. D. *Using Microprocessors and Microcomputers, the Motorola Family.* Englewood Cliffs, NJ: Prentice Hall, 1994.

3.11. Питання до розділу 3

1. Як організовується зв язок між процесором і основною пам яттю?
2. Який порядок виконання команд в комп ютері?
3. Як кодуються команди в комп ютері?
4. Що таке асемблерна мова і для чого використовується асемблер?
5. Як класифікуються команди за типами операцій?
6. Назвіть команди обробки даних
7. Назвіть базові операції зсуву
8. Назвіть команди переміщення даних
9. Поясніть принципи організації послідовного виконання команд і розгалуження
10. Назвіть команди передачі керування
11. Назвіть команди переходу
12. Назвіть команди пропуску
13. Назвіть команди звертання до підпрограм
14. Поясніть принципи конвеєрного виконання команд
15. Якою є продуктивність 4-ярусного конвеєра з тактом 20 не при виконанні 100 команд?
16. Назвіть можливі конфлікти, які можуть сповільнити конвеєр
17. Які використовуються формати команд при роботі з основною пам яттю?
18. Які формати команд використовуються при роботі з регістрами процесора?

19. Які головні критерії вибору формату команд?

20. Поясніть різницю між акумуляторною архітектурою, стековою архітектурою та архітектурою на основі регістрів загального призначення

21. Поясніть різницю між архітектурами системи команд типу регістр регістр, регістр пам'ять і пам'ять пам'ять

22. Які переваги та недоліки команд з фіксованим та зі змінним форматом? Який формат є вживанішим в сучасних комп'ютерах і чому?

23. Яким чином знаходяться дані в пам'яті коли в команді відсутня адресна частина?

24. Яка програма має більше команд: та, що складається з безадресних команд, одноадресних команд, чи з двоадресних команд? Чому?

25. Що таке спосіб адресації?

26. Які є способи адресації пам'яті? їх призначення?

27. Як організовується стекова пам ять?

28. Поясніть порядок організації обчислень при використанні стекової адресації

29. Наведіть приклади використання інфіксної, префіксної та постфіксної форм запису арифметичних виразів

30. Наведіть приклади безпосередньої, прямої, непрямої, відносної та базової адресацій

31. Чим відрізняється індексна адресація від базової?

32. Чому необхідна велика кількість різних способів адресації?

33. Які формати команд використовуються в системі IBM 370? їх відмінності

34. Які формати команд використовуються в машині Cyber-70?

35. Які формати команд використовуються в комп'ютері DLX?

36. Дайте класифікацію архітектур комп'ютера за складом системи команд

37. Яка різниця між комп'ютерами із складною та простою системами команд?

38. Які особливості має комп'ютер з доповненою системою команд?

39. Які переваги має комп'ютер з орієнтованою системою команд?

Розділ 4

ТГрацесар ушЯерамшіаш кти'ютєрси

Процесор - це центральний пристрій комп'ютера. Він виконує задані програмою перетворення інформації та здійснює керування всім обчислювальним процесом і взаємодією складових частин комп'ютерної системи.

Основною функцією процесора як центрального пристрою комп'ютера є виконання послідовності команд, які зберігаються в основній пам'яті. Послідовність операцій, необхідних для виконання однієї команди, називається командним циклом, який може бути поділений на дві основні фази: фазу вибірки та фазу виконання. Спочатку процесор вибирає команду з пам'яті в рамках.фази вибірки. Фаза виконання включає декодування команди, вибірку з основної пам'яті вказаних в команді операндів, виконання операції, яка вказана в полі коду операції команди, а також запам'ятовування результатів. Робота процесора в рамках командного циклу задається послідовністю мікрооперацій, які формує пристрій керування.

Крім виконання арифметичних та логічних операцій, процесор керує роботою інших вузлів комп'ютера. Зокрема, він здійснює керування введенням-виведенням інформації, наприклад пересиланням даних між пристроями введення-виведення та основною пам'яттю.

Основні елементи процесора - арифметико-логічний пристрій, пристрій керування і регістрова пам'ять або, як її ще називають, надоперативний запам'ятовуючий пристрій. До складу регістрової пам'яті, в свою чергу, входять, як було показано в розділі З, наступні вузли - програмний лічильник, регістри: адреси, команди, даних, слова стану програми, а також регістровий файл, який складається з програмно доступних регістрів. Подібним чином і пристрій керування та арифметико-логічний пристрій поділяються на складові, які будуть розглянуті в наступних розділах.

Залежно від зв'язків між функціональними вузлами процесора і організації їх взаємодії розрізняють кілька структур процесора. Далі будуть розглянуті структури, які знайшли використання в сучасних комп'ютерах.

4. *1.* Процесор *комп'ютера із складною системою команд*

4.1.1. Одношинна структура процесора

Почнемо розгляд процесора з аналізу його структури та організації роботи. Однією з найпростіших структур процесора є одношинна структура. Одношинну структуру процесора і його зв'язки з іншими пристроями комп'ютера показано на рис. 4.1. Як бачимо, до складу процесора входять пристрій керування, арифметико-логічний пристрій АЛП з вхідним РгУ та вихідним Рг2 регістрами, і регістрова пам'ять, до складу якої входять регістр команд РгК, регістр даних РгД, регістр адреси РгА, програмний лічильник ПЛ, та регістро-

вий файл - стек програмно доступних регістрів РгО, Ргі... Рг(п-І). Обмін інформацією між названими пристроями здійснюється через спільну внутрішню шину процесора. Зв'язок процесора з основною пам'яттю проводиться через регістри адрес РгА та даних РгД.

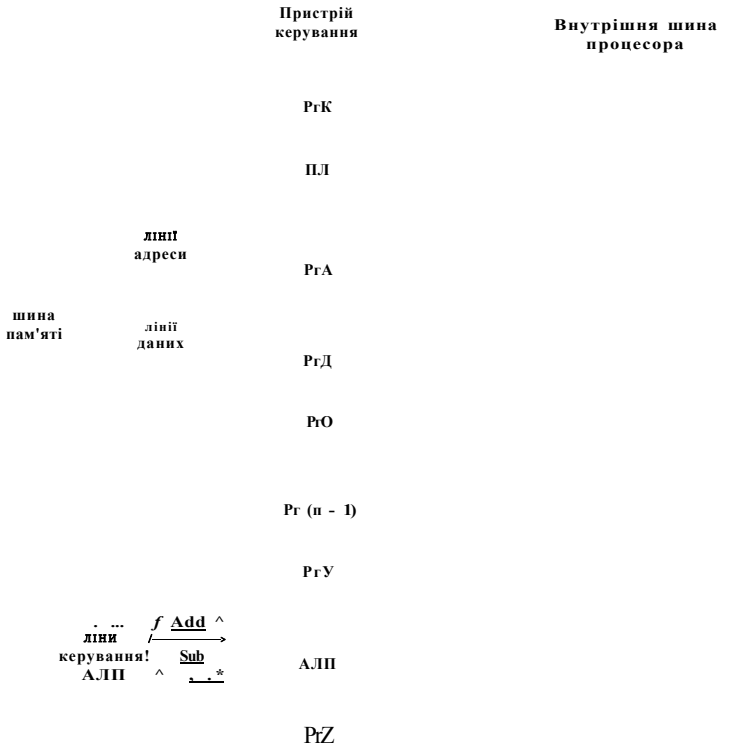


Рис. 4.1. Однотипна структура процесора

РгА зберігає адресу даного або команди при зверненні до основної пам'яті. РгК зберігає команду після її зчитування з основної пам'яті. РгД зберігає операнд при його запису або зчитуванні з основної пам'яті. ПЛ - програмний лічильник, який підраховує команди та зберігає адресу поточної команди.

Регістри загального призначення $Рг_0$-$Рг_{(п-1)}$ регістрового файла є програмно доступними. Вони можуть використовуватися програмістом в якості адресних регістрів, індексних регістрів при виконанні операцій модифікації адрес або в якості регістрів для зберігання проміжних результатів обчислень. Більшість комп'ютерів мають в складі процесора тригери для зберігання бітів стану процесора, або як їх ще називають, прапорців. Кожен прапорець має спеціальне призначення. Частина прапорців вказує на результати арифметичних і логічних операцій: додатній результат (Р), від'ємний результат (N), нульовий результат *(Х),* перенос (С), арифметичне переповнення (V) тощо.

Різні команди вказують процесору, коли встановити чи очистити ці тригери. Інша частина прапорців вказує режими захисту пам'яті. Існують також прапорці, які вказують пріоритети виконуваних програм. В деяких процесорах додаткові тригери слугують для зберігання кодів умов, формуючи регістр кодів умов. Взяті разом описані прапорці формують слово стану програми (ССП), а відповідні тригери - регістр ССП.

4.1.2. Основні операції процесора

Можна виділити дві фази виконання команди в процесорі: фаза вибірки та фаза виконання. Фаза вибірки передбачає вибірку вмісту комірки основної пам'яті, в якій зберігається команда, за значенням програмного лічильника ПЛ і засилання команди в РгК (РгК := [[ПЛ]]), а також прирощення ПЛ на одиницю: [ПЛ] := [ПЛ] + 1. Фаза виконання команди передбачає її дешифрування та виконання операцій, вказаних в коді операції команди. Для того, щоб побачити, як обидві фази виконуються в процесорі, розглянемо його основні операції, до виконання яких задіяні представлені на рис. 4.1 елементи.

4.1.2.1. Вибірка слова з пам'яті

Нехай адреса комірки основної пам'яті знаходиться в регістрі Ргі, а дані потрібно розмістити в регістрі Рг2.

Для вибірки із основної пам'яті необхідно виконати наступну послідовність операцій:

1. РгА := Ргі (запис до регістра адреси РгА вмісту регістра Ргі).
2. Зчитування (виконання операції зчитування команди з комірки основної пам'яті до регістра РгД шляхом подання сигналу Read на вхід керування режимом роботи основної пам'яті та сигналу запису до регістра РгД).
3. Чекання на сигнал підтвердження зчитування.
4. Рг2 := РгД (запис до регістра Рг2 даного з регістра РгД).

Пункт 3 виконується при асинхронному принципі обміну між процесором і основною пам'яттю, коли потрібно чекати на сигнал підтвердження зчитування. При синхронному принципі обміну чекати на сигнал підтвердження зчитування не потрібно, оскільки до подання сигналу запису в регістр РгД передбачається гарантована наявність даного на його вході.

4.1.2.2. Запам'ятовування слова в пам'яті

Нехай слово, яке запам'ятовується в основній пам'яті, знаходиться в регістрі Рг2, а адреса - в регістрі Ргі. Тоді послідовність операцій буде наступною:

1. РгА := Ргі (запис до регістра адреси РгА вмісту регістра Ргі).
2. РгД := Рг2 (запис до регістра даних РгД вмісту регістра Рг2), запис (виконання операції запису слова з регістра РгД до комірки основної пам'яті шляхом подання сигналу Write на вхід керування режимом роботи основної пам'яті).
3. Чекання на сигнал підтвердження запису (при асинхронному принципі обміну між процесорам і основною пам'яттю).

4.1.2.3. Обмін даними між регістрами

Символьне зображення вхідних і вихідних елементів регістрів процесора показане на рис. 4.2 у вигляді ключів, які пропускають або не пропускають інформацію з входу на вихід, залежно від значення сигналів керування на їх входах. Тут сигнал керування входом і-го регістра Ргі позначено як Ргі in, а сигнал керування виходом і-го регістра Ргі позначено як Ргі out. В конкретній схемі регістра це можуть бути, наприклад, вхід запису числа до регістра та його тристабільний вихід.

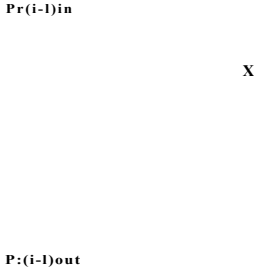


Рис. 4.2. Фрагмент схеми процесора з вхідними та вихідними елементами регістрів

Подання 1 на вхід керування регістра Priout з'єднує вихід регістра Ргі з шиною, а подання 1 на вхід керування регістра Ргііп записує число з шини в регістр Ргі. Подаючи на регістри вказані сигнали, можна переписувати числа з одного регістра в інший наприклад, для перезапису числа із регістра РгЗ до регістра Рг5 необхідно подати наступні сигнали: РгЗош, Рг5іп.

4.1.2.4. Виконання арифметичних і логічних операцій

Арифметико-логічний пристрій (АЛП) процесора призначений для виконання операцій обробки даних. Тип виконуваної операції вказується кодом на вході керування АЛП. В АЛП, зокрема, виконуються такі операції: зсув - зміщення кодів, які зберігаються в регістрах регістрового файлу, вліво або вправо на задане число розрядів; додавання до слова 1 або -1 - операція рахунку; дешифрування - перетворення двійкових кодів у сигнали (однорядний код); шифрування - перетворення однорядного коду в двійковий; порівняння - визначення відношення старшинства двох чисел або їх рівності; порозрядне доповнення - формування оберненого коду; порозрядні логічні множення і додавання двох чисел; порозрядне додавання двох чисел по модулю; додавання двох чисел. Звичайно, цей перелік може бути розширений.

Розглянемо виконання операції додавання двох чисел з регістрів Ргі і Рг2 з записом результату в регістр РгЗ на одношинній структурі процесора, представленій на рис. 4.1:

1) Priout, РгУіп (запис до вхідного регістра АЛП РгУ вмісту регістра Ргі).

2) Pr2out, Add, PrZin (подання числа з регістра Рг2 на внутрішню шину процесора, звідки воно поступає на другий вхід АЛП, виконання в АЛП операції додавання чисел з регістра РгУ та з шини і запам'ятовування результату в регістрі PrZ).

3) Рйоиі, РгЗіп (запис до регістра РгЗ вмісту регістра Рг2).

Подібним чином виконуються інші вище перераховані операції. Необхідно відзначити, що сигнали Ргіоиі та Рг)іп, де і та; - номери регістрів, мають бути рознесеними в часі для забезпечення коректного перезапису інформації з одного регістра до іншого з врахуванням часу спрацювання їх вхідних та вихідних схем, ємності провідників шини та затримки в комбінаційних схемах АЛП. Цей час визначає такт роботи процесора.

4.1.3. Багатошинна структура процесора

Наведена на рис. 4.1 одношинна структура процесора є достатньо простою. Тому вона широко використовується при побудові процесорів реальних комп'ютерів. Зокрема, за такою схемою побудовано більшість мікроконтролерів, від яких не вимагається висока швидкодія. Якщо ж така вимога існує, то застосовується багатошинна організація процесора, в якій завдяки наявності багатьох шин забезпечується можливість паралельного обміну інформації між функціональними вузлами процесора і, тим самим, суттєво підвищується швидкість опрацювання інформації. Як приклад на рис. 4.3 показано можливий варіант двошинної структури процесора.

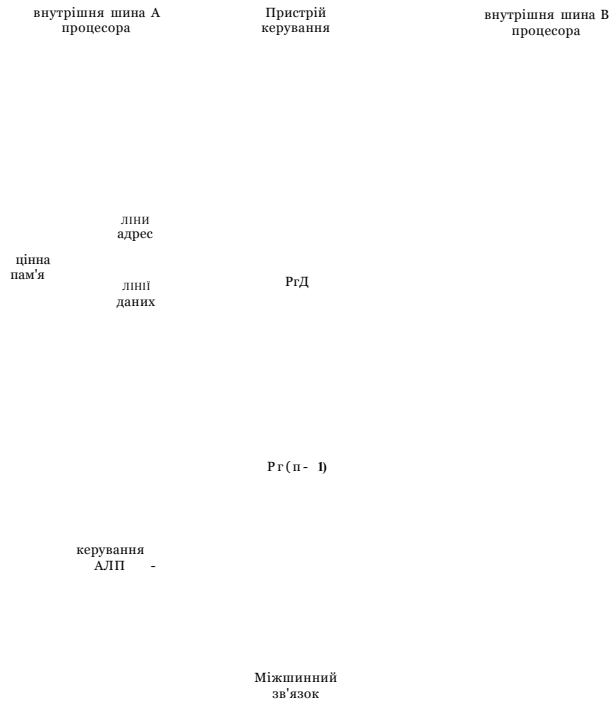


Рис. 4.3. Двошинна структура процесора

Тут входи та виходи регістрів процесора приєднані до різних шин, що дозволяє одночасно здійснити обмін між двома парами регістрів, а не між однією, як це було в одношинній структурі процесора. Тим самим в два рази зростає кількість переданої одночасно інформації. Міст зв'язку між шинами, який може бути відчинений або зачинений, призначений для обміну інформацією між шинами.

Розглянемо виконання на цій структурі процесора операції РгЗ : = РгО + Ргі:

1) PrOout, Prlout, Add, PrZin (подання на шину А числа з регістра Ргі та на шину В числа з регістра РгО, виконання в АЛП операції додавання та запис результату до регістра PrZ).

2) PrZout, РгЗіп (запис числа з регістра PrZ до регістра РгЗ через міст М зв'язку між шинами).

Як видно, для виконання операції додавання тут потрібно лише два такти, тоді як в одношинній структурі процесора було потрібно три такти.

Зрозуміло, що із збільшенням кількості шин швидкість обміну зростає. Так, в тришинній структурі процесора операція додавання двох чисел буде виконана за один такт.

Збільшення кількості шин ускладнює структуру процесора та збільшує кількість необхідного обладнання на його побудову. Адже кожна додаткова шина - це велика кількість додаткових провідників, які займають багато місця на кристалі. Однак потреба підвищення продуктивності змушує розробників застосовувати саме багатошинну структуру процесора. Більшість процесорів сучасних високопродуктивних комп'ютерів є багатошинними.

4.7.4. *Приклади виконання операцій в процесорі*

4.1.4.1. Виконання операції додавання двох чисел

Об'єднаємо послідовність елементарних операцій, необхідних для виконання однієї команди. Розглянемо команду "Додати вміст деякої комірки основної пам'яті до вмісту регістра Ргі із записом результату до цього ж регістра, причому адреса комірки основної пам'яті задана в адресному полі команди".

Виконання цієї команди вимагає наступних дій:

1. Вибірки команди з основної пам'яті та її запис до регістра команди РгК.

2. Вибірки першого операнда з основної пам'яті та його запис до одного з регістрів надоперативної пам'яті процесора.

3. Виконання додавання.

4. Засилання результату в регістр Ргі.

Програма виконання цих дій на одношинній структурі процесора (рис. 4.1) буде мати наступний вигляд:

Фаза вибірки :

1) ПЛоиг, РгАіп, зчитування, очищення регістра PrY, включення переносу та операції додавання в АЛП, PrZin (адреса команди з програмного лічильника ПЛ подана на шину та записалась до регістра адреси РгА, на вхід керування режимом роботи основної пам'яті подано сигнал Read, на вхід скиду регістра PrY подано сигнал Reset, в АЛП виконалась операція додавання 1 до вмісту програмного лічильника ПЛ та її результат записався в регістр PrZ).

2) PrZout, ПЛіп, чекання підтвердження сигналу зчитування (результат додавання з регістра PrZ записався в програмний лічильник ПЛ, команда з основної пам'яті записалась в регістр РгД).

3) РгДоШ:, РгКіп (команда з регістра даних РгД записалась до регістра команди РгК).

Фаза виконання:

4) (Поле адреси PrK)out, PrAin, зчитування (адреса з поля адреси регістра команди PrK переписується до регістра адреси PrA, на вхід керування режимом роботи основної пам'яті подано сигнал Read).

5) Prlout, PrYin, чекання підтвердження сигналу зчитування (число з регістра Prі записалось до регістра PrY, а число з основної пам'яті записалось до регістра PrД).

6) PrДоиі, Add, PrZin (число з регістра PrД поступило на шину та додалося в АЛП до числа з регістра PrY, а результат записався в регістр PrZ).

7) PrZout, Prlin, End (число з регістра PrZ переписалось в регістр Prі, кінець виконання операції).

Тут Add - код операції додавання, який поступає на вхід АЛП та вказує йому тип виконуваної операції.

4.1.4.2. Виконання операції переходу

Адреса переходу зазвичай одержується шляхом додавання значення X, яке знаходиться в адресному полі команди, до біжучого значення ПЛ.

Програма має наступний вигляд при виконанні безумовного переходу:

1) ПЛоиі, PrAin, зчитування, очищення регістра У, включення переносу та операції додавання в АЛП, PrZin (адреса команди з програмного лічильника ПЛ подана на шину та записалась до регістра адреси PrA, на вхід керування режимом роботи основної пам'яті подано сигнал Read, на вхід скиду регістра PrY подано сигнал Reset, в АЛП виконалась операція додавання 1 до вмісту програмного лічильника ПЛ та її результат записався в регістр PrZ).

2) PrZout, ПЛin, чекання підтвердження сигналу зчитування (результат додавання з регістра PrZ записався в програмний лічильник ПЛ).

3) PrДош% PrKin (команда з регістра даних PrД записалась до регістра команди PrK).

4) nvTout, PrYin (число з програмного лічильника ПЛ записалось до регістра PrY).

5) (Поле адреси PrK)out, ADD, PrZin (число з поля адреси регістра команди PrK поступило на шину та додалося в АЛП до числа з регістра PrY, а результат записався в регістр PrZ).

6) PrZout, ПЛin, End (число з регістра PrZ переписалось в програмний лічильник ПЛ, кінець виконання операції).

При виконанні умовного переходу по значенню ПЛ = 0 крок 4 заміниться на наступний:

7) ПЛоиI, PrYin, if ПЛ=0 then End (запис вмісту програмного лічильника до регістра PrY і його аналіз на рівність 0. Якщо це так - кінець виконання операції).

4.1.5. Особливості побудови процесора комп'ютера із складною системою команд

Вище розглядалася структура та організація роботи процесора комп'ютера, який із середини вісімдесятих років минулого століття був віднесений до комп'ютерів з складною системою команд КССК (CISC). Аналізуючи його роботу і розглянуті в розділі 2 формат команди та склад системи команд названого комп'ютера, можна прийти до висновку, що для процесора комп'ютера із складною системою команд характерні наступні особливості:

• виконання команди за багато тактів, оскільки для цього потрібно здійснити багаторазові операції звернення до основної пам'яті та до програмно-доступних регістрів процесора;

• орієнтація АЛП на виконання великої кількості операцій, що пов'язано з розширеним складом системи команд;

• складна система розпізнавання команди, що пов'язано з великою кількістю методів адресації та великою кількістю форматів команд різної розрядності;

• програмне дешифрування команд з метою зменшення затрат обладнання;

• складна організація конвеєризації виконання команд, що пов'язано, в першу чергу, з різнотипністю їх виконання;

• орієнтація структури на виконання команд типу регістр-пам'ять та пам'ять-пам'ять.

Вказані особливості стримують побудову високопродуктивних комп'ютерів на основі процесора розглянутого типу. Вони були враховані при створенні процесорів комп'ютерів із простою системою команд.

4.2. Процесор комп'ютера з простою системою команд

4.2.1. Вимоги до процесора комп'ютера з простою системою команд

При розгляді системи команд комп'ютера ми ознайомилися з архітектурою комп'ютерів із простою системою команд КПСК (RISC). Виходячи з основних принципів реалізації цих комп'ютерів, можна виділити наступні вимоги, яких необхідно притримуватися при побудові їх процесора:

• Довільна комп'ютерна команда, незалежно від її типу, має виконуватися за один такт (чи однотактовий цикл).

• Пристрій керування та арифметико-логічний пристрій процесора мають орієнтуватися на виконання мінімальної кількості спрощених команд, що статистично переважають у програмах, причому в системі команд відносно небагато операцій та режимів адресації операндів (способів адресації).

• Команди обробки даних мають реалізуватися лише у формі "регістр-регістр". Обміни з основною пам'яттю виконуються лише за допомогою команд завантаження/запису (архітектура load/store).

• Дешифрування команд із спрощеними форматами має виконуватися лише апаратно, аби збільшити швидкодію.

• Необхідно забезпечити високий рівень конвеєризації виконання команд.

• Регістрова пам'ять має включати велику кількість програмно-доступних регістрів.

При цьому необхідно проводити оптимізацію структури процесора, що проектується, з метою забезпечення найшвидшого виконання обраних команд та передбачити можливість додавання до отриманого списку інших команд, якщо вони не ускладнюють процесора.

4.2.2. Базові принципи побудови процесора комп'ютера з простою системою команд

Для ілюстрації базових принципів побудови процесора комп'ютера з простою системою команд ми використаємо архітектуру комп'ютера, яка була запропонована для навчальних цілей Джоном Хеннессі та Дейвідом Паттерсоном і отримала назву DLX. Ця

архітектура узагальнює особливості архітектур наступних сучасних комп'ютерів: AMD 29000, DEC3100, HP850, IBM801, Inteli860, MIPS M/120A, MIPS M/1000, M88000, RISC1, SGI 4D/60, SPARCstation-1, SUN-4/110, SUN-4/260.

Регістровий файл процесора комп'ютера DLX вміщує 32 регістри загального призначення (R0...R31) для зберігання цілих чисел, 32 регістри (F0...31) для зберігання даних з рухомою комою. Набір команд цього комп'ютера включає типові арифметичні й логічні операції, операції з фіксованою та рухомою комою, операції пересилання даних, операції керування потокоїМ команд і системні операції. У арифметичних командах використовується триадресний формат, типовий для комп'ютерів з архітектурою КПСК, а для звернення до пам'яті використовуються операції завантаження і запису вмісту регістрів у пам'ять.

Основою проектування структури процесора комп'ютера з простою системою команд є часова діаграма виконання команд з найбільшою складністю, до числа яких належить, зокрема, команда завантаження слова. Розглянемо цикл виконання команди вибірки з основної пам'яті (завантаження) слова LW R5, 16(R26). В комп'ютері DLX командний цикл поділений на п'ять фаз. Тому для виконання вказаної команди потрібно виконати наступні фази:

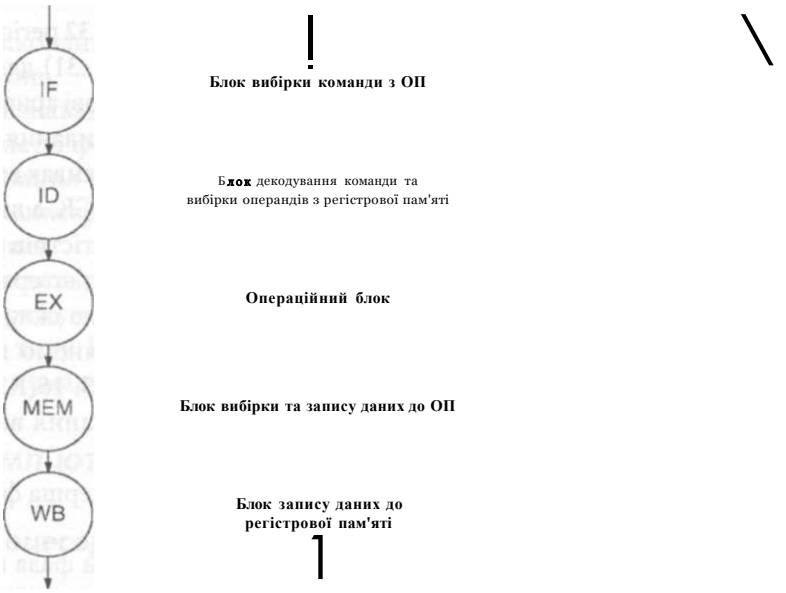
- вибрати зазначену команду з основної пам'яті (перша фаза виконання команди із назвою IF (Instruction Fetch);
- декодувати команду та вибрати операнди (друга фаза виконання команди із назвою ID (Instruction Detecting);
- виконати команду, тобто обрахувати виконавчу адресу операнда 16 + [R26] (третя фаза виконання команди із назвою EX (Execution);
- вибрати операнд із основної пам'яті (четверта фаза виконання команди із назвою MEM (Memory);
- переслати вибраний з основної пам'яті операнд до регістра R5 регістрового файла (п'ята фаза виконання команди із назвою WB (Write Back).

Використані назви фаз дещо узагальнюють притаманну лише команді LW семантику кожної окремої фази. Це коректно, бо нашою метою є наближення до такої послідовності фаз, яка задовольняє вимогам будь-якої команди із заданої до реалізації множини з метою досягнення найбільшої швидкодії. Інші команди не завжди вимагають реалізації усього переліченого набору фаз, тому що мають меншу часову складність.

На рис. 4.4 наведена часова діаграма виконання п'ятифазової команди завантаження LW. Залежно від структури процесора ця команда може бути виконаною за різний час, який буде складатися з суми проміжків часу, необхідних для виконання кожної фази. Розглянемо підхід до побудови процесора з тим, щоб задовольнити вимогу, згідно з якою довільна комп'ютерна команда, незалежно від її типу, має виконуватися за один такт (чи однотактовий цикл), яка ставиться до процесорів комп'ютера з простою системою команд.

Рис. 4.4. Часова діаграма виконання п'ятифазової команди завантаження LW

Представимо алгоритм виконання команди у вигляді потокового графа, кожна з вершин якого позначає оператор відповідної фази виконання команди (рис. 4.5а).



а б

Рис. 4.5 а - отоковий граф виконання команди; б - структура процесора, орієнтованого на його реалізацію

Для того, щоб команда виконувалася за один такт, потрібно апаратно відобразити алгоритм її виконання, тобто поставити у відповідність кожному оператору алгоритму функціональні вузли процесора, які їх виконують, та з'єднати їх між собою згідно із зв'язками вершин потокового графа алгоритму. Тоді структура процесора комп'ютера з простою системою команд, який виконує названі фази, може бути подана наступним рисунком (рис. 4.5б). Як бачимо, процесор містить п'ять послідовно з'єднаних блоків: вибірки команди з основної пам'яті, декодування операндів та вибірки команди з регістрової пам'яті, операційний, вибірки та запису даних до основної пам'яті, запису даних до регістрової пам'яті. Кожен з цих блоків виконує відповідну фазу командного циклу та передає результати до наступного блоку. Результатом послідовної роботи цих блоків є виконання команди

Деталізована структура процесора комп'ютера *ОІХ,* яка побудована на основі описаного вище підходу, представлена на рис. 4.6.

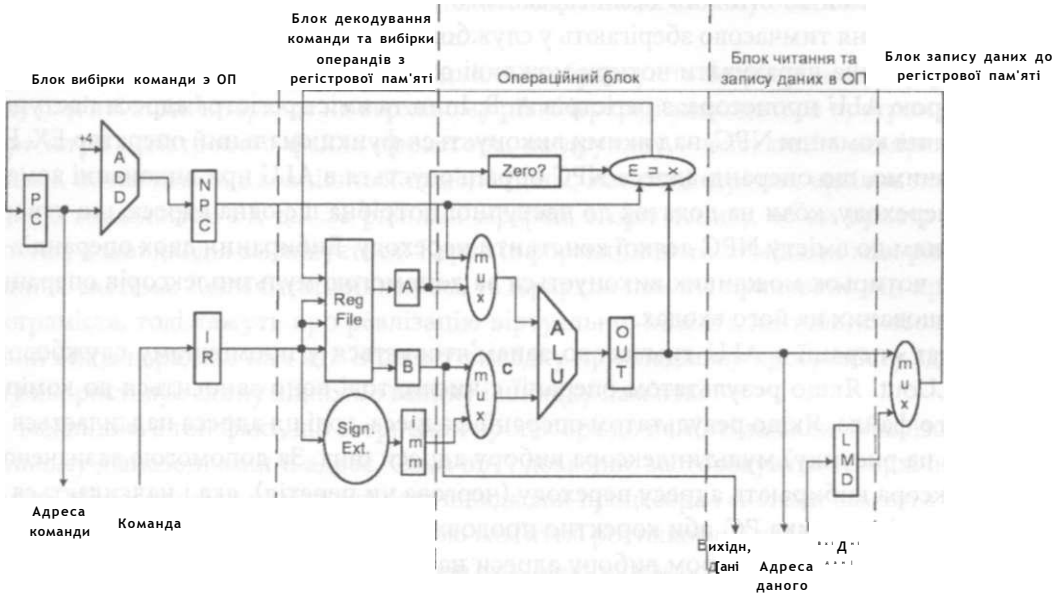


Рис. 4.6. Структура процесЬра комп'ютера з простою системою команд

На цій схемі лініями відділені блоки процесора, показані на рис. 4.5б, які виконують відповідну фазу командного циклу потокового графа алгоритму виконання команди. Перший оператор IF виконується на наступних елементах: програмному лічильнику PC (Program Counter), суматорі Adder та двох регістрах NPC і IR. Вміст програмного лічильника PC визначає адресу команди в основній пам'яті. Комбінаційний суматор Adder обраховує адресу наступної за чергою виконання команди. При цьому враховано, що впорядкована послідовність команд (програма) складається з чотирибайтових команд (усі команди мають формати довжиною 32 біти), які розміщено в основній пам'яті за послідовними адресами 0,4, 8, C і т. д. Через це константа зсуву адреси (пересування покажчика на наступну за чергою команду) дорівнює +4. Визначене за допомогою суматора значення адреси вибирання наступної команди зберігається у регістрі NPC (next PC). Зчитаний з основної пам'яті код поточної команди записується до регістра команди IR.

Поля щойно вибраної команди містять адреси програмно-доступних регістрів регістрової файла процесора. Вміст зазначених полів формату команди в рамках оператора ID надсилається на адресні входи регістрової пам'яті Regs, а відповідні надісланим адресам коди операндів завантажуються до внутрішніх, програхмно-недосяжних, тобто службових, регістрів A і B.

Існує ще один тип операнда з назвою "безпосередній" (Imm). Його задають прямо у форматі команди. Як правило, довжина безпосереднього операнда не перевищує половини довжини формату команди. В комп'ютері DLX безпосередній операнд має довжину 32/2 = 16 бітів. У той самий час бажано зафіксувати довжину формату даних такою, що дорівнює довжині формату команди, адже різноманіть довжин форматів суттєво пригальмовує комп'ютер. Якщо усі формати даних, як і формати команд, матимуть довжину 32 біти, тоді безпосередньому операнду не вистачатиме ще 16 бітів, аби бути стандартним за довжиною. Тому тут використаний комбінаційний вузол Sign Extend, який виконує

знакове розширення 16-бітового безпосереднього операнда до 32-х бітів. Результат знакового розширення тимчасово зберігають у службовому регістрі Imm.

В цілому можна нарахувати чотири можливі операнди на вході арифметико логічного пристрою ALU процесора: з регістрів A, B, Imm та вміст регістра адреси наступної для виконання команди NPC, над якими виконується функціональний оператор EX. Наперед зазначимо, що операнд-адреса NPC опрацьовується в ALU при виконанні команд умовного переходу, коли на додаток до наступної потрібна ще одна адреса, що утворена додаванням до вмісту NPC деякої константи переходу. Вибирання двох операндів на вхід ALU із чотирьох можливих виконується за допомогою мультиплексорів операндів mux, розташованих на його входах.

Результат операції з ALU тимчасово запам'ятовується у проміжному службовому регістрі ALUout. Якщо результатом операції є число, тоді воно заноситься до комірки регістрового файла. Якщо результатом операції є адреса, тоді ця адреса надсилається до (верхнього на рисунку) мультиплексора вибору адреси mux. За допомогою зазначеного мультиплексора вибирають адресу переходу (чергова чи перехід), яка і надсилається до програмного лічильника PC, аби коректно продовжити виконання програми.

Керування мультиплексором вибору адреси наступної команди покладено на вузол Zero?, де вміст службового регістра A порівнюється із нулем (дорівнює нулю, більше нуля, менше нуля і т. д., залежно від виду виконуваної у поточний час операції умовного переходу). Результат порівняння є бінарним логічним значенням (так або ні). Саме цей бінарний результат керує роботою мультиплексора вибирання адреси наступної команди.

При виконанні фази MEM результат-адреса з виходу ALU надсилається до основної пам'яті як отримана адреса комірки цієї пам'яті (для команд збереження/завантаження)

Результатом на виході правого на рисунку мультиплексора може бути або вміст основної пам'яті (при виконанні команди завантаження LW слова з основної пам'яті до регістра регістрового файла), або результат виконання арифметичної, зсувної, логічної чи іншої операції в ALU (наприклад, при виконанні команд ADD, SUB і т. д.). Такий результат в рамках виконання фази WB засобами мікропрограмування зберігають в регістрі регістрового файла. Отже, зазначений мультиплексор, керований регістром поточної команди, комутує на вхід регістрового файла потрібну інформацію

Таким чином, апаратно відобразивши потоковий граф алгоритму виконання команди, вдалося забезпечити вимогу, щоб вона виконувалася за один такт. Як видно з вище приведеного опису роботи процесора, для спрощення пояснення тут були використані проміжні регістри для запису операндів, які можуть бути видалені. Порівняно з процесором комп'ютера із складною системою команд виконання команди в приведеній структурі процесора суттєво спростилося. Далі розглянемо це детальніше, але спочатку проведемо аналіз взаємодії процесора з основною пам'яттю.

4.2.3. Взаємодія процесора з пам'яттю в комп'ютері з простою системою команд

Відомий так званий парадокс пам'яті - пам'ять може мати малий об'єм, проте бути швидкою і задовольняти вимоги процесора щодо швидкодії, або мати відносно великий об'єм і бути повільною. Немає пам'яті відносно великої і, водночас, швидкої. Ставити зараз питання про основну пам'ять, об'єм якої задовольняє системного програміста (сис-

темні програми) є, безперечно, нереальним завданням. Об'єму основної пам'яті завжди не вистачає і, можливо, не вистачатиме.

Аби подолати зазначену невідповідність, вибудовують багаторівневу ієрархічну систему пам'яті комп'ютера, де на верхньому рівні ієрархії знаходяться програмно-доступні регістри регістрового файла процесора, на другому - кеш, потім комірки основної пам'яті, потім система зовнішньої пам'яті (диск, магнітні стрічки, архівна пам'ять з лазерною технологією і т. д.). За рівнями ієрархії, згори донизу, об'єм пристроїв пам'яті зростає, а швидкодія зменшується. Керує інформаційними обмінами між рівнями операційна система. Коли багаторівневість та ієрархію пам'яті приховано від прикладного програміста, тоді кажуть про реалізацію віртуальної пам'яті, де, ніби, скасовано зазначений вище парадокс пам'яті. В цьому випадку прикладному програмісту здається, що він використовує єдину відносно велику і швидку пам'ять.

Важливим є той факт, що звернення процесора до пам'яті завжди локалізовано в невеликому діапазоні змін її адрес. Саме він і дозволяє застосовувати ієрархічну систему пам'яті, аби розв'язати невідповідність швидкодій процесора і системи пам'яті з одним рівнем ієрархії. Між процесором і основною пам'яттю розташована кеш пам'ять (рис. 4.7) - це швидка буферна пам'ять невеликого об'єму. Кеш пам'ять працює на близькій тактовій частоті до процесора і не пригальмовує його роботу. Кеш (cache у перекладі з англійської - тайник) лишається прозорою для програміста, тому що система команд процесора, як правило, не містить команд роботи з кеш пам'яттю.

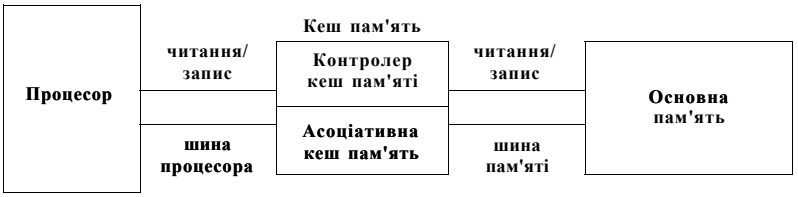


Рис. 4.7. Кеш пам'ять у складі ядра комп'ютера

При поясненні роботи кеш пам'яті можна допустити, що процесор також не «бачить» кеш і генерує адреси пам'яті так, ніби кеш пам'яті не має. Проте кеш пам'ять, як правило, існує, і на апаратному рівні перехоплює сигнали процесора читання/запису, а коли треба, то надає процесору швидкі копії інформаційних кодів, які тимчасово зберігає у власній робочій пам'яті. Якщо кеш пам'ять спроможна підмінити собою основну пам'ять (у понад 96-98 відсотків випадків), тоді вона за рахунок власних ресурсів задовольняє запит процесора. Процесор не пригальмовується і продовжує працювати на повній швидкості. Коли «підміна» основної пам'яті неможлива (менше від двох-чотирьох відсотків випадків), тоді кеш пам'ять залучає до роботи основну пам'ять, обмін з якою суттєво пригальмовує процесор.

Усі завдання, пов'язані із перехопленням запитів від процесора до основної пам'яті, вирішує контролер кеш пам'яті, який є її складовою частиною. Другою частиною кеш пам'яті є невелика робоча пам'ять, де зберігають вміст копій комірок основної пам'яті, що брали участь в обслуговуванні останніх, тобто найбільш «свіжих» запитів процесора. Важливо, що вміст комірок основної пам'яті копіюється до кеш пам'яті разом із своїми адресами. Саме ці копійовані адреси і дозволяють контролеру кеш пам'яті приймати

рішення про спроможність задовольнити конкретний процесорний запит без залучення до обміну повільної основної пам'яті.

Спрощений варіант структури комп'ютера, в якому використовується кеш пам'ять, подано на рис. 4.8. Пристрій керування надсилає керуючі сигнали до процесора та основної пам'яті. З процесора сигнали станів, якими можуть бути біти регістра команди і інше, надходять до пристрою керування, аби реалізувати розгалуження мікропрограми.

Для зберігання даних і команд використано розділені кеш пам'яті даних і команд Гарвардської архітектури. В свою чергу, кеш пам'яті зв'язані з єдиною пам'яттю Принстонської архітектури. Ясно, що обмін в підсистемі "основна пам'ять - кеш пам'ять даних» є двостороннім, а в підсистемі «основна пам'ять - кеш пам'ять команд" - одностороннім.

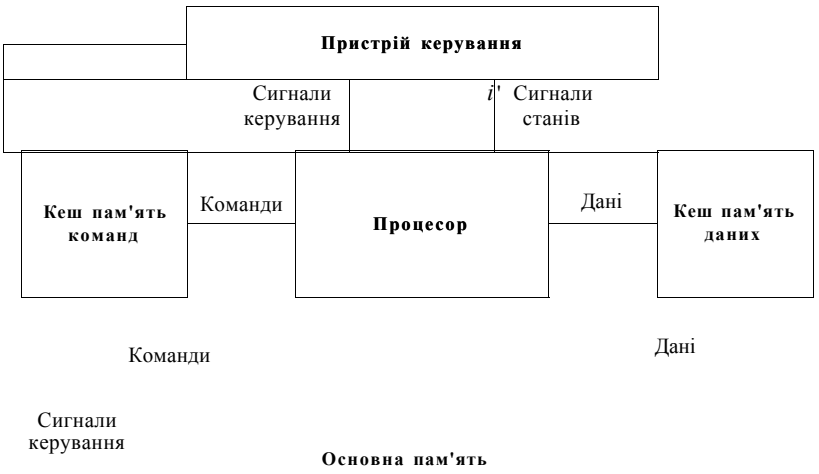


Рис. 4.8. Спрощена структура комп'ютера ОЬХ

Можна побачити, що наведена структура ефективно поєднала риси Принстонської та Гарвардської архітектур.

Надалі при розгляді принципів виконання команд в процесорі будемо розглядати і його взаємодію з основною пам'яттю через кеш пам'ять команд та даних.

4.2.4. *Виконання команд в процесорі комп'ютера з простою системою команд*

4.2.4.1. Фаза вибирання команди

Виконання команд в процесорі комп'ютера з простою системою команд розглянемо на прикладі процесора комп'ютера *DLX,* структура якого була розглянута в п. 4.2.2.

В рамках фази вибирання команди 1Т виконуються наступні мікродії (мікрооперації):

$$Ш = IM[PC];$$

$$ОТС = PC+4.$$

Виконання першої мікродії спричинює завантаження до 32-бітового регістра поточної команди IR вмісту чотирьох послідовно розташованих комірок пам'яті команд, починаючи від адреси (PC) + 0 і завершуючи адресою (PC) + 3. Як і завжди, за допомогою (PC) позначено вміст програмного лічильника PC.

Друга мікродія (NPC = PC + 4) вираховує "планову" адресу наступної за чергою команди з послідовного потоку. Тобто тут визначається і тимчасово зберігається у регістрі NPC вміст програмного лічильника. "Логічне" вибирання вмісту чотирьох послідовно розташованих однобайтових комірок замість однієї чотирибайтової підкреслює те, що навіть на рівні мікродій адресування комірок пам'яті вказують логічно. За умови коли адреса байта кратна чотирьом, тобто дотримано вирівнювання границь адрес команд, із пам яті, одразу (за одне звернення) вибирають чотири байти і навіть цілі пакети по чотири байти. Отриманий з пам'яті даних код тлумачать як 32-розрядне слово, а з пам'яті команд - як одну команду

Зауважимо, що обидві наведені мікродії теоретично є сумісними в часі. Саме тому вони можуть і мають виконуватися паралельно. Ці мікродії утворюють єдину мікрокоманду

Важливою властивістю фази IF є те, що вона не приймає до уваги існування відомого «парадокса пам'яті». Дійсно, у фазі IF за часовими витратами мікродія вибирання з повільних комірок пам'яті команд є тотожною мікродії, що працює з внутрішніми, а саме тому надшвидкими регістрами процесора

4.2.4.2. *Фаза декодування команди*

В рамках фази декодування команди ID виконуються наступні мікродії (мікрооперації):

$$A = Regs[IR_{6} \quad Ji$$
$$B = Regs\ [IR_{u} \quad J;$$
$$Imm = ((TR_{16})^{16} \#\# IR_{16\,31}).$$

У наведеному мікрокоді, що складений з трьох сумісних у часі, тобто придатних до одночасного (паралельного) виконання мікродій, вжито наступні позначення:

- A, B, Imm - внутрішні службові 32-розрядні регістри для проміжного збереження кодів; всі три регістри є "прозорими" для програміста в тому сенсі, що вони аж ніяк не відбиті в машинних командах.
- Regs [address] - номер регістра регістрового файла процесора;
- R_{x} - поле регістра команд, яке містить групу послідовно розташованих бітів - від біта з номером X до біта з номером Y включно; звернемо увагу на нумерацію бітів у слові процесора - лівий біт має 0-й номер, а правий - 31-й номер.
- IR_{6} - поле регістра команд, що містить бінарний номер регістра - джерела даного (див. формат команд); довжина поля рівна 5 бітам, що дозволяє позначати та адресувати 32 регістри - від R_{0} до $R_{м}$
- $IR_{u\ 15}$ - також п'ятибітове поле номера ще одного регістра - джерела даного з множини R_{0}-..R_{31}

Вже зауважувалося, що поле безпосереднього операнда (Immediate або Imm) у форматі команди має довжину лише 16 бітів. В той же час, розрядність інформаційного тракту в усіх трьох службових регістрів рівна 32 біти. До того, як записати 16-бітовий код IR_{16} безпосереднього операнда з регістра команд до службового регістра Imm, цей

код треба розширити з врахуванням знака (тобто розряду IR_{16}) до 32-бітового значення а наступним стандартним алгоритмом знакового розширення:

$$IR_{16}\#\#IR_{16}\ \#\#...\#\#\ IR_{16}\#IR_{16\,3\,1}$$

Тут символом ## позначено операцію зчеплення (конкатенацію).

Важливо, що в рамках фази ID виконується декодування (розпізнавання) команди та вибирання усіх можливих варіантів операндів поточної команди, незалежно від її типу, з метою збільшення швидкодії. З іншого боку, можливість одночасного вибирання усіх варіантів операндів потенційно обумовлено відповідною структурою форматів команд, де фіксовано розташування полів-покажчиків на джерела даних та приймачі операндів і результатів. Іншими словами, формати команд процесора втілюють техніку кодування форматів команд, що відома під назвою "fixed-field coding".

4.2.4.3. Фаза виконання та формування ефективної адреси

Мікродії, що виконують в рамках фази виконання та визначення ефективної адреси EX, вже залежать від типу поточної команди. Враховуючи це, вигідно поділити усі команди процесора на наступні три групи:

- команди виконання операцій завантаження операндів з основної пам'яті та збереження результатів у основній пам'яті;
- команди виконання операцій арифметико-логічного пристрою;
- команди виконання операцій умовних та безумовних переходів.

Далі розглядаємо притаманні кожній групі команд мікродії

Команди виконання операції звернення до пам'яті даних

В рамках фази EX операції звернення до пам'яті даних (load/store instructions) готують значення ефективної адреси, тобто вираховують значення бінарного коду, що є адресою комірки пам'яті даних. Виконується наступна мікродія:

$$ALUoutput = A + Imm.$$

Позначене в мікродії додавання безпосереднього операнда (однієї з компонент адреси) з вмістом робочого регістра A (другої компоненти) реалізується в ALU. Отримана сума (вона завжди ціла і додатна, переносом нехтують) записується до ще одного, прозорого для програміста, регістра ALUoutput. Цей регістр має розрядність 32 біти. Він зберігає коди результатів, що з'являються на виході комбінаційної схеми з назвою ALU. Якщо пригадати дію команди LW, тоді призначення та форма запису наведеної мікродії стає зрозумілою.

Команди виконання операції арифметико логічного пристрою типу регістр регістр.

Прикладом команди типу регістр-регістр є команда ADD R1,R2,R3. Виконується вказана командою наступна мікродія

$$ALUoutput = A\ op\ B.$$

Тут узагальнене позначення (op) можна конкретизувати як (add), (sub) тощо, залежно від конкретного виду поточної команди, яка опрацьовується в інформаційному тракті комп'ютера Підкреслимо, що на попередній щодо EX фазі, а саме на фазі ID, до службових регістрів уже завантажено вміст вибраних регістрів регістрового файла з адресами R2 і R3. Результат дії поки що не збережено в регістрі R1, але його тимчасово записано до службового регістра ALUoutput.

Команда виконання операції арифметико-погічного пристрою типу регістр-безпосередній операнд.

Прикладом команди типу регістр-безпосередній операнд є команда ADD R1,R2,#23. Виконується наступна мікродія:

ALUoutput = A op Imm.

Ясно, що замість вмісту регістра R3 в операції бере участь безпосередній операнд, який з урахуванням «знакового» розширення до 32-х бітів зчитується із службового регістра Imm.

Команда умовного переходу.

Прикладом команди умовного переходу є команда BNEZ R5, data. Виконуються наступні мікродії:

ALUoutput = NPC + Imm;
Cond (ition) = (A *op* 0).

Обидві мікродії є сумісними в часі. За допомогою першої мікродії вираховується цільова адреса умовного переходу. При цьому до вже визначеної адреси наступної команди (NPC), тобто такої, яка розташована безпосередньо за командою умовного переходу, додається константа зі службового регістра Imm, (саме він містить значення data). Друга мікродія визначає (істинним чи хибним) є значення умови Condition умовного переходу. Заради цього вміст службового регістра A, що є збіжним із вмістом регістра R5 (у команді-прикладі), порівнюється на основі операції (op) з нулем. Згідно з семантикою команди BNEZ отримується, що Cond = true, коли вміст регістра R5 ненульовий, або Cond = false, якщо регістр R5 містить нуль. Фізично Cond є однобітним регістром у складі вузла Zero?

Звернемо увагу на те, що зараз сформовано лише значення двох необхідних елементів виконання умовного переходу, а саме: однобітний код умови Cond та цільова адреса переходу, яка тимчасово зберігається у службовому вихідному регістрі ALU, тобто в ALUoutput. Безпосереднє виконання умовного переходу, що полягає в природній (за чергою) чи неприродній (коли виконують стрибок до цільової адреси) зміні вмісту програмного лічильника PC, виконується на наступній фазі.

4.2.4.4. Фаза *звернення до пам'яті та завершення умовного переходу*

Звернення до пам'яті.

В рамках фази звернення до пам'яті MEM виконуються наступні мікродії:

LMD = DM [ALUoutput];
DM [ALUoutput] = B.

Звернення до пам'яті застосовується в командах завантаження (наприклад, LW R6, 112(R3)) та в командах збереження (наприклад, SW 112(R3),R6).

Перша мікродія виконується тільки у випадку команди Load (завантаження). Тут за допомогою попередньо розрахованої адреси пам'яті даних, яка тимчасово зберігається у службовому регістрі ALUoutput, здійснюється доступ до пам'яті, що і позначено записом DM [ALUoutput].

Фізично з пам'яті завжди зчитується 32 біти (або навіть цілі пакети з 32-х бітових структурних одиниць). Отриманий з пам'яті даних бінарний код тимчасово завантажу-

ється до ще одного службового регістра LMD (Load Memory Data). І тільки на наступній фазі WB, яка поки що не розглядалася, зчитаний код пересилається з регістра LMD до конкретної комірки (в нашому прикладі - це комірка з адресою R6) регістрового файла.

Друга мікродія реалізується лише при виконанні команди Store (збереження). Тут до комірки пам'яті даних за адресою, що зберігається в службовому регістрі ALUoutput = (R3) + 112, засилається вміст службового регістра B. При цьому (в наведеному прикладі) для команди SW має місце тотожність (B)=(R6).

Умовний перехід.

Виконується наступна мікродія:

if(condition) PC = ALUoutput else PC = NPC.

Здійснюється природна (cond=0) або неприродна (cond =1) заміна вмісту програмного лічильника PC з метою реалізації наступної за умовним переходом команди.

4.2.4.5. *Фаза зворотного запису*

На цій, вже останній фазі виконання команди, у разі потреби результат, що отримано на попередніх фазах, записується до деякої комірки RX регістрового файла. Наприклад, в R1, якщо поточною є команда ADD R1,R2,R3, або в R6 для команди LW R6,#112(R3). Тому вона і має назву зворотного запису (write/back - WB).

Потрібно звернути увагу, що на цій фазі змінюється програмний стан комп'ютера. Після її виконання поновлення попереднього стану є неможливим.

Команди виконання операції арифметико-логічного пристрою типу "регістр-регістр".

Виконується наступна мікродія:

$$Regs[IR_{16}\ J = ALUoutput.$$

Мікродія зберігає результат операції ALU в регістрі призначення (наприклад, в регістрі R1 на прикладі команди ADD). Зрозуміло, що біти 16...20 відповідного формату команди містять бінарний номер регістра призначення.

Команди виконання операції арифметико-логічного пристрою типу "регістр-безпосередній операнд".

Виконується наступна мікродія:

$$Regs[IR_{u\ \ 15}\ ]= ALUoutput.$$

Ця мікродія також зберігає результат операції, наприклад, SUB R5, R4, #1002, в комірці R5 регістрового файла.

Команда завантаження.

Прикладом команди завантаження є команда LW R6,112(R3). Виконується наступна мікродія:

$$Regs[IR_{11}\_\_{}_{15}] = LMD.$$

Наведена мікрокоманда надсилає до комірки R6 регістрового файла вміст комірки пам'яті даних за адресою 112 + (R3), яке на попередніх фазах вже було вибране з пам'яті і тимчасово зберігалося в службовому регістрі LMD. На цьому опис мікродій керування роботою комп'ютера DLX завершено. В табл. 4.1. наведено приклади алгоритмів виконання команд в комп'ютері DLX.

Таблиця 4.1

Приклади алгоритмів виконання команд в комп'ютері DLX

Приклад команди	Алгоритм виконання команди
	Арифметичні та логічні команди
ADD R1,R2,R3	Regs[Rl] = Regs[R2) + Regs[R3]
ADDI R1,R2,#3	RegsfRl] = Regs[R2] + 3
LHI R1, #42	Regs[Rl] =42 ##0[16]
SLLI R1,R2,#5	Regs[Rl] = Regs[R2} « 5
SLT R1, R2, R3	if (Regs[R2] < Regs[R3]) Regs [Rl] = 1 else Regs[Rl] = 0
	Команди збереження й завантаження
LW R1, 30(R2)	Regs [Rl] = $_{32}$ DM [30 + Regs[R2]]
LW R1, 1000(R0)	Regs [Rl] = ,^ DM [1000 + 0]
LB R1, 40(R3)	Regs [Rl] = $_{32}$ (DM [40 + Regs[R3]]$_0$)[24] ## DM [40 + Regs[R3]]
LBU R1, 40(R3)	Regs [Rl] = $_{32}$ 0[24] ## DM [40 + Regs[R3]]
LH R1, 40(R3)	Regs [Rl] = $_{32}$ (DM [40 + Regs[R3]]$_0$)"## DM [40 + Regs[R3]] ## DM [41 + Regs[R3]]
SW 500(R4), R3	DM[500 + Regs[R4]] = $_{32}$ Regs[R3]
SH 502(R2), R3	DM[502 + Regs[R2]] = $_{16}$ Regs[R3]
SB 41(R3), R2	DM[41 + Regs[R3]] = „ Regs[R2]$_{м}$ „
	Команди керування програмою
I name	PC = name;
	((PC + 4) - 2[25]) =< name =< ((PC + 4) + 2[25]);
JAL name	R31 = PC + 4; PC = name;
	((PC + 4) - 2[25]) =< name =< ((PC + 4) + 2[25]);
JALR R2	Regs[R31] = PC + 4; PC = Regs[R2];
BEQZ R4, name	if (Regs[R4] == 0) PC = name;
	((PC + 4) - 2[15]) =< name =< ((PC + 4) + 2[15]);
BNEZ R4, name	if (Regs[R4] != 0) PC = name;
	((PC + 4) - 2[15]) =< name =< ((PC + 4) + 2[15]);

Пояснимо на прикладі синтаксис запису алгоритмів виконання окремих машинних команд комп'ютера *DLX* (див. таблицю 4.1). Розглянемо запис:

г а д і Ш] ^ $_3$ = $_{16}$ (П М [К е $_{8\,5}$ [К 8]] $_0$)[8] ## *BM* [адяв]].

Запис означає наступне. Оновлюють лише 16 молодших (правих) бітів регістра Ш9. До них пересилається двобайтовий бінарний код, в якому молодший правий байт береться з пам'яті даних за адресою, що є збіжною з вмістом регістра 118. Старший лівий байт утворюється восьмиразовим повторюванням нульового (старшого) розряду щойно згаданого правого байта. Парою символів ## позначено операцію конкатенації (зчеплення) двох байтів до двобайтового півслова. Наведеного таблицею 4.1 достатньо, аби синтезувати неподані алгоритми виконання інших команд комп'ютера ПЬХ.

4.2.5. *Конвеєрна структура процесора комп'ютера з простою системою команд*

4.2.5.1. Конвеєрний процесор

Конвеєрну структуру процесора комп'ютера з простою системою команд розглянемо на прикладі вище описаного комп'ютера *BIX.* Вище виконання типової команди в комп'ютері ОЬХ було розділено на наступні фази:

1. IF"- вибірка команди (за адресою, заданою лічильником команд, із пам'яті зчитується команда).

2. ID - декодування команди/вибірка операндів з регістрів.

3. EX - виконання операції та обчислення ефективної адреси пам'яті.

4. MEM - звернення до пам'яті.

5. WB - запам'ятовування результату.

На рис. 4.6 представлена схема процесора, що виконує вказані вище фази виконання команд без перекриття. Щоб конвеєризувати цю схему, можна просто розбити виконання команд на вказані вище фази, відвівши для виконання кожної фази один такт синхронізації, і починати в кожному такті виконання нової команди. Природно, для зберігання проміжних результатів кожної фази необхідно використовувати конвеєрні регістри. На рис. 4.9 показана схема процесора з конвеєрними регістрами, які забезпечують передачу даних і керуючих кодів з одного ярусу конвеєра на наступний. Хоча загальний час виконання однієї команди в такому конвеєрі складатиме п'ять тактів, у кожному такті апаратура виконуватиме в суміщеному режимі п'ять різних команд.

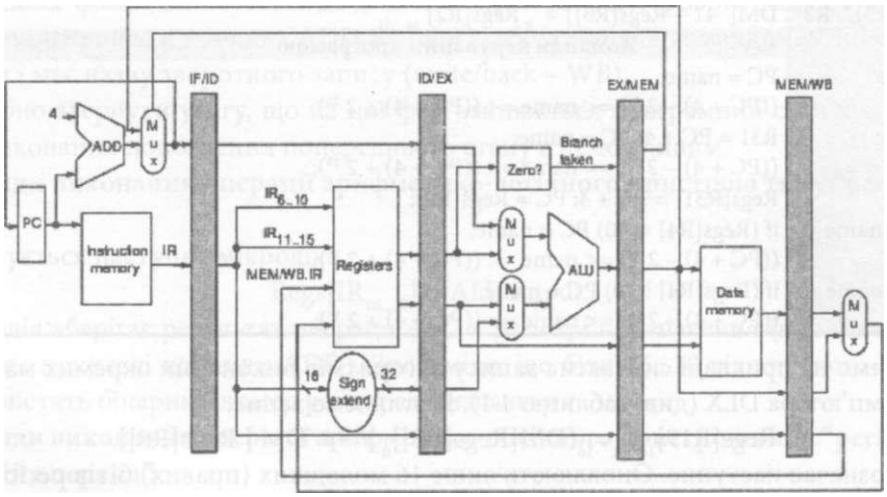


Рис. 4.9. Конвеєрна структура процесора комп'ютера DLX (з кеш пам'яттю даних та команд)

Конвеєрний процесор комп'ютера DLX структуровано наступними ярусами конвеєра: IF, ID, EX, MEM, WB. Апаратура кожного ярусу реалізує притаманні їй мікрооперації. Наприклад, на першому ярусі виконується вибирання команди з пам'яті команд IM за вмістом програмного лічильника PC, приріст (інкремент) на +4 (з врахуванням логічного байтового адресування пам'яті команд) поточної адреси за допомогою комбінаційного суматора ADD та занесення значення наступної адреси до поля NPC (Next PC), інтегрованого до конвеєрного регістра IF/ID. Мультиплексор Mux, що керується відповідним однобітним полем конвеєрного регістра EX/MEM, визначає джерело запису до NPC - або наступна за чергою адреса, або цільова адреса умовного чи безумовного переходу. Важливо, що обов'язок змінювати природне адресування послідовності вибирання команд з пам'яті команд покладено на вміст команди, яка пройшла фазу конвеєра MEM.

Конвеєрні регістри виконують функцію збереження вмісту інтегрованих до них регістра команди IR, робочих регістрів A, B тощо. Конвеєрні регістри розташовано на межах ярусів. Вони мають назви, відповідні граничним ярусам, наприклад IF/ID. Тоді поле A конвеєрного регістра позначається як EX/MEM.A.

До апаратури другого ярусу ID належать регістровий файл Regs, який містить множину програмно-доступних регістрів, та поширювач знаку Sign extend, що конвертує 16-бітові безпосередні знакові константи у 32-бітові стандартні операнди формату з фіксованою комою.

Апаратура третього ярусу містить комбінаційний ALU із мультиплексорами на кожному вході і схему (Zero?) визначення істинності чи хибності умови команди умовного переходу.

Призначення інших вузлів є зрозумілим з рисунка. Можна на додачу зауважити, що регістровий файл має два порти на читання і один на запис. Ця особливість є прямим наслідком запроваджених в комп'ютері DLX системи команд і конвеєрного принципу роботи. Регістровий файл Regs працює у кожній команді на двох ярусах конвеєра - ID (два читання) та WB (один запис). Означені фази двох різних команд можуть збігатися у часі. Аби запобігти колізії, потрібна реалізація одночасного читання та подвійного запису до цього файла. Крім того, необхідно запобігати запису даних до того ж самого регістра.

Роботу конвеєра можна умовно представити у вигляді часової діаграми суміщеного в часі виконання фаз команди, як це було показано раніше, або у вигляді зміщених в часі схем тракту даних комп'ютера DLX (рис. 4.10), де кожна схема зображає ту ж саму фазу команди. Тут CCi - тактові інтервали, IM - пам'ять команд, REG - регістровий файл, ALU - арифметико логічний пристрій, DM - пам'ять даних.

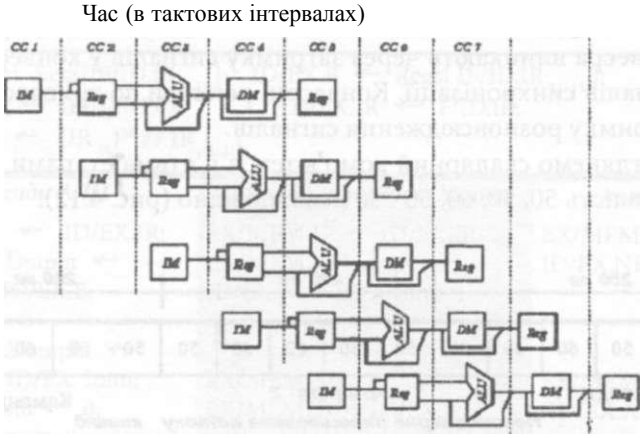


Рис. 4.10. Робота конвеєра у вигляді зміщених в часі схем тракту даних

Цей рисунок добре відображає поєднання в часі виконання різних етапів команд. Тут перша опрацьована команда знаходиться вгорі, остання - знизу, CC - тактовий інтервал. Проте частіше для представлення роботи конвеєра використовуються часові діаграми (рис. 4.11), на яких зазвичай зображаються виконувані команди, номери тактів і етапи виконання команд.

Номер команди	Номер такту								
	1	2	3	4	5	6	7	8	9
J	IF	ID	EX	MEM	WB				

i+1		ІБ	Ш	ЕХ	МЕМ				
i+2			ІБ	Ю	ЕХ	МЕМ			
i+3				ІБ	ГО	ЕХ	МЕМ	ШВ	
i+4					ЦР	Ш	ЕХ	МЕМ	ШВ

Рис. 4.11. Часова діаграма роботи конвеєра

З приведеної часової діаграми стає зрозуміло, для чого потрібні дві незалежні пам'яті - команд та даних. Це дозволить на тому ж самому тактовому інтервалі без колізії поєднувати виконання фаз ІБ та МЕМ двох команд. При цьому, безперечно, зростають вимоги до швидкодії запам'ятовуючих пристроїв. Такі пристрої повинні відпрацьовувати звертання за один цикл (в нашому випадку за один тактовий імпульс), що обумовлює раціональність застосування відокремлених кеш пам'ятей команд та даних.

Конвеєризація збільшує пропускну спроможність процесора (кількість виконаних команд за одиницю часу), але вона не скорочує час виконання окремої команди. Насправді, вона навіть дещо збільшує час виконання кожної команди із-за затримок в конвеєрних регістрах. Проте збільшення пропускної спроможності означає, що програма виконуватиметься швидше порівняно зі скалярною (не конвеєрною) схемою.

Той факт, що час виконання кожної команди в конвеєрі не зменшується, накладає деякі обмеження на практичну довжину конвеєра. Окрім обмежень, пов'язаних із затримкою конвеєра, є також обмеження, що виникають в результаті незбалансованості затримки на кожному його ярусі та із-за накладних витрат на конвеєризацію. Частота синхронізації не може бути вищою, а, отже, такт синхронізації не може бути меншим, ніж час, необхідний для роботи найбільш повільного ярусу конвеєра. Накладні витрати на організацію конвеєра виникають через затримку сигналів у конвеєрних регістрах та із-за перекосів сигналів синхронізації. Конвеєрні регістри до тривалості такту додають час установки і затримку розповсюдження сигналів.

Як приклад розглянемо скалярний комп'ютер із п'ятьма етапами виконання операцій, які мають тривалість 50, 50, 60, 50 і 50 не відповідно (рис. 4.12).

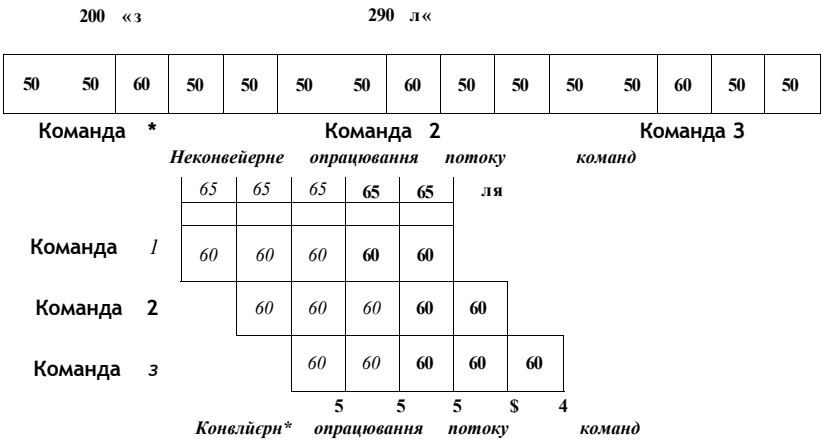


Рис. 4.12. Конвеєр потоку команд

Хай накладні витрати на організацію конвеєрної обробки складають 5 не. Тоді середній час виконання команди в скалярному процесорі буде рівний 260 не. Якщо ж використовується конвеєрна організація, тривалість такту буде рівна тривалості найповільнішої фази обробки плюс накладні витрати, тобто 65 не. Цей час відповідає середньому часу виконання команди в конвеєрі. Таким чином, маємо чотирикратне прискорення, одержане в результаті конвеєризації.

Конвеєризація ефективна тільки тоді, коли завантаження конвеєра близьке до повного, а швидкість подачі нових команд і операндів відповідає максимальній продуктивності конвеєра. Якщо відбудеться затримка, то паралельно виконуватиметься менше операцій, і сумарна продуктивність знизиться. Такі затримки можуть відбуватися в результаті виникнення конфліктних ситуацій. Далі будуть розглянуті різні типи конфліктів, що виникають при виконанні команд в конвеєрі, і способи їх вирішення. Але спочатку розглянемо мікродії ярусів вже для варіанта конвеєрного процесора.

4.2.5.2. Мікродії ярусів конвеєрного процесора

Відразу зазначимо, що всі мікродії одного ярусу конвеєра мають бути сумісними в часі та виконуються паралельно в рамках однієї фази (як правило, за один тактовий інтервал). Мікродії, що реалізуються в кожному ярусі конвеєра комп'ютера ОЬХ, зведено в табл. 4.2.

Таблиця 4.2

Ярус	*Мікродії в конвеєрі DLX*		
IF	IF/ID.IR *- Mem[PC]; IF/ID.NPC, PC *- (if EX/MEM.cond (EX/MEM.NPC} else (PC+4));		
ID	ID/EX.A *- Regs[IF/ID.IR J; ID/EX.B «- Regs[IF/ID.IR]; ID/EX.NPC *- IF/ID.NPC; ID/EX.IR *- IF/ID.IR; ID/EX.Imm *- (IR J ' ' ## IR		
EX	*команди АЛП*	*команди читання 1 запису*	*команди переходу*
	EX/MEM.IR *- ID/EX.IR; EX.MEM.ALUutput *- ID/EX.A op ID/EX.B; or EX/MEM.ALUoutput *- ID/EX. A op ID/EX. Imm; EX/MEM. cond *- 0;	EX/MEM.IR *- ID/EX.IR; EX/MEM.ALUoutput <- ID/EX.A + ID/EX.Imm; EX/MEM.cond 0; EX/MEM. B *- ID/EX.B;	EX/MEM.АЕиоШрШ *- ГО/ЕХ.ИРС + ГО/ЕХ.Ітт; EX/MEM.соп<1 *~ (ГО/EX.A op 0);
MEM	MEM/WB.IR *~ EX/MEM.IR; MEM/WB.ALUoutput *- EX/MEM.ALUoutput;	MEM/WB.IR EX/MEM.IR; MEM/WB.LMD Mem [EX/MEM. ALUoutput]; or Mem [EX/MEM.ALUoutput EX/MEM.B;	*Дій немає*
WB	Regs[MEM/WB.IR J *- MEM/WB.ALUoutput; or Regs[MEM/WB.IR J *- MEM/WB.ALUoutput;	Regs[MEM/WB.IR „ *^ MEM/WB.LMD;	*Дій немає*

Мікродії ярусу IF.

Перша мікродія вибирає нову команду з пам'яті команд за адресою, що зберігається в PC, і записує її до поля IR (Instruction Register) конвеєрного регістра IF/ID. В той самий час друга мікродія змінює вміст поля NPC конвеєрного регістра і програмний лічильник за алгоритмом: якщо бітове поле cond (condition - умова) попередньої команди, яка пройшла фазу EX, є одиницею (true), тоді порушується природна черговість і вміст IF/ID.NPC та PC отримує значення поля EX/MEM.NPC конвеєрного регістра EX/MEM; інакше записується наступна адреса (PC+4) з врахуванням байтової логічної структури адреси пам'яті.

Мікродії ярусу ID.

Усі чотири мікродії є сумісними і виконуються в часі паралельно. Перша мікродія вибирає перший операнд з програмно керованого регістра регістрового файла до службового регістра A, що є інтегрованим до конвеєрного регістра ID/EX. При цьому адреса програмно керованого регістра визначається вмістом розрядів 8..10 поля IR конвеєрного регістра IF/1D. Тут вибирається операнд. Такі ж за призначенням дії виконує друга мікрооперація, але з іншим джерелом і приймачем. Третя і четверта мікродії зберігають контекст команди, що знаходиться на поточній сходинці. Це необхідно для її коректного просування конвеєром. Четверта мікродія вибирає (та знаково розширює з 16 до 32-х бітів) до службового регістра Imm (immediate - безпосередній) операнд, який містився у розрядах 16...31 поля IR конвеєрного регістра. Поточну фазу ID можна розширити у назві додатковим означенням Operand Fetch (вибирання операндів).

Мікродії ярусу EX (команди арифметико-логічного пристрою).

Важливо відзначити, що на фазі EX вперше від початку виконання команди має бути визначеним її тип. Перша мікродія зберігає вміст регістра команди. Четверта мікродія забороняє командам ALU впливати на послідовність вибирання команд з пам'яті. Друга і третя мікродії утворюють альтернативу (або). Кожна з них визначає пару операндів для операції or і при цьому записує результат or до службового (програмно-некерованого) вихідного регістра ALU під назвою ALUoutput.

Мікродії ярусу EX (команди load/store).

Перша мікродія зберігає контекст регістра команди, друга вираховує виконавчу (ефективну) адресу пам'яті даних на основі бази (Immediate - безпосередній операнд), третя зберігає вміст службового, програмно-некерованого регістра B, четверта забороняє поточній команді змінювати природний порядок адресування команд.

Мікродії ярусу EX (команда branch).

Перша мікродія вираховує цільову адресу можливого переходу та зберігає її у робочому (некерованому програмістом) вихідному регістрі ALUoutput, а конкретно - у полі ALUoutput конвеєрного регістра EX/MEM. Друга мікродія вираховує істинне або хибне значення логічної умови, що визначається порівнянням в деякому, тобто or розумінні, службового регістра A, визначеного за вмістом на фазі ID, з нулем (дорівнює нулю, не дорівнює нулю, тощо). Логічне значення умови записується до поля cond конвеєрного регістра EX/MEM з метою дозволу зміни природного порядку вибирання команд програми, коли cond=1. Контексти не зберігаються, що свідчить про неформальне завершення опрацювання цієї команди в конвеєрі.

Мікродії ярусу MEM (команди арифметико логічного пристрою).

Активних мікродій обробки інформації немає, що свідчить про транзитний характер опрацювання команди на цій сходинці. Обидві мікродії лише зберігають для подальшого користування вміст регістра команд і вихідного регістра ALU.

Мікродії ярусу MEM (команди load/store).

Перша мікродія виконує транзитне пересилання вмісту коду операції з відповідного поля вхідного конвеєрного регістра до відповідного поля вихідного конвеєрного регістра ярусу. Це свідчить про те, що виконання команди (лише - завантаження) має продовжуватися в наступному ярусі конвеєра. При завантаженні виконується друга мікродія, а при збереженні - третя. Виконавча (ефективна) адреса пам'яті даних визначається вмістом службового вихідного регістра ALU. При завантаженні вміст комірки пам'яті даних зберігається в проміжному регістрі LMD (Load Memory Data), а при збереженні вміст службового регістра B записується до комірки пам'яті даних

Важливо, що дана мікропрограма ігнорує існування відомого парадоксу пам'яті, що коректно тільки за умови використання кеш пам'яті даних та системи переривань у випадку «невлучення до кеш» («покарання» за невлучення - це певна кількістю додаткових тактових інтервалів, аби погодити швидкодію процесора і пам'яті даних за рахунок пригальмовування операцій в скалярному процесорі).

Мікродії ярусу WB (команди арифметико логічного пристрою).

Завжди виконується лише одна мікрооперація з двох зазначених. В кожному випадку результат обробки операндів в ALU з поля конвеєрного регістра MEM/WB.ALUoutput записується до регістра регістрового файла процесора. Використання двох мікрокоманд замість однієї пояснюється тим, що у форматі команд load DLX повного дотримання правила «фіксоване розташування полів» немає. За рахунок цього адреса призначення у форматі команди рухається: може визначатися розрядами 16...20 або розрядами 11...15 команди. Так чи інакше, але вказана «рухомість» адреси поля призначення ускладнює апаратний пристрій керування і може зменшити його швидкодію

Мікродії ярусу WB (команда load).

Зазначимо, що команда store (збереження) на цьому ярусі виконання не потребує мікродій. Тут завершується виконання лише команди завантаження операнда з комірки пам'яті даних до регістра регістрового файла процесора

Операнд зберігається у полі LMD вхідного конвеєрного регістра MEM/WB, а адреса комірки (регістра) регістрового файла міститься у полі MEM/WB.IR 11**...I5.** Важливо, що регістровий файл повинен реалізувати два порти, а саме, два порти на читання та один порт на запис. При цьому, якщо дві адреси на читання постачає конвеєрний регістр (IF/ID), то адресу на запис і дані постачає щойно розглянутий конвеєрний регістр (MEM/WB).

4.3. Суперконвеєрні процесори

Можлива така організація виконання деякої послідовності команд в процесорі, коли всі однойменні фази виконання цих команди послідовно, тобто спочатку проводиться вибірка всіх команд, далі їх декодування і т. д., як це показано на рис. 4.13. для послідовності із двох команд

IF1	IF2	ID1	ID2	EX1	EX2	ME1	ME2	WB1	WB2	IF1	IF2	ID1	ID2	EX1	EX2	ME1	ME2	WB1	WB2

Рис. 4.13. Послідовне виконання однойменних фаз двох команд

Такий підхід не прискорює роботу процесора, але при конвеєрному опрацюванні команд може виявитися доцільним, оскільки в ярусах конвеєра (рис. 4.14) знаходяться результати виконання декількох фаз різних команд, що при наявності конфліктів дозволяє ефективніше їх вирішувати, аніж у звичайному конвеєрі команд. Процесор з конвеєром команд, в якому послідовно виконуються декілька фаз над різними командами, називається суперконвеєрним.

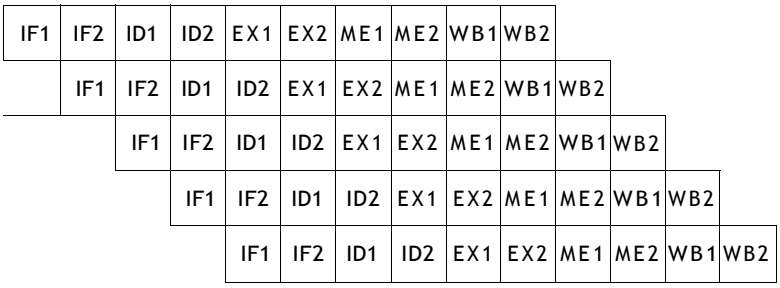


Рис. 4.14. Діаграма виконання команди в суперконвеєрному процесорі при послідовному виконанні фаз двох команд

Як видно з приведеної на рис. 4.14 діаграми, при послідовному виконанні фаз двох команд в одному такті роботи конвеєра кожна з фаз повинна виконуватись двічі. Коли послідовно виконується k фаз команд, то в кожному такті кожна з фаз має виконуватися k раз. Це говорить про те, що внутрішня частота роботи ярусів конвеєра суперконвеєрного процесора є в k разів вищою їх зовнішньої частоти, з якою відбувається обмін інформацією між ярусами.

Потрібно відзначити, що для організації суперконвеєрного опрацювання команд необхідне деяке додаткове обладнання порівняно з конвеєрним. Це, зокрема, регістри для зберігання проміжних результатів послідовно виконуваних фаз різних команд.

4.4. *Суперскалярні процесори*

Вище була розглянута конвеєрна структура процесора, коли засоби виконання ярусів потокового графа алгоритму розділяються конвеєрними регістрами. Щоб підвищити продуктивність конвеєрного процесора потрібно далі спрощувати операції його ярусів та поглиблювати глибину конвеєра. Це і робиться в сучасних процесорах, в яких глибина конвеєра досягає двадцяти і більше ярусів. Наприклад, процесор комп'ютера UltraSPARC III має 10 ярусів конвеєра, а процесор комп'ютера Pentium IV - 20 ярусів конвеєра. Однак процес спрощення операцій ярусів конвеєра має межу, коли операції не піддаються поділу. Наприклад, фаза вибірки команди з пам'яті не може бути поділеною на простіші фази. Тоді для підвищення продуктивності процесора необхідно використовувати паралельне включення декількох конвеєрів команд. Такі процесори з декількома конвеєрами команд дозволяють одночасно виконувати кілька скалярних команд, а тому дістали назву суперскалярних.

Першу суперскалярну архітектуру розробив Джон Коук (John Cocke, IBM, 1987 рік), що отримала назву America. Він і запропонував термін "суперскаляр". Вже потім модифікований варіант архітектури America під назвою POWER-1 (Performance Optimization With Enhanced RISC) впровадили до серійних систем RISC System/6000 фірми IBM.

Нарешті, підмножину архітектури POWER-1 реалізовано в процесорах Power PC, які є основою комп'ютерів Apple Macintosh. Іншими прикладами суперскалярних процесорів є процесори систем UltraSparc фірми Sun та Alpha фірми DEC.

Структура суперскалярного процесора та його зв'язки з кеш пам'яттю даних і команд, показані на рис. 4.15.

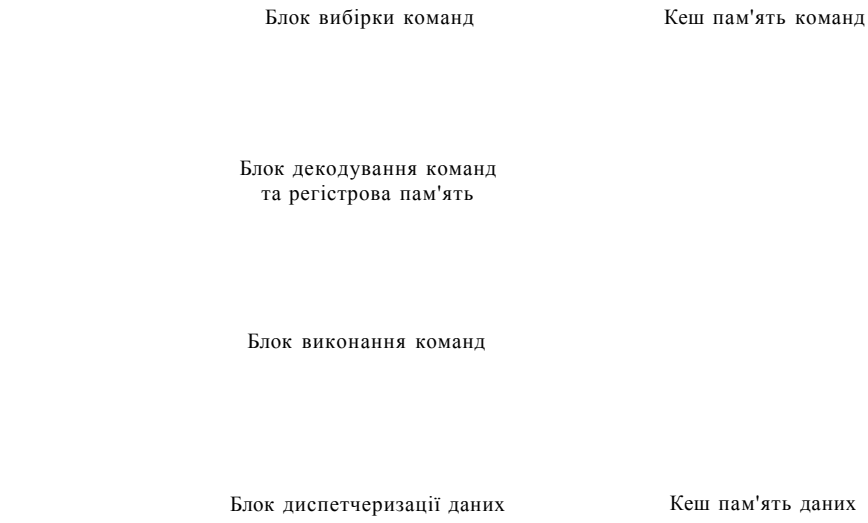


Рис. 4.15. Структура суперскалярного процесора

Тут одночасно вибирається та декодується декілька команд, а блок виконання команд включає кілька функціональних блоків. Для забезпечення одночасного читання та запису кількох операндів кеш пам'ять будується за модульним принципом.

Зрозуміло, що підвищення продуктивності такого процесора досягається шляхом його конвеєризації. Діаграма виконання команд в суперскалярному процесорі, який має два конвеєри команд, показана на рис. 4.16а.

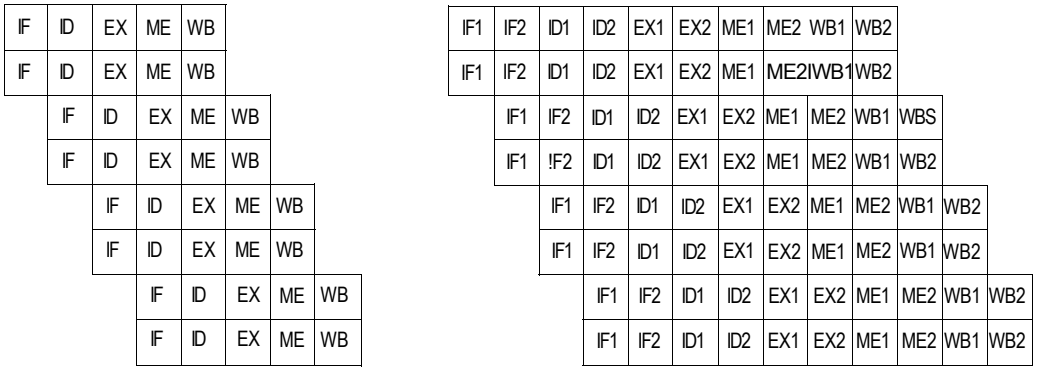


Рис. 4.16. Діаграма виконання команд в суперскалярному процесорі з двома конвеєрами команд, коли в одному такті виконується одна (а) та дві (Ь) фази команди

Можливе суміщення суперскалярного та суперконвеєрного опрацювання команд, як це показано на рис. 4.16 Ь.

4.5. *Процесор векторного комп'ютера*

Вище були розглянуті скалярні та суперскалярні процесори, в яких операції виконуються над скалярними даними. Однак існує значна кількість завдань, коли опрацюванню за одними процедурами підлягають великі масиви (вектори) даних. У цьому випадку виглядає доцільним розгляд можливості модифікації комп'ютера під виконання цього класу завдань. До цих пір така модифікація здійснювалась в потужних комп'ютерах, але на даний час вона почала поширюватись на всі типи комп'ютерів. Відповідно комп'ютери, орієнтовані на опрацювання векторів даних, дістали назву векторних.

Різницю між виконанням скалярної та векторної операції наглядно відображає рис. 4.17, з якого видно, що скалярна операція передбачає виконання додавання над двома даними, тоді як векторна - над двома векторами даних.

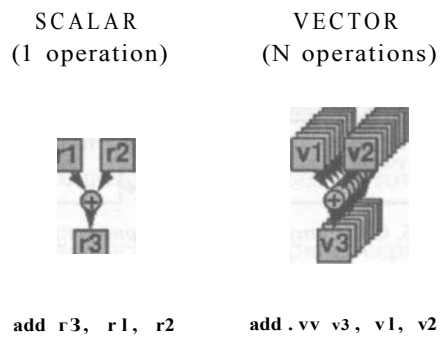


Рис. 4.17. Виконання скалярної та векторної операцій додавання

Аби зрозуміти стиль програмування векторних комп'ютерів, наведемо приклад програми із скалярними і векторними кодами. Запишемо програму обчислення виразу У= а * Х + У, де У, Х - вектори, а а - скаляр. Нехай вектори мають довжину по 64 елементи. Векторна програма має вигляд:

```
LD      F0, a        ; load scalar a
LV      VI, Rx       ; load vector X
MULTS   V2, F0, VI   ; vector-scalar mult.
LV      V3,Ry        ; load vector Y
ADDV    V4, V2, V3   ;add
SV      Ry, V4       ; store vector
```

Відповідна скалярна програма має вигляд:

```
Loop:   LD      F0,a
        ADDI    R4,Rx,#512   ; last address to load
        LD      F2,0(Rx)     ; load X(l)
        MULTD   F2,F0,F2     ; a*X(l)
        LD      F4, 0(Ry)    ; load Y(l)
        ADDD    F4.F2.F4     ; a*X(l)+Y(l)
        SD      F4,0(Ry)     ; store into Y(l)
        ADDI    Rx,Rx,#8     ; increment index
        ADDI    Ry,Ry,#8     ; increment index
        SUB     R20,R4,Rx    ; compute bound
        BNZ     R20, loop    ; check if done
```

У скалярній програмі курсивом позначено залежності, яких немає у векторному варіанті програми. Обидва варіанти програми можна порівняти за наступними кількісними характеристиками:

1. За кількістю операцій: 578(2+9*64) проти 321 (1 +5*64); кількість операцій у векторній програмі зменшено в 1,8 разу.

2. За кількістю команд: 578(2+9*64) проти 6-ти команд у векторній програмі; перевага в 96 разів.

В таблиці 4.3 наведені характеристики кількох промислових векторних комп'ютерів, з якої видно доцільність їх створення з огляду на досягнуту продуктивність.

Таблиця 4.3

Тип машини	Рік випуску	Частота, MHz	Кількість регістрів	Кількість елементів	Кількість пристроїв float point	Кількість пристроїв load/store	Продуктивність (MFLOFS)
Cray-1	1976	80	8	64	6	1	160
Cray XMP	1983	120	8	64	8	2L, IS	940
Cray YMP	1988	166	8	64	8	2L, IS	2667
Cray C-90	1991	240	8	128	8	4	15238(16)
Cray T-90	1996	455	8	128	8	4	57600(32)
Conv. C-1	1984	10	8	128	4	1	20(1)
Conv. C-4	1994	133	16	128	3	1	3240(4)
Fuj. VP200	1982	133	8-256	32-1024	3	2	533(1)
Fuj. VP300	1996	100	8-256	32-1024	3	2	N/A
NEC SX/2	1984	160	8 + 8K	256 + var	16	8	1300(1)
NEC SX/3	1995	400	8 + 8K	256 + var	16	8	25600(4)

Таким чином, процесори векторних комп'ютерів виконують команди над векторами даних. Структура цих процесорів за складом та зв'язками повторює вже розглянуті вище структури процесорів, тобто це можуть бути процесори векторних комп'ютерів із

складною та простою системою команд, конвеєрні та суперконвеєрні, а також процесори супервекторних комп'ютерів, коли в процесорі є декілька конвеєрів команд. Основна їх відмінність - забезпечення одночасного виконання однієї команди над вектором даних. Це, зокрема, дозволяє будувати їх блоки виконання команд за конвеєрним принципом і при цьому позбутися конфліктів, які суттєво гальмують роботу конвеєра чи ускладнюють його структуру.

Для вияснення базових принципів побудови процесорів векторних комп'ютерів розглянемо структуру та систему команд процесора векторного варіанта комп'ютера БЬХ, а саме комп'ютера ЭЬХУ. До складу процесора, структура якого приведена на рис. 4.18, входять пристрій векторного читання запису, регістрові файли з векторними та скалярними регістрами, а також операційний пристрій з набором конвеєрних операційних пристроїв додавання, множення та ділення з рухомою комою та виконання арифметичних і логічних операцій над цілими числами.

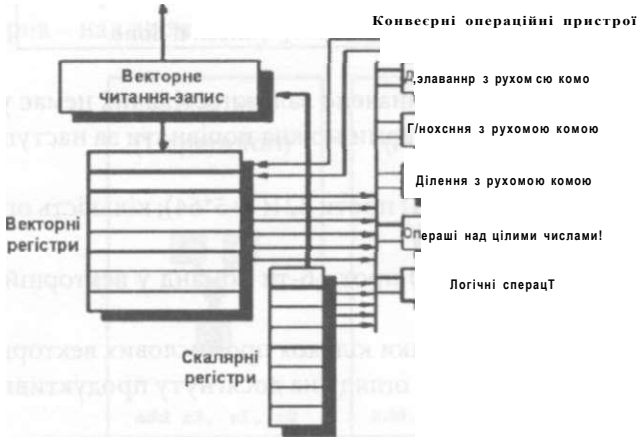


Рис. 4.18. Структура процесора комп'ютера DLXV

Цей комп'ютер має векторні команди, які наведено в табл. 4.4.

Таблиця 4.4

Команда	Операнди	Операція	Коментар
ADDV	V1.V2.V3	V1=V2+V3	VECTOR+VECTOR
ADDSV	V1.F0.V2	V1=F0+V2	SCALAR+VECTOR
MULTV	V1.V2.V3	Vl=V2xV3	VECTORxVECTOR
MULSV	VI, F0.V2	V1=F0XV2	SCALAR x VECTOR
LV	VI, R1	Vl=M[Rl..Rl+63]	LOAD, STRIDE=1
LVWS	V1.R1.R2	V1=[R1..R1+63*R2]	LOAD, STRIDE=R2
LVI	V1.R1.V2	V1=[R1+V2i,i=0..63	indirect ("gather")
CeqV	VM.V1.V2	VMASKi=(V1I=V2I)	comp. Set mask
MOV	VLR.Rl	Vec. Len. Reg = R1	set vector length
MOV	VM, R1	Vec. Mask=Rl	set vector mask

Приведений процесор є процесором комп'ютера з простою системою команд. До цього типу належать усі векторні суперкомп'ютери: Cray, Convex, Fujitsu, Hitachi, NEC.

Хоча потрібно зауважити, що існують і векторні комп'ютери з архітектурою «иам'ять-пам'ять», коли всі векторні операції є операціями типу пам'ять-пам'ять наприклад, CDC-6600.

Подібно до приведеного на рис. 4.18, процесори векторних комп'ютерів містять наступні основні компоненти

• Векторні регістри; це регістрий файл фіксованої ємності що вміщує вектор даних. Цей файл має як мінімум 2 порти на читання і один порт на запис та зазвичай включає 8-32 векторних регістри, кожний з яких є 64-128-розрядним

• Конвеєрні операційні пристрої. Зазвичай застосовують 4-8 операційних пристроїв, а саме: додавання, множення і ділення з фіксованою та рухомою комою, зсуву тощо

• Векторний вузол читання-запису, також конвеєрний, який опрацьовує вектори даних. Водночас застосовують декілька таких вузлів

• Скалярні регістри, які містять один скаляр з рухомою комою або адресу

• Багатошинні магістралі або комутаційні мережі, які з'єднують між собою всі зазначені компоненти, аби прискорити роботу процесора в цілому

Перші векторні комп'ютери STAR-100 фірми CDC та ASC фірми TI були створені в 1972 році. Це були векторні комп'ютери з архітектурою типу пам'ять-пам'ять

Значний внесок в теорію побудови векторних комп'ютерів зробив видатний американський конструктор векторних суперкомп'ютерів Сеймур Крей. На рис. 4.19 зображено його перший векторний суперкомп'ютер Cray-1 та векторні суперкомп'ютери Cray-2 і Cray-YMP.

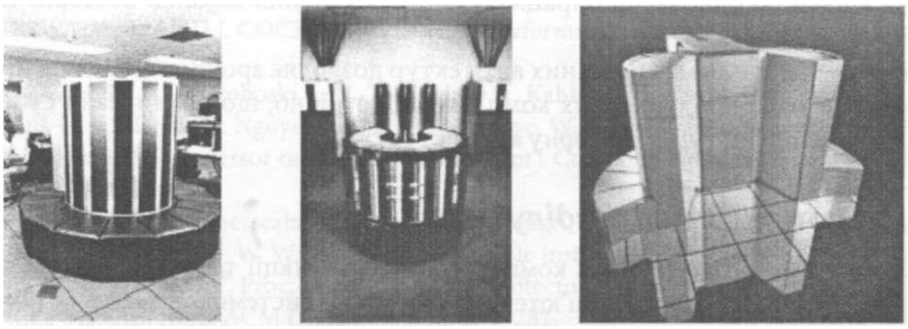


Рис. 4.19. Векторні суперкомп'ютери Cray-1, Cray-2 і Cray-YMP

Комп'ютер CRAY-1 був створений в 1976 році і мав архітектуру типу «регістр-регістр», тому він був найшвидшим серед векторних та скалярних комп'ютерів свого часу. В 1981 році на ринку з'явився значно потужніший векторний комп'ютер CYBER-205 фірми CDC з тією ж базовою архітектурою, що і STAR-100, але більшою кількістю векторних функціональних блоків. Це були векторні комп'ютери з архітектурою типу "пам'ять-пам'ять". В 1983 році фірма Cray Research поставила на ринок векторний суперкомп'ютер CRAY X-MP, а через кілька років - CRAY-2, який мав вищу тактову частоту та іще більший рівень конвеєризації. В 1988 році фірма Cray Research створила значно швидший, ніж X-MP суперкомп'ютер CRAY Y-MP, котрий мав 8 конвеєрних процесорів, кожний з яких працював з тактом 6 не.

Одночасно потужні векторні суперкомп'ютери почали створюватись і в інших державах. Зокрема, в середині 80-х років в Японії були створені суперкомп'ютери Fujitsu

VP100 і VP200, за ними Hitachi S810 і NEC SX/2, які за технічними характеристиками не поступалися комп'ютерам фірми Cray Research

Протягом наступних 20 років векторні комп'ютери мали швидкий розвиток і з екзотичних перетворились в широковживаний клас потужних комп'ютерів.

4.6. *Класифікація архітектури комп'ютера за рівнем суміщення опрацювання команд та даних*

Виходячи з вищенаведеного розгляду різних принципів побудови процесорів, можна зробити наступну класифікацію архітектури комп'ютера за рівнем суміщення в них опрацювання команд та даних

- за відсутністю та наявністю конвеєра команд: комп'ютери без конвеєра команд та комп'ютери з конвеєром команд
- за відсутністю та наявністю конвеєра даних: комп'ютери без конвеєра даних та комп'ютери з конвеєром даних
- за кількістю послідовно виконуваних фаз команд в конвеєрі: конвеєрні та суперконвеєрні
- за кількістю одночасно опрацьовуваних даних за однією командою: скалярні та векторні
- за кількістю одночасно опрацьовуваних скалярних команд: скалярні та суперскалярні
- за кількістю одночасно опрацьовуваних векторних команд: векторні та супервекторні

Проведений вище аналіз названих архітектур дозволяє зробити висновок про те, що для побудови високопродуктивних комп'ютерів потрібно, щоб вони мали суперконвеєрну суперскалярну та супервекторну архітектуру

4.7. *Короткий зміст розділу*

Розглянуто місце процесора в комп'ютері, його функції та склад. Приведена одношинна структура процесора комп'ютера із складною системою команд та розглянуто виконання на ній основних операцій процесора: вибірки слова із пам'яті, запам'ятовування слова в пам'яті, обміну між регістрами, виконання арифметичних і логічних операцій. Проведено порівняння одношинної з багатошинною структурою процесора

Виділено особливості побудови процесора комп'ютера із складною системою команд та сформовано вимоги до процесора комп'ютера з простою системою команд. Описані базові принципи побудови процесора комп'ютера з простою системою команд, а також взаємодія процесора з основною пам'яттю в комп'ютері з простою системою команд

Розглянуто фази виконання команд в процесорі комп'ютера з простою системою команд: вибирання та декодування команди, виконання та формування ефективної адреси, звернення до пам'яті/завершення умовного переходу, зворотного запису. Описана конвеєрна структура процесора з простою системою команд на прикладі процесора комп'ютера DLX. Розглянуто принципи побудови суперконвеєрним та суперскалярних процесорів, а також процесорів векторних комп'ютерів. Виходячи з вищенаведеного розгляду різних принципів побудови процесорів, зроблено класифікацію архітектури комп'ютера за рівнем суміщення в ньому опрацювання команд та даних.

4.8. Література для подальшого читання

Питання побудови процесорів комп'ютерів з складною системою команд детально розглянуті в багатьох підручниках та наукових працях, зокрема [1-4, 7, 8]. Вимоги до процесора комп'ютера з простою системою команд та базові принципи його побудови детально описані в роботах [5, 6]. Там же, а також в роботах [7-8] запропоновано принципи побудови процесора комп'ютера DLX і процесора векторних комп'ютерів, а в роботах [9-10] розглянуті принципи побудови суперконвеєрним та суперскалярних процесорів.

4.9. Література до розділу 4

1. Каган Б.М. Электронные вычислительные машины и системы. - М.: Энергия, 1979. - 528 с.

2. Каган Б. М., Каневский М. М. Цифровые вычислительные машины и системы. - М.: Энергия, 1974. - 680 с.

3. Tanenbaum, A. *Structured Computer Organization,* 4th ed. Upper Saddle River, NJ: Prentice Hall, 1999.

4. Stallings, W. *Computer Organization and Architecture,* 5th ed., New York, NY: Macmillan Publishing Company, 2000.

5. D. Patterson, J. Hennessy. Computer Architecture. A Quantitative Approach. Morgan Kaufmann Publishers, Inc. 1996.

6. Patterson, D. A., 8c Hennessy, J. L. *Computer Organization and Design, The Hardware/Software Interface,* 2nd ed., San Mateo, CA: Morgan Kaufmann, 1997.

7. AGERWALA, T. AND J. COCKE [1987]. "High performance reduced instruction set processors", IBM Tech. Rep. (March).

8. Bakoglu, H. B., G. F. Grohoski, L E. Thatcher, J. A. Kahle, C R. Moore, D. P. Tuttle, W. E. Maule, W. R. Hardell, D. A. Hicks, M. Nguyen phu, R. K. Montoye, W. T. Glover, and S. Dhawan [1989]. «IBM second-generation RISC processor organization,» Proc. Int'l Conf. on Computer Design, IEEE (October), Rye, N.Y., 138-142.

9. Johnson, M. [1990]. Superscalar Microprocessor Design, Prentice Hall, Englewood Cliffs, N.J.

10. JOUPPI, N. P. AND D. W. WALL [1989]. "Available instruction-level parallelism for superscalar and superpipelined processors", Proc. Third Conf. on Architectural Support for Programming Languages and Operating Systems, IEEE/ACM (April), Boston, 272-282.

4.10. Питання до розділу 4

1. Місце процесора в комп'ютері та його функції.
2. Що таке командний цикл?
3. Дві основні фази командного циклу.
4. Основні вузли процесора.
5. Одношинна структура процесора комп'ютера із складною системою команд і його зв'язки з іншими пристроями комп'ютера.
6. Виконання процесором операції "Вибірка слова з пам'яті".
7. Виконання процесором операції "Запам'ятовування слова в пам'яті".
8. Виконання процесором операції обміну між регістрами.
9. Виконання процесором арифметичних і логічних операцій.
10. Порівняння одношинної та багатошинної структур процесора комп'ютера із складною системою команд.

11. Чому в процесорі комп'ютера із складною системою команд команда виконується за багато тактів?

12. Чому в процесорі комп'ютера із складною системою команд потрібна складна система розпізнавання команди?

13. Чому в процесорі комп'ютера із складною системою команд організація конвеєризації виконання команд складніша, ніж у процесорі комп'ютера з простою системою команд?

14. Основні вимоги до процесора комп'ютера з простою системою команд.

15. Сформуйте правила вибору системи команд комп'ютера з простою системою команд.

16. Чому в системі команд комп'ютера з простою системою команд відносно небагато операцій та способів адресації?

17. Чому в комп'ютері з простою системою команд команди обробки даних мають реалізуватися лише у формі "регістр-регістр"?

18. Чому в комп'ютері з простою системою команд обміни з основною пам'яттю виконуються лише за допомогою команд завантаження/запису?

19. Чому в процесорі комп'ютера з простою системою команд дешифрування команд із спрощеними форматами має виконуватися лише апаратно?

20. Що є основою проектування структури процесора комп'ютера з простою системою команд?

21. Як будується процесор для того, щоб команда виконувалася за один такт?

22. Поясніть принципи роботи процесора комп'ютера DLX.

23. Опишіть фази виконання команди в процесорі комп'ютера DLX.

24. Поясніть роботу конвеєрного процесора комп'ютера DLX.

25. Проаналізуйте та поясніть мікродії, що виконуються на сходинці IF конвеєра комп'ютера DLX.

26. Проаналізуйте та поясніть мікродії, що виконуються на сходинці ID конвеєра комп'ютера DLX.

27. Проаналізуйте та поясніть мікродії, що виконуються на сходинці EX конвеєра комп'ютера DLX при виконанні команди АЛП.

28. Проаналізуйте та поясніть мікродії, що виконуються на сходинці EX конвеєра комп'ютера DLX при виконанні команд завантаження і збереження (load/store).

29. Проаналізуйте та поясніть мікродії, що виконуються на сходинці EX конвеєра комп'ютера DLX при виконанні команди умовного переходу (branch).

30. Проаналізуйте та поясніть мікродії, що виконуються на сходинці MEM конвеєра комп'ютера DLX при виконанні команд АЛП.

31. Проаналізуйте та поясніть мікродії, що виконуються на сходинці MEM конвеєра комп'ютера DLX при виконанні команд завантаження або збереження.

32. Проаналізуйте та поясніть мікродії, що виконуються на сходинці WB конвеєра комп'ютера DLX при виконанні команд АЛП.

33. Проаналізуйте та поясніть мікродії, що виконуються на сходинці WB конвеєра комп'ютера DLX при виконанні команди load.

34. Основна ідея суперконвеєрних процесорів.

35. Суперскалярні процесори - структура та принцип роботи.

36. Процесори векторних комп'ютерів - структура та принцип роботи.

37. Наведіть класифікацію архітектури комп'ютера за рівнем суміщення в ньому опрацювання команд та даних.

Розділ 5

Запшшштя/ конфліктам/ & кші&еєрі команд

У попередньому розділі були розглянуті структури процесора, які забезпечують суміщений в часі режихМ виконання кількох команд, коли вони є незалежними одна від одної. Це суміщення називається конвеєрним виконанням команд. У даному розділі буде розглянуто ряд методів підвищення ефективності конвеєрного виконання команд. Потреба в цьому викликана тим, що при реалізації конвеєрного виконання команд виникають ситуації, які перешкоджають виконанню чергової команди з потоку команд в призначеному для неї такті. Такі ситуації називаються конфліктами, або ризиками. Конфлікти знижують продуктивність конвеєра, яка могла б бути досягнута в ідеальному випадку. Більше того, конфлікти можуть звести нанівець всі затрати на створення конвеєра команд.

Існує три класи конфліктів:

- Структурні конфлікти, які виникають з причини браку ресурсів, коли апаратні засоби не можуть підтримувати всі можливі комбінації команд в режимі одночасного виконання з перекриттям.
- Конфлікти за даними, що виникають у разі, коли виконання наступної команди залежить від результату виконання попередньої команди.
- Конфлікти керування, які виникають при конвеєризації команд передачі керування, які змінюють значення лічильника команд.

Конфлікти в конвеєрі призводять до необхідності призупинення виконання команд. Звичайно, якщо призупиняється виконання якої-небудь команди в конвеєрі, то виконання всіх наступних за нею команд також призупиняється і, зрозуміло, під час призупинення не вибирається жодна нова команда. При цьому команди, передуючі призупиненій, можуть продовжувати виконуватися.

Спочатку будуть розглянуті методи, що дозволяють знизити вплив структурних конфліктів, а також конфліктів за даними та конфліктів керування, а потім питання розширення можливостей комп'ютера по використанню паралелізму, закладеного в програмах. Далі буде проведено аналіз сучасних технологій компіляторів, які використовуються для збільшення ступеня паралелізму рівня команд.

5. 7. *Структурні конфлікти*

Конвеєризація виконання команд в комп'ютері вимагає значних додаткових затрат обладнання, особливо якщо забезпечується безконфліктне виконання в конвеєрі всіх можливих комбінацій команд. Якщо яка-небудь комбінація команд не може бути вико-

нана в конвеєрному режимі через причини браку ресурсів, то говорять, що в комп'ютері є структурний конфлікт.

Існує дві групи структурних конфліктів. До першої групи належать структурні конфлікти, які виникають через потребу порушення тактової частоти роботи конвеєра. До другої групи належать структурні конфлікти, які виникають у зв'язку з необхідністю очікування на звільнення ресурсів комп'ютера.

Найбільш типовим прикладом комп'ютерів, у яких можлива поява структурних конфліктів першої групи, є комп'ютери з не повністю конвеєрними функціональними пристроями. Час виконання операції в такому пристрої може складати декілька тактів синхронізації конвеєра. В цьому випадку послідовні команди, які використовують даний функціональний пристрій, не можуть надходити до нього в кожному такті. Прикладом такого не повністю конвеєрного функціонального пристрою може бути пристрій додавання-множення на основі одного суматора, в якому додавання виконується за один такт, а множення з метою економії обладнання реалізується шляхом ітераційного виконання операції додавання. Такий підхід є виправданим при нечастому виконанні операції множення.

Можлива також інша, але досить подібна за впливом на конвеєр ситуація, коли в останньому включений пристрій, затримка в якому більша одного такту конвеєра. Наприклад, це може бути паралельний однотактовий поділювач двійкових чисел, затримка в якому перевищує такт конвеєра, а його конвеєризація є невиправдано дорогою із-за причини маловживаності цієї операції.

Поява структурних конфліктів першої групи викликає необхідність призупинення роботи конвеєра до закінчення роботи функціонального пристрою, звернення до якого викликало конфлікт. Кількість тактів простою конвеєра рівна відношенню часу виконання операції в функціональному пристрої до величини такту конвеєра. На рис. 5.1 наведено діаграму виконання на симуляторі WinDLX нижче приведеного фрагмента деякої програми

```
addff6,f5,f4
xor rl,r3,r4
and r5,r6,r7,
```

у якій виник структурний конфлікт першої групи, викликаний наявністю в програмі команди додавання чисел з рухомою комою addf. Фаза виконання EX цієї команди є довшою в три рази, ніж такт роботи конвеєра. Тому фази MEM команд addf та and співпали, тобто вимагається одночасний доступ до ресурсів фази MEM, що і стало причиною конфлікту. Через те виконання команди and призупиняється (Stall - зупинка). Час призупинення визначається алгоритмом та структурою пристрою додавання чисел з рухомою комою.

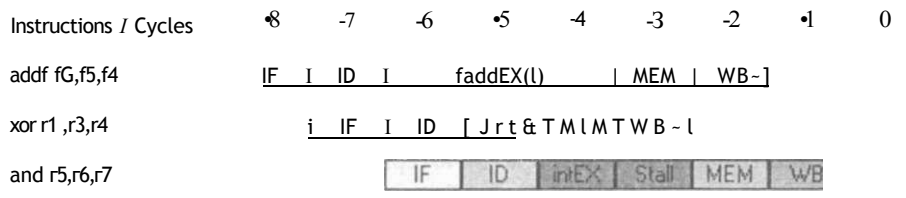


Рис. 5.1. Структурний конфлікт першої групи

Друга група структурних конфліктів пов'язана з недостатньою кількістю деяких ресурсів (функціональних блоків, портів і т. д.), що перешкоджає виконанню довільної послідовності команд в конвеєрі без його призупинення. Наприклад, процесор може мати тільки один порт запису в регістрову пам'ять, але при певних обставинах конвеєру може виникнути необхідність виконати два записи в регістрову пам'ять в одному такті. Це призводить до структурного конфлікту. Коли послідовність команд натрапляє на такий конфлікт, виконання однієї з команд призупиняється до тих пір, поки не стане доступним необхідний пристрій.

На рис. 5.2 показано приклад такого конфлікту для наступного фрагмента програми:

cvti2ffl.fi

суй2ff2,f2

ггаЛГ &, *Н, а*

тиШб,г3,г4.

Структурний конфлікт виникає на такті -8, коли відбувається звернення до пристрою множення для виконання фази EX команди типі f6.f3.f4. В цей час пристрій множення зайнятий виконанням фази EX попередньої команди множення, яка триває 5 тактів. Щоб зняти цей конфлікт, потрібно просто призупинити конвеєр на 4 такти, поки відбувається одночасне звернення до пристрою множення (Б-Бїаіі - структурна зупинка). Подібне призупинення часто називається конвеєрною булькою, оскільки булька проходить по конвеєру, займаючи місце, але не виконуючи ніякої корисної роботи. Потрібно відзначити, що якби множення виконувалось за один такт, структурний конфліктне виникнув би.

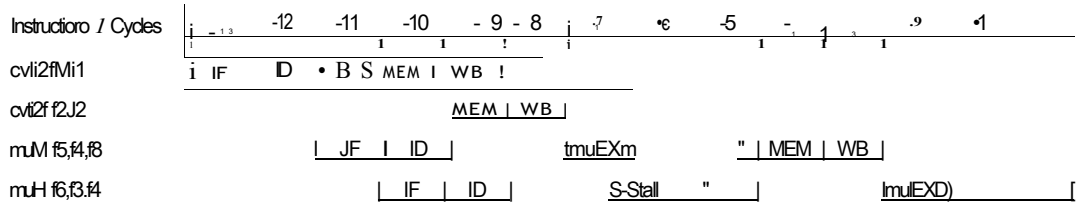


Рис. 5.2. Приклад структурного конфлікту при зверненні до пристрою множення

Структурні конфлікти другої групи виникають також при одночасному зверненні до пам'яті даних, наприклад, при необхідності читання операнда для однієї команди та запису результату для другої. Такі конфлікти виникають і в комп'ютерах з однією пам'яттю команд і даних. У цьому випадку, коли одна команда, яка знаходиться в конвеєрі, здійснює звернення до пам'яті за даними, воно конфліктуватиме з вибіркою пізнішої в черзі команди, яка вибирається з тієї ж пам'яті. В компютері DLX можливий конфлікт, наприклад, при одночасному зверненні до основної пам'яті з боку кеш пам'яті даних та команд.

Зрозуміло, що комп'ютер, в якому забезпечена підтримка конвеєрного виконання команд без структурних конфліктів, завжди матиме вищу продуктивність порівняно з комп'ютером із структурними конфліктами. Виникає питання: чому розробники допускають наявність структурних конфліктів? Цьому є дві причини. По-перше, ряд структурних конфліктів принципово дуже важко або й неможливо ліквідувати для всіх випадків роботи конвеєра, наприклад, вирішити питання одночасного доступу до пам'яті

даних. По-друге, конвеєризація всіх функціональних пристроїв може виявитися дуже дорогою. Комп'ютери, які допускають два звернення до пам'яті в одному такті, повинні мати пам'ять, яка характеризується подвоєною пропускною спроможністю, наприклад, це може бути досягнуто шляхом використання окремих блоків кеш пам'яті для команд і даних. Аналогічно, повністю конвеєрний пристрій ділення з рухомою комою вимагає значної кількості вентилів. Якщо структурні конфлікти не виникатимуть дуже часто, то не завжди варто платити за те, щоб їх обійти. Тому розробляється скалярний, або не повністю конвеєрний пристрій, що має меншу загальну затримку, ніж повністю конвеєрний. Наприклад, розробники пристроїв з рухомою комою комп'ютерів CDC7600 і MIPS R2010 вважали за краще мати меншу затримку виконання операцій замість повної їх конвеєризації.

5.2. Конфлікти за даними

5.2.1. Типи конфліктів за даними

Одним з чинників, який істотно впливає на продуктивність конвеєрних комп'ютерів, є логічні залежності між командами. Такі залежності великою мірою обмежують потенційний паралелізм суміжних операцій виконання команд, що забезпечується відповідними апаратними засобами. Ступінь впливу цих залежностей визначається як структурою комп'ютера (в основному, організацією керування конвеєром команд і характеристиками функціональних пристроїв), так і характеристиками програм.

Конфлікт за даними виникає при наявності залежності між командами, коли вони розташовані в програмі близько одна до одної. В цьому випадку для забезпечення перекриття виконання операцій залежних команд може виникнути необхідність у зміні порядку звернення за операндами.

Відомі три можливі конфлікти за даними залежно від порядку операцій читання і запису. Розглянемо дві команди - і та j, при цьому команда і передує команді j. Можливі наступні конфлікти:

1. WAR (write after read) - читання після запису. Команда) намагається прочитати ще не оновлений командою і операнд (рис. 5.3). Якби не було конвеєра, то спочатку попередня команда і записала б до комірки X операнд, який пізніше був би зчитаний командою). У випадку використання конвеєрного опрацювання команд, як це видно з рисунка, фаза читання команди j виконується раніше фази запису команди і. Таким чином, команда j може некоректно набути старого значення.

Рис. 5.3. Конфлікт "запис після читання"

2. RAW (read after write) - запис після читання. Команда j намагається записати операнд до регістра призначення ще до того, як попередній вміст цього регістра прочитає команда і (рис. 5.4). Якби не було конвеєра, то спочатку попередня команда і прочитала

б з комірки X операнд, а пізніше до цієї комірки був би записаний інший операнд командою]. У випадку використання конвеєрного опрацювання команд, як це видно з рисунка, фаза запису команди] виконується раніше фази читання команди і. Таким чином, команда і може набути некоректного нового значення.

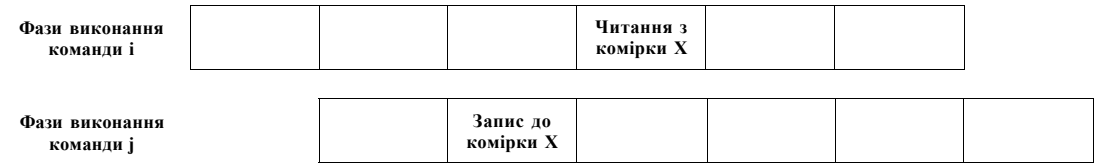


Рис. 5.4. Конфлікт "читання після запису"

3. WAW (write after write) - запис після запису. Команда j намагається записати операнд до регістра призначення ще до того, як цей запис провела команда і, тобто записи закінчуються в неправильному порядку (рис. 5.5). Якби не було конвеєра, то спочатку попередня команда і записала б до комірки X операнд, а пізніше до цієї комірки був би записаний інший операнд командою j. У випадку використання конвеєрного опрацювання команд, як це видно з рисунка, фаза запису команди j виконується раніше фази запису команди і. В результаті комірка тимчасово отримує некоректний вміст, чим може «скористатися» проміжна k-та команда.

Рис. 5.5. Конфлікт "запис після запису"

Можливий також випадок RAR (read affer read) - читання після читання, але він небезпеки не створює, і тому не розглядається. Означення, класифікацію та перші методи скасування конфліктів за даними (в оригіналі - data hazards) запропонував Роберт Келлер в 1975 році.

5.2.2. *Методи зменшення впливу конфліктів за даними на роботу конвеєра команд*

Застосовуються наступні методи зменшення впливу зазначених вище залежностей між даними на роботу конвеєра команд:

- Призупинення виконання команди, тобто затримка з переходом від виконання операції декодування ID до виконання операції виконання EX в конвеєрі доти, доки залежність даних не вичерпується плином часу.
- Випереджувальне пересилання з ярусів конвеєра результатів попередньої команди до потрібного ярусу конвеєра, в якому виконується наступна команда (це потребує додаткових затрат обладнання та ускладнює керування).
- Статична диспетчеризація послідовності команд у програмі під час компіляції з метою зменшення впливу конфліктів за даними на роботу конвеєра команд шляхом зміни порядку виконання залежних одна від одної команд.

" Динамічна диспетчеризація послідовності команд у програмі під час компіляції з тією ж, що й статична диспетчеризація, метою.

• Перейменування регістрів.

Розглянемо далі названі методи зменшення впливу конфліктів за даними на роботу конвеєра команд детальніше.

5.2.3. *Призупинення виконання команди*

Найпростішим рішенням для зменшення впливу конфліктів на роботу конвеєра команд при наявності залежності між даними є призупинення виконання команди] на декілька тактів з тим, щоб завершилось виконання команди і, або тієї її фази, яка викликала конфлікт. Відповідно затримається і виконання команд, які йдуть слідом за командою і.

Розглянемо фрагмент програми, виконання якого вимагає призупинення роботи конвеєра:

```
ш       R1,0(R2)
SUB     R4,R1,R5
AND     R6,R1,R7
OR      R8,R1,R9
```

Рис. 5.6 ілюструє роботу конвеєра при виконанні поданого вище фрагмента програми.

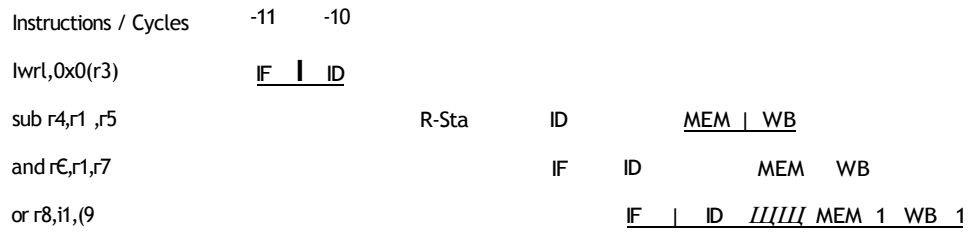


Рис. 5.6. Призупинення конвеєра

Тут друга команда SUB потребує результату виконання першої команди, який з'являється лише на такті 12 (фаза MEM), через що виконання призупиняється (зупинка R-Stall) до завершення вказаного такту. Призупинення конвеєра знижує його ефективність, тому частіше використовують інші шляхи ліквідації впливу залежностей між даними.

5.2.4. *Випереджувальне пересилання*

Найпоширенішим способом запобігання конфліктам в конвеєрі команд, поява яких викликана залежностями між даними, є випереджувальне пересилання даних із попередніх ярусів конвеєра в яруси, де ці дані потрібні, минаючи проміжні яруси конвеєра. Розглянемо застосування випереджувального пересилання на кількох прикладах для конкретних фрагментів коду програми.

Випереджувальне пересилання даних з метою усунення конфлікту і пригальмовування конвеєра комп'ютера DLX для коду програми

```
SUB R5 , R6 , R7
LW   R4,0(R5)
SW   12(R5),R6
```

ілюструє рис. 5.7, на якому показано результат виконання приведеної програми на симуляторі комп'ютера DLX WinDLX. Залежність за даними виникає за рахунок використання другою командою вмісту регістра R5 за умови, що цей вміст оновлюється попередньою (першою) командою фрагмента. Цю залежність долають шляхом випереджувального пересилання результату першої команди (що використовується потім як зсувний компонент адреси) з інтегрованого (до конвеєрного регістра) поля ALUoutput (на такті -7) на вхід ALU (на такті -6). На рисунку це позначено похилою додатковою лінією. Випереджувальне пересилання грунтується на тому, що реально зсувна компонента формується вчасно, вона вже існує в надрах конвеєра, але не є ще занесеною до регістра регістрового файла (для цього треба виконати ще фази MEM і WB).

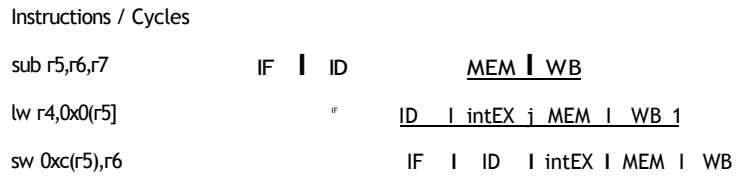


Рис. 5.7. Випереджувальне пересилання

Випереджувального пересилання вимагає і остання (третя) команда через те, що використовує вміст регістра Я4, який формується другою командою. Третій команді потрібно ще одне випереджувальне пересилання зсувної компоненти виконавчої адреси (вмісту регістра Я5), що позначено на рисунку іще однією додатковою лінією.

Потрібно зазначити, що випереджувальне пересилання не завжди є можливим. Розглянемо виконання в конвеєрі наступного фрагмента коду

```
LW    R1,0(R2)
ADD   R4,R1,R2
AND   R6,R1,R7
XOR   R3 , R1 , R5
```

Залежність даних обумовлена використанням командами, які надходять після першої команди, вмісту регістра R1, який у конвеєрі формується із стандартною затримкою. При цьому друга команда використовує випереджувальне пересилання даних від першої команди після завершення фази MEM, коли вже отримано майбутній вміст регістра R1. Але оскільки цей вміст отримано лише на такті -6, а він був потрібний на такті -7, то з'явилась затримка R-Stall, тому, що принципово неможливе пересилання до минулого (рис. 5.8).

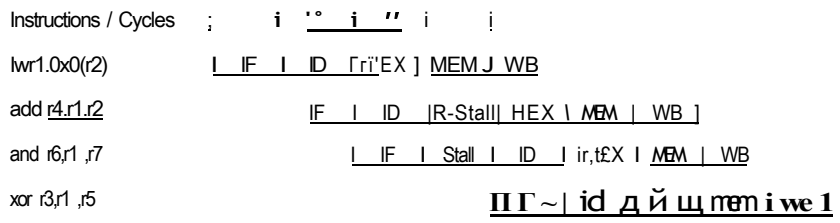


Рис. 5.8. Можливі та неможливі випереджувальні пересилання

Для забезпечення реалізації в конвеєрі випереджувального пересилання в ньому формуються додаткові канали пересилання даних з відповідними мультиплексорами для підключення входів функціональних блоків комп'ютера до цих каналів та додаткові буферні регістри

5.2.5. *Статична диспетчеризація послідовності команд у програмі під час компіляції*

У деяких комп'ютерах для зменшення кількості конфліктів за даними, аж до повної їх ліквідації, застосовують програмні методи, що передбачають створення оптимізуючих компіляторів, які орієнтовані на усунення можливості появи конфліктів у конвеєрі іще на стадії компіляції програми. Оптимізуючи компілятори, реорганізовують послідовність команд та створюють такий об'єктний код, щоб між конфліктуючими командами (командами, здатними до конфлікту) знаходилась достатня кількість неконфліктуючих команд. Якщо компілятору цього зробити не вдається, то він вставляє між конфліктуючими командами необхідну кількість команд типу «немає операції». Такий підхід виключає необхідність призупинення роботи конвеєра і виконання випереджувального пересилання, яке вимагає додаткових затрат обладнання та ускладнює керування

Для виявлення конфліктуючих команд в оптимізуючому компіляторі потрібно реалізувати відповідні засоби аналізу команд. Ознаками наявності конфліктів за даними є наступні

- для конфліктів за даними типу RAW - наявність збіжностей в адресах читання попередніх команд та адресах запису наступних команд,
- для конфліктів за даними типу WAR - наявність збіжностей в адресах запису попередніх команд та адресах читання наступних команд,
- для конфліктів за даними типу WAW - наявність збіжностей в адресах запису попередніх команд та адресах запису наступних команд

Тобто оптимізуючий компілятор повинен проводити аналіз на збіжність адрес запису і читання сусідніх команд, знаходити конфліктуючі команди, та розводити їх в часі

Статична диспетчеризація послідовності команд у програмі під час компіляції використовувалася, починаючи з 60-х років минулого століття, і викликала підвищений інтерес у 80-х роках, коли конвеєрні машини стали поширенішими

Нехай, наприклад, є послідовність операторів: a = b + c;d = e-f. Нижче наведена не оптимізована програма для виконання цих операторів

```
LW r2,0(r0)
LW r3,4(rO)
ADD r4,r2,r3
```

```
5\У  8(г0),г4
ІЛУ  г5,12(г0)
1Л¥г6,16(гО)
БиВ  г7,г5,г6
     20(г0),г7
```

Вхідні дані та результати цієї програми розміщені в комірках пам'яті даних, наведених в табл. 5.1.

Таблиця 5.1

Операнд	Адреса комірки пам'яті
а	8
Ь	0
с	4
а	20
е	12
і	16

Діаграма виконання програми приведена на рис. 5.9. Як видно з рисунка, тут наявні чотири затримки, викликанані залежністю за даними: друга і третя команди - звернення до регістра г3, третя і четверта команди - звернення до регістра г4, шоста і сьома команди - звернення до регістра гб, сьома і восьма команди - звернення до регістра г7.

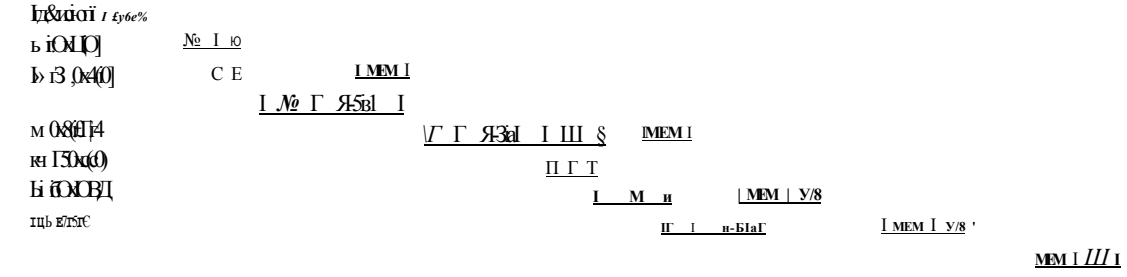


Рис. 5.9. Діаграма виконання неоптимізованої програми

Знаходимо конфліктуючі команди в наведеній вище програмі та розводемо їх в часі. Нижче наведена оптимізована програма для виконання тих же операторів

```
Ш   г2,0(г0)
Ж   г3,4(гО)
ІЛ¥г5,12(гО)
ІЛУг6,16(гО)

АОЭ  г4,г2,г3
БИВ  г7,г5,г6
Б\У  8(г0),г4
Б\У  20(г0),г7
```

Вхідні дані та результати розміщені в тих же комірках пам'яті, приведених в табл. 5.1. Тоді діаграма виконання програми буде мати вигляд, як це приведено на рис. 5.10 (отримана з використанням симулятора \VinDLX).

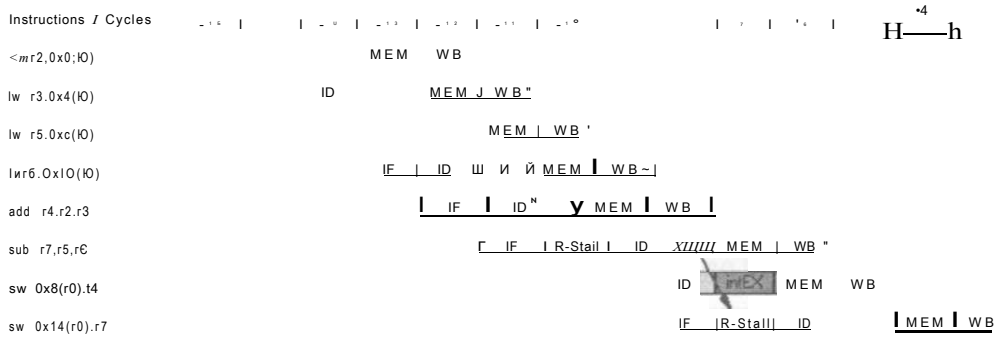


Рис. 5.10. Діаграма виконання оптимізованої програми

Як наслідок, повністю усунені два блокування (стосовно регістрів г3 і г4), та в двох блокуваннях (стосовно гб і г7) забрано по одному такту затримки. Тобто в цілому забрано 6 тактів. Залежності, які залишилися, можна забрати, використавши схеми обходу.

Відзначимо, що використання різних регістрів для першого і другого операторів було достатньо важливим для реалізації такого правильного планування. Зокрема, якби змінна e була б завантажена в той же самий регістр, що b або c, то таке планування не було б коректним. У загальному випадку планування конвеєра може вимагати збільшеної кількості регістрів.

Статична диспетчеризація передбачає реорганізацію послідовності команд, серед яких відсутні команди переходу, за винятком початку і кінця цієї послідовності. Планування такої послідовності команд здійснюється досить просто, оскільки в компіляторі передбачено, що кожна наступна команда виконуватиметься, якщо виконується перша з них. Тому можна побудувати граф залежностей цих команд і впорядкувати їх так, щоб мінімізувати кількість призупинень конвеєра. Для простих конвеєрів використання статичної диспетчеризації дає цілком задовільні результати. Проте, коли довжина конвеєра збільшується і його затримки зростають, потрібні складніші алгоритми диспетчеризації.

5.2.6. *Динамічна диспетчеризація послідовності команд у програмі під час компіляції*

Динамічна диспетчеризація передбачає проведення компіляції програми узгоджено з процесом виконання команд в комп'ютері з метою мінімізації кількості призупинень конвеєра. У англомовній літературі разом з терміном "динамічна диспетчеризація" часто застосовуються терміни "out-of-order execution" - невпорядковане виконання і "out-of-order issue" - невпорядковане видавання, які практично завжди наявні при динамічній диспетчеризації. При динамічній оптимізації в конвеєрі команд розміщується пристрій виявлення конфліктів, який інформує компілятора про наявність конфліктів з тим, щоб він міг реагувати в процесі компіляції на ці конфлікти. Одним з варіантів реагування є

буферизація команд, що чекають вирішення конфлікту, і подання подальших, логічно не пов'язаних команд, в конвеєр. При цьому буферизовані команди можуть подаватися в потрібний ярус конвеєра, що забезпечується створенням в ньому комутуючих магістралей, які, крім того, забезпечують засилання результату операції безпосередньо в буфер, що зберігає логічно залежну команду, затриману через конфлікт, або безпосередньо на вхід функціонального пристрою до того, як цей результат буде записаний в регістровий файл або в пам'ять.

При динамічній диспетчеризації команди можуть подаватися на виконання не в тому порядку, в якому вони розташовані в програмі, проте засоби виявлення і усунення конфліктів між логічно зв'язаними командами мають забезпечувати отримання результатів відповідно до заданої програми.

5.2.7. *Перейменування регістрів*

Ще одним апаратним методом мінімізації конфліктів за даними є метод перейменування регістрів. При реалізації цього методу в адресних полях команди вказуються номери не фізичних, а логічних (уявних) регістрів. Номери логічних регістрів динамічно відображаються на номери фізичних регістрів, які розміщуються в регістровому файлі процесора, за допомогою таблиць відображення, котрі оновлюються після декодування кожної команди. Кожен новий результат записується в фізичний регістр.

Проте попереднє (тимчасове) значення кожного логічного регістра зберігається і може бути відновлене, якщо виконання команди має бути перерване через виникнення виняткової ситуації або у випадку невірного передбачення напряму умовного переходу. В процесі виконання програми генерується велика кількість тимчасових результатів. Ці тимчасові результати записуються в регістрові файли разом з постійними результатами. Тимчасовий результат стає новим постійним результатом, коли завершується виконання команди. У свою чергу, завершення виконання команди відбувається, коли виконання всіх попередніх команд успішно завершено в заданому програмою порядку.

Програміст має справу тільки з логічними регістрами. Реалізація фізичних регістрів від нього прихована.

Метод перейменування регістрів спрощує контроль залежностей за даними. У комп'ютері, який може виконувати команди не у порядку їх розташування в програмі, номери логічних регістрів можуть стати двозначними, оскільки один і той же регістр може бути призначений послідовно для зберігання різних значень. Але оскільки номери фізичних регістрів унікально ідентифікують кожен результат, всі неоднозначності усуваються.

5.3. *Конфлікти керування*

5.3.7. *Типи конфліктів керування*

Конфлікти керування можуть викликати ще більше зниження продуктивності конвеєра, ніж конфлікти за даними. Вони викликані наявністю в програмах команд керування, які змінюють хід обчислювального процесу: безумовний та умовний переходи, пропуск, виклик процедури та повернення з неї. Виконання команд керування може

призводити до необхідності очистки конвеєра та завантаження нової послідовності команд, що знижує його продуктивність. Те, що команда належить до команд керування, можна виявити лише після її декодування, тобто за кілька тактів після її надходження до конвеєра (в комп'ютері DLX через 2 такти).

За цей час на перші яруси конвеєра вже надійдуть нові команди. При виявленні команди керування потрібно виявити, чи здійснюється перехід, і якщо так, то очистити конвеєр від наступних за нею команд та завантажити команду, розміщену за адресою переходу. Для цього спочатку потрібно визначити виконавчу адресу переходу, яка може бути або безпосередньо в адресному полі команди, або її потрібно обчислити в наступних ярусах конвеєра. Таким чином, реалізація в конвеєрі команд керування вимагає виконання додаткових операцій, що рівнозначно його зупинці на відповідну кількість тактів

Особливо складною для виконання в конвеєрі є команда умовного переходу. Коли виконується команда умовного переходу, вона може або змінити вміст лічильника команд, або залишити його без змін. Якщо команда умовного переходу замінює лічильник команд значенням адреси, обчисленої в команді, то перехід називається здійснимим. В іншому випадку він називається нездійснимим. Для забезпечення опрацювання в конвеєрі команд умовного переходу потрібно передбачити проведення відповідних дій. Простий метод роботи з умовними переходами полягає в призупиненні конвеєра як тільки виявлена команда умовного переходу до тих пір, поки вона не досягне ярусу конвеєра, який обчислює нове значення лічильника команд. Реалізація цього методу для наведеного нижче фрагмента програми

```
dd r7,r8,rl
bnez r4,$TEXT
and r5,r7,r7
add r2,r2,r3
```

відображена на рис. 5.11, де після декодування команди BNEZ (перехід, якщо не рівний нулю) призупиняється виконання наступної команди.

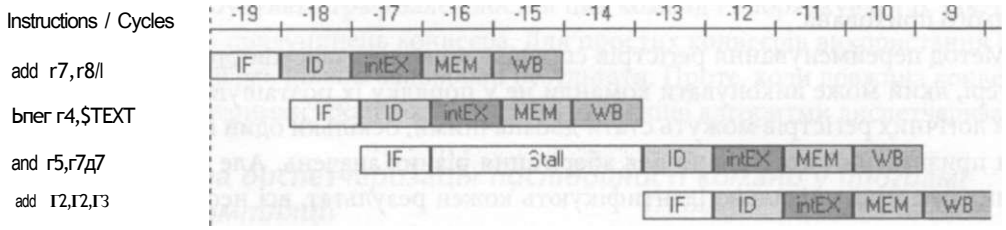


Рис. 5.11. Призупинення конвеєра при виконанні команди умовного переходу

Комп'ютер, в якому здійснюються призупинення при виявленні команд умовних переходів, суттєво втрачає прискорення, що одержується за рахунок конвеєрної організації. При цьому, чим більша глибина конвеєра, тим більші втрати на командах умовного переходу

Таким чином, для зменшення втрат в конвеєрі через конфлікти керування потрібно звернути увагу в першу чергу на організацію вибірки команди, до якої здійснюється перехід, та на скорочення часу виконання команд умовного переходу

5.3.2. Зниження втрат на вибірку команди, до якої здійснюється перехід

Для зниження втрат на вибірку команди, до якої здійснюється перехід, використовуються декілька підходів:

- обчислення виконавчої адреси команди переходу в ярусі декодування команди;
- використання буфера адрес переходів;
- використання буфера команд, до яких здійснюються переходи;
- використання буфера циклу.

Обчислення виконавчої адреси команди переходу в ярусі декодування команди передбачає на етапі декодування команди визначення не тільки її належності до команд керування, але і адреси переходу. Ця адреса може бути в полі самої команди, а якщо ні, то має бути обчислена в цьому ж ярусі. Для забезпечення обчислення адреси переходу в ярус декодування вводиться додатковий суматор. Тим самим час зупинки конвеєра може бути зменшений до одного такту.

При використанні буфера адрес переходів, який є асоціативною пам'яттю, в ньому зберігаються адреси декількох останніх команд переходів, які є ознакою пошуку, та відповідні їм адреси переходу. Перед вибіркою з пам'яті чергової команди її адреса (вміст лічильника команд) порівнюється з адресами наявних в буфері команд і, якщо відбулося співпадіння, з буфера вибирається адреса переходу (рис. 5.12), а сигнал про наявність адреси в буфері пропускає її через мультиплексор. Тоді вибірка команди переходу може бути здійснена вже в наступному такті. При відсутності адреси команди в буфері адреса переходу шукається в ярусі декодування та, разом з адресою команди, заноситься до буфера адрес переходів. При необхідності заміщення інформації в буфері адрес переходів використовуються алгоритми заміщення типу FIFO, LRU, RAND.

Рис. 5.12. Використання буфера адрес переходів

Кращим, ніж використання буфера адрес переходів, є використання буфера команд, до яких здійснюється перехід (рис. 5.13), коли в буфер разом з адресою команди переходу, яка є ознакою пошуку, записуються коди команд, до яких здійснюються переходи, що дозволяє при повторному виконанні такої команди виключити не тільки фазу обчислення адреси переходу, але і фазу вибірки команди. При відсутності в буфері команди, до якої здійснюється перехід, її адреса визначається в ярусі декодування, проводиться її зчитування з пам'яті команд та, разом з адресою команди переходу, вона заноситься до буфера.

Рис. 5.13. Використання буфера команд переходів

Особливо ефективним є використання буфера адрес переходів та буфера команд, до яких здійснюється перехід при виконанні циклів.

Іще більший ефект дає використання буфера циклу, до якого із збереженням попереднього порядку записується деяка кількість команд, що виконувались останніми. Буфер входить до першого ярусу конвеєра, в якому виконується вибірка команд (рис. 5.14). Коли відбувається перехід, то спочатку здійснюється звернення до буфера. Якщо це був повторний цикл або ітерація, то всі його команди будуть вже наявними в буфері, який є меншим за об'ємом і швидшим основної пам'яті та кеш пам'яті команд. Тим самим всі наступні ітерації будуть виконуватись із зчитуванням команд з буфера, що суттєво прискорює роботу комп'ютера. Описаний підхід зокрема був використаний в комп'ютерах CRAY-1 фірми Cray Research та CDC 6600 фірми Control Data Corporation.

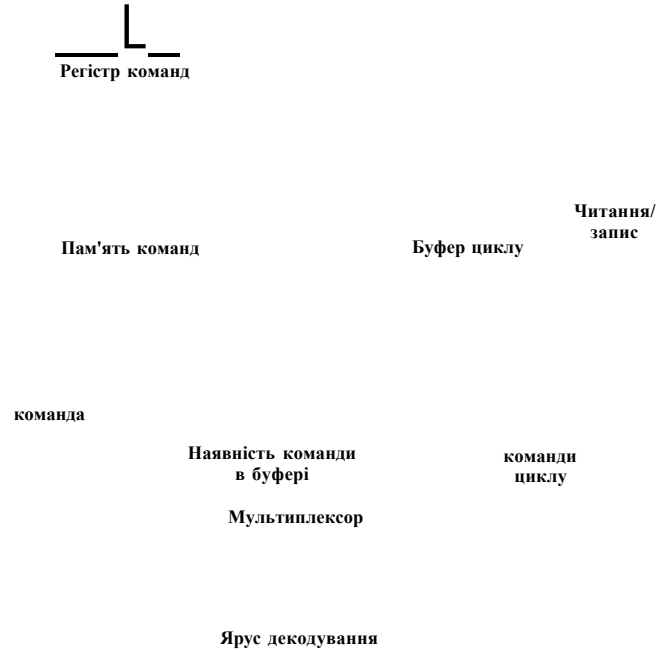


Рис. 5.14. Використання буфера циклу

5.3.3. Зниження втрат на виконання команд умовного переходу

Число тактів, що втрачаються під час призупинення конвеєра через наявність умовних переходів, може бути зменшено наступними способами:

- Введенням буфера попередньої вибірки для виявлення, здійснимим чи нездійснимим є умовний перехід та підготовки до переходу на початкових ярусах конвеєра.
- Дублюванням початкових ярусів конвеєра для підготовки до переходу, аж до виявлення, здійснимим чи нездійснимим є умовний перехід.
- Затримкою переходу, тобто виконанням наступних за командою переходу команд незалежно від напряму переходу.
- Статичним та динамічним передбаченнями переходу, тобто попереднім обчисленням значення лічильника команд (цільової адреси) для здійснимого переходу.

5.3.3.7. *Введення буфера попередньої вибірки*

Буфер попередньої вибірки встановлюється після ярусу вибірки команди. Він складається з двох блоків пам'яті типу FIFO та блоку обчислення цільової адреси переходу, і включений в конвеєр команд, як це показано на рис. 5.15.

Рис. 5.15. Конвеєр з буфером попередньої вибірки команд

За значенням програмного лічильника ПЛ команди з пам'яті команд зчитуються в один з буферів FIFO. Блок обчислення цільової адреси аналізує кожну зчитану команду. При виявленні команди умовного переходу він обчислює адресу переходу та керує блоками FIFO і програмним лічильником з тим, щоб забезпечити паралельне зчитування обох можливих послідовностей команд з адреси переходу в блоки FIFO. Після обчислення адреси переходу блок обчислення цільової адреси підключає до входу конвеєра команд через мультиплексор МП відповідний буфер, а вміст іншого буфера стирається.

За рахунок попередньої вибірки для виявлення того, здійсненним чи нездійсненним є умовний перехід, таким підходом вдається зменшити втрати на виконання команд умовного переходу. При цьому буфер попередньої вибірки можна розглядати як декілька додаткових ярусів конвеєра, які збільшують його початкову затримку та не впливають на продуктивність. З рис. 5.12 видно, що такий підхід ускладнює роботу конвеєра. Крім того, при наявності в програмі декількох команд умовного переходу підряд це вимагає включення додаткових блоків FIFO та іще більше ускладнює роботу комп'ютера.

5.3.3.2. Дублювання початкових ярусів конвеєра

Подібним до описаного вище способом зниження втрат на виконання команд умовного переходу є дублювання початкових ярусів конвеєра, тобто створення двох паралельних гілок початкових ярусів конвеєра команд, як це показано на рис. 5.16.

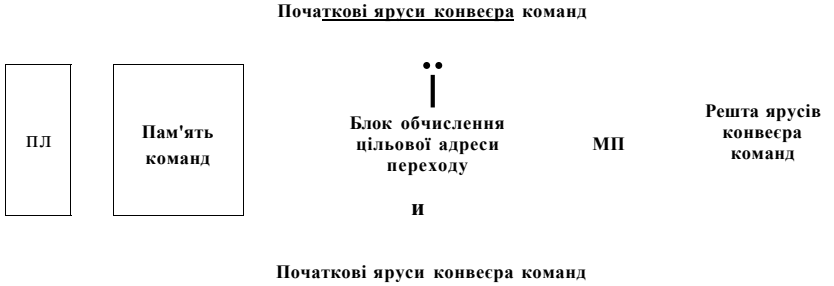


Рис. 5.16. Конвеєр з дублюванням початкових ярусів

В одній із паралельних гілок початкових ярусів конвеєра команд послідовність команд відповідає випадку, коли умова переходу виконується, а в іншій - коли не виконується. Кількість ярусів у цих вітках визначається кількістю тактів, потрібних для обчислення адреси переходу блоком обчислення цільової адреси переходу. Після отримання адреси переходу блок обчислення цільової адреси переходу через мультиплексор МП

підключає до решти ярусів конвеєра команд відповідну вітку початкових ярусів.

Тут також з'являються проблеми, коли до прийняття рішення щодо поточної команди переходу в конвеєр надходить нова команда. Тоді знову вимагаються додаткові паралельні вітки початкових ярусів конвеєра команд.

Описаний метод, як і попередній, знайшов застосування в кількох версіях сім'ї комп'ютерів IBM 360/370.

5.3.3.3. *Затримка переходу*

Ми вже бачили, що проста схема виконання в конвеєрі команд умовного переходу полягає в блокуванні виконання будь-якої команди, наступної за командою умовного переходу, до тих пір, поки не стане відомим напрям переходу. Рис. 5.11 відображав саме такий підхід. Привабливість такого рішення полягає в його простоті.

На противагу цьому методу стратегія затриманого переходу передбачає продовження виконання команд, які ідуть в програмі за командою переходу, незалежно від її результату. Це має сенс лише тоді, коли наступні за командою переходу команди мають бути виконаними незалежно від того, був перехід чи ні, а команда переходу не має на них впливу.

Пошук таких команд здійснюється на етапі компіляції програми, а якщо знайдена їх кількість недостатня, то на вільні місця вставляються команди типу «немає операції».

5.3.3.4. *Статичне передбачення переходу*

Передбачення переходу є одним з найефективніших способів подолання конфліктів керування. Ідея цього способу полягає в тому, що до моменту виконання команди умовного переходу, або відразу після її надходження в конвеєр робиться припущення про напрям виконання умовного переходу залежно від імовірності його здійсненності або нездійсненності. Команди подаються в конвеєр відповідно до прийнятого припущення. Якщо припущення виявилося правильним, то жодних втрат, пов'язаних із командою переходу, в конвеєрі не буде. При помилковому припущенні конвеєр необхідно повернути до стану, з якого почалася помилкова вибірка команд, що рівнозначно призупиненню конвеєра на втрачену кількість тактів. Вигода від застосування цього методу тим більша, чим вища точність передбачення, тобто відношення кількості правильних передбачень до загальної їх кількості.

На даний час розроблено багато способів передбачення переходу. Залежно від того, на основі чого робиться передбачення, розрізняють статичне та динамічне передбачення переходів.

Статичне передбачення переходів здійснюється на етапі компіляції програми на основі деякої апріорної інформації про неї. Найширше застосування знайшли наступні методи статичного передбачення умовного переходу:

- метод повернення;
- метод профілювання, передбачення здійснюється за результатами використання інформації про профіль виконання програми, яка зібрана в результаті попередніх її запусків;
- результат переходу визначається кодом операції команди переходу;
- результат переходу визначається напрямом переходу.

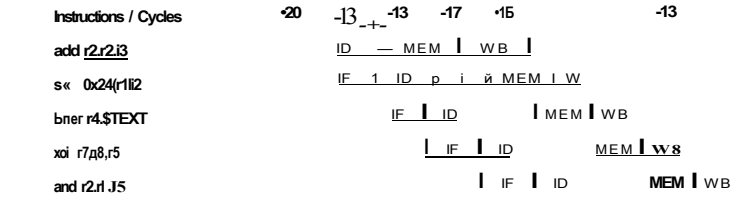
Метод повернення грунтується на передбаченні, що перехід не відбувається ніколи,

або що перехід відбувається завжди. В першому випадку умовний перехід прогнозується як нездійсненний. При цьому апаратура повинна просто продовжувати виконання програми, неначебто умовний перехід зовсім не виконувався. В цьому випадку необхідно поклопотатися про те, щоб не змінити стан комп'ютера до тих пір, поки напрям переходу не стане остаточно відомим.

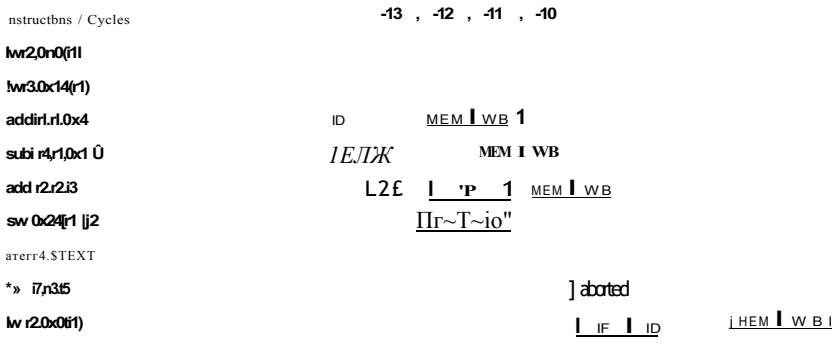
У деяких комп'ютерах ця схема з нездійсненними за прогнозом умовними переходами реалізована шляхом продовження вибірки команд, неначе умовний перехід був звичайною командою. Поведінка конвеєра виглядає так, ніби нічого незвичайного не відбувається (рис. 5.17a). Проте, якщо умовний перехід насправді виконується, то необхідно просто очистити конвеєр від команд, вибраних слідом за командою умовного переходу, і заново повторити вибірку команд (рис. 5.17b).

Розглянемо фрагмент програми

```
loop:
   lw   r2,0(r1)
   lw   r3,20(rl)
   addi  rl,rl,#4
   subi  r4,rl,#16
   add   r2,r2,r3
   sw   36(rl),r2
   bnez  r4,loop
   xor   r7,r8,r5
   and   r2дT,r5
   add   r3,r8,r2
   trap  0
```



a)



b)

Рис. 5.17. Діаграма роботи модифікованого конвеєра

Альтернативна схема прогнозує перехід як здійсненний. Як тільки команда умовного переходу декодована і обчислена цільова адреса переходу, припускається, що перехід здійсненний, і проводиться вибірка команд та їх виконання, починаючи з цільової адреси.

Метод повернення є простим в реалізації і достатньо ефективним для деяких класів програм, тому широко використовується в комп'ютерах, зокрема в МІРБ-Х, **БиргЗРАРчС,** 1486, М68020, МС88000, УАХ11/780.

Під профілюванням розуміється виконання програми для деякого еталонного набору вихідних даних, в процесі якого збирається інформація про результат кожної команди умовного переходу. Тобто тут прогноз переходів базується на інформації про профіль виконання програми, зібраної під час попередніх її проходжень. Ключовим моментом тут є те, що поведінка переходів при виконанні програми часто повторюється. Ті команди, які частіше закінчуються переходом, прогнозуються як здійсненні, інші - як нездійсненні. Вибір фіксується в спеціальному розряді команди. При виконанні програми конвеєр команд враховує значення цього розряду. Проведені дослідження показуй т? достатньо успішне прогнозування переходів з використанням цієї стратегії.

Іще одним досить простим і ефективним є метод статичного передбачення умовного переходу за кодом операції команди переходу. Тут одні команди прогнозуються як здійсненні, наприклад "більше нуля", а інші - як нездійсненні, наприклад "переповненн ***ї***

Перехід в програмі може відбутися в двох напрямках - вперед або назад, залежно від того, більша адреса переходу від вмісту програмного лічильника чи менша. Логічно передбачити, що команди з напрямком переходу назад є більш імовірними, ніж команди переходу вперед, оскільки більшість команд переходу використовується для організації циклів, коли відбуваються переходи до початку циклу. Ця стратегія і покладена в основу методу передбачення результату переходу за напрямом переходу.

5.3.3.5. Динамічне передбачення переходу

Динамічне передбачення переходу здійснюється в ході обчислень, виходячи з інформації про попередні переходи. Порівняно зі статичним динамічне передбачення має вищу точність, тобто більше припущень є правильними, але є значно складнішим.

При реалізації методів динамічного передбачення створюється таблиця історії переходів.

Найпростішим варіантом є однорозрядна таблиця історії переходів, в якій зберігається результат останнього виконання команди переходу. Якщо ця команда завершилася переходом, то у відповідну комірку таблиці записується одиниця, в іншому випадку - нуль. Передбачення переходу для чергової команди збігається із здійснимим переходом попередньої. Після виконання цієї команди, якщо передбачення не здійснилося, вміст комірки таблиці коригується.

Таблиця історії переходів реалізується в складі буфера адрес переходу (рис. 5.12). Кожен рядок буфера адрес переходу включає адресу команди переходу, прогнозовану адресу наступної команди (адресу переходу) і передісторію команди переходу (рис. 5.18). Біти передісторії є інформацією про виконання або невиконання умов переходу даної команди у минулому. Звернення до буфера адрес переходу (порівняння з полями адрес команд переходу) проводиться за допомогою поточного значення програмного лічильника на етапі вибірки чергової команди. За передісторією команди прогнозується виконання або

невиконання умов команди переходу і проводиться вибірка та дешифрування команд із прогнозованої гілки програми. При цьому, якщо виявлений збіг, то дана команда є командою умовного переходу, і адреса переходу має бути використаною в якості наступного значення програмного лічильника, якщо збігу немає, то команда не є командою переходу.

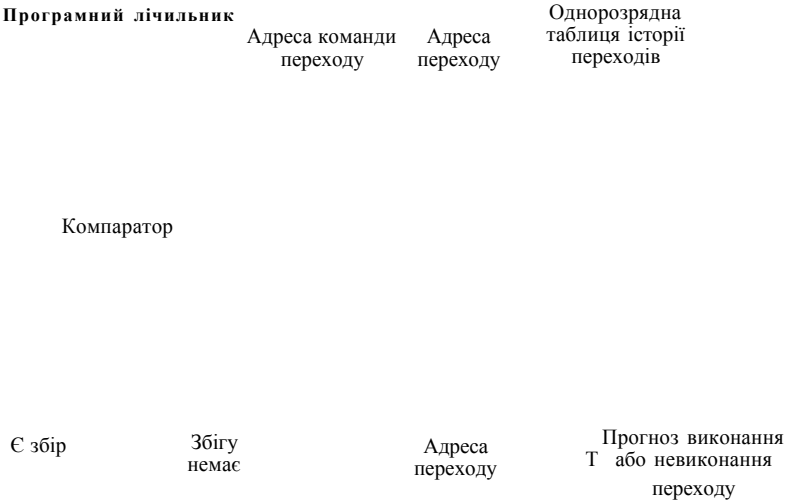


Рис. 5.18. Динамічне передбачення переходу

Більш ефективним є використання таблиці історії переходів з більшою розрядністю комірок. Практичне використання знайшли таблиці з дво- та трирозрядними комірками. Вважається, що передісторія переходу, що містить інформацію про два попередні випадки виконання цієї команди, дозволяє прогнозувати розвиток подій з цілком достатньою вірогідністю. При надходженні команди умовного переходу в конвеєр відбувається звернення до таблиці історії переходів та, залежно від вмісту відповідної комірки, робиться прогноз, який визначає подальший порядок читання команд програми. Після визначення фактичного результату переходу до вмісту комірки додається одиниця, якщо перехід відбувся, та віднімається одиниця, якщо перехід не відбувся.

В якості адреси таблиці історії переходів може бути використана адреса команди умовного переходу, вміст регістру локальної історії або регістру глобальної історії, та комбінація вказаних даних. Цим визначається вибрана стратегія динамічного передбачення.

Якщо в якості адреси таблиці історії переходів використовується адреса команди умовного переходу, тобто вміст програмного лічильника, як це показано на рис.5.18, то такий підхід дозволяє враховувати поведінку кожної команди умовного переходу, яка в більшості випадків є, як правило, здійсненною або, зазвичай, нездійсненною. Використання таблиці історії переходів дозволяє розділити команди із здійсненним і з нездійсненним умовним переходом. Функціонування цього способу формування коду передбачення, який має назву однорівневої схеми передбачення, для випадку, коли таблиця історії переходів є кількарозрядною, показано на рис. 5.19а.

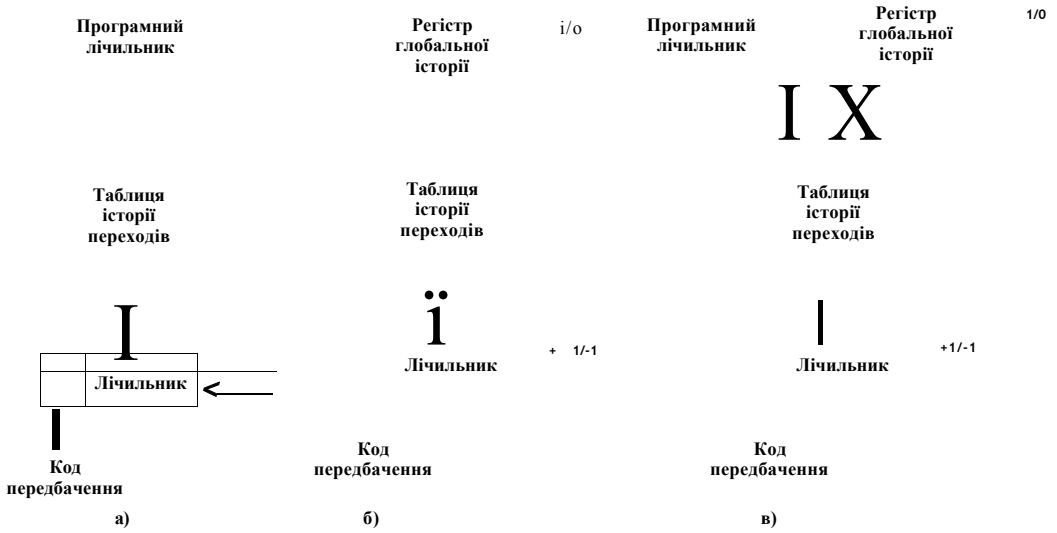


Рис. 5.19. Однорівнева схема передбачення з формуванням адреси таблиці історії переходів: а - у програмному лічильнику; б-е регістрі глобальної історії; в - з комбінованим формуванням адреси

Вміст комірки, зчитаний з таблиці історії переходів за адресою з програмного лічильника, записується в лічильник, в якому здійснюється додавання або віднімання одиниці. Лічильник працює в режимі насичення, тобто його вміст не змінюється при додаванні одиниці, коли він має максимальне значення, та не змінюється при відніманні одиниці, коли він має нульове значення. В якості передбачення використовується старший розряд лічильника. Якщо він рівний одиниці, то передбачається, що перехід є здійсненний, якщо нуль - нездійсненний. Значення цього розряду надходить в конвеєр для керування вибіркою подальших команд, а вміст лічильника після модифікації повертається за тією ж адресою в таблицю.

Описаний підхід забезпечує високу імовірність передбачення для багатократно виконуваних команд переходу. Однак для одноразово виконуваних команд переходу цей підхід не діє. Тому для таких команд переходу потрібно врахувати результати переходу попередніх команд, оскільки між ними є взаємозалежність, і це дозволяє підвищити кількість правильних передбачень. Для забезпечення врахування результатів переходу попередніх команд до схеми передбачення вводиться регістр глобальної історії, вміст якого відображає історію виконання п останніх команд умовного переходу, де п - розрядність регістра. Це є зсувний регістр, вміст якого зсувається на один розряд після кожного виконання команди умовного переходу, а до звільненого розряду заноситься одиниця або нуль залежно від наявності чи відсутності переходу відповідно. Кожному значенню регістра відповідає своя комірка в таблиці історії (рис. 5.196). Вміст цієї комірки модифікується як і в попередньому способі, а її старший розряд передбачає результат команди переходу.

Підвищення точності передбачення досягається одночасним врахуванням як результатів попереднього виконання даної команди переходу, так і результатів виконання інших команд переходу. Це реалізується формуванням адреси таблиці історії переходів

шляхом об'єднання адреси команди переходу та вмісту регістру глобальної історії. Для такого об'єднання використовується або операція конкатенації (зчеплення) (рис. 5.19в), або операція додавання за модулем 2. При цьому можуть використовуватися як всі розряди адреси команди переходу та вмісту регістру глобальної історії, так і лише деяка їх кількість.

Потрібно відзначити, що в якості таблиці історії переходів зазвичай використовується асоціативна пам'ять, що дозволяє суттєво прискорити пошук коду переходу.

Наведені схеми названі однорівневими, оскільки в них задіяно один рівень таблиць. Такі схеми передбачення використані в наступних комп'ютерах: Alpha 21064 та 21064, A M D K5, R10000, Power PC620, UltraSPARC та інших.

Для врахування конкретних результатів переходів кожної команди переходу часто використовується таблиця локальної історії, яка складається з регістрів, у кожному з яких подібно до регістру глобальної історії фіксуються результати переходів конкретної команди. Дворівнева схема передбачення переходу, яка реалізує цей підхід, показана на рис. 5.20.

Рис. 5.20. Дворівнева схема передбачення переходу з використанням таблиці локальної історії

Для різних програм різні стратегії передбачення дають різну точність. Тому в ряді комп'ютерів застосовуються гібридні схеми передбачення переходів, коли в кожному конкретному випадку застосовується та схема передбачення, від якої очікується найвища точність передбачення. Структура такої гібридної схеми показана на рис. 5.21.

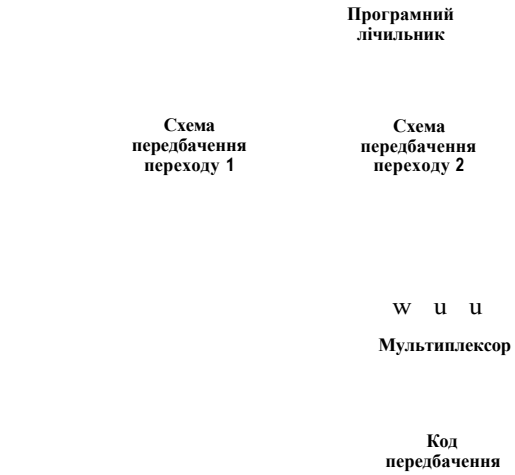


Рис. 5.21. Структура гібридної схеми передбачення переходу

Адресування конкретної схеми передбачення переходу та вибір схеми передбачення переходу здійснюються від програмного лічильника, тобто адресою команди, для якої здійснюється передбачення. Оновлення таблиць історії проводиться за раніше описаним правилом. Такі схеми передбачення є досить складними, але забезпечують найвищу точність передбачення.

5.4. Покращена структура комп'ютера Із спрощеною системою команд

На основі проведеного вище аналізу конфліктів у конвеєрі та способів їх мінімізації можна провести покращання структури комп'ютера із спрощеною системою команд. В якості прикладу на рис. 5.22 подано покращену структуру тракту виконання команд комп'ютера БЬХ.

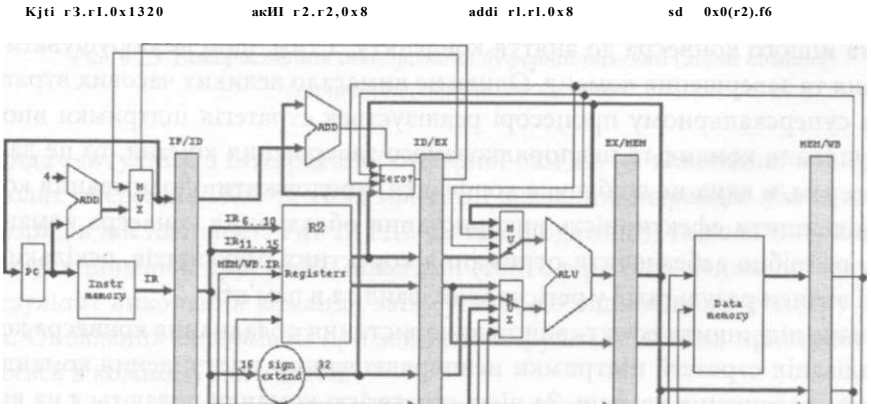


Рис. 5.22. Покращений конвеєр комп'ютера DLX

Структура додатково містить:

- суматор ADD, який прискорено, вже на сходинці ID, обраховує цільову адресу переходу;

- вузол Zero перенесено на один ярус вперед, щоб прискорити реакцію конвеєра на команду умовного переходу та синхронізувати цю реакцію з обрахуванням цільової адреси переходу. В результаті зменшено на такт затримку переходу;

- введено додаткові інформаційні шини, що дозволяють прискорено надсилати до вузлів процесора коди операндів при реалізації випередження. При цьому кількість входів мультиплексорів АЛП збільшено.

Кольорами відтінені відповідності станів вузлів конвеєра командам, які цей конвеєр опрацьовує.

5.5. *Особливості запобігання конфліктам в суперскалярних процесорах*

В суперскалярних процесорах, як ми вже бачили з їх розгляду в розділі 5, паралельно працює кілька конвеєрів команд. Тому в цих процесорах, як і в процесорах із простою системою команд, можливі раніше описані конфлікти при конвеєрному виконанні команд. Крім того, в них на додаток можливі конфлікти, які пов'язані з тим, що команди знаходяться в паралельних вітках. Тому тут ускладнюються питання запобігання раніше описаним структурним конфліктам, конфліктам за даними та конфліктам керування, і, крім того, добавляються конфлікти, пов'язані із забезпеченням впорядкованого поступлення команд на виконання та впорядкованого завершення команд.

Під впорядкованим поступленням команд та впорядкованим завершенням команд розуміють таким чином впорядковану черговість поступлення команд на виконання та черговість завершення команд, яка визначена програмою. В іншому випадку говорять про невпорядковане поступлення команд та невпорядковане завершення команд.

В перших суперскалярних процесорах типу Pentium було реалізоване впорядковане поступлення команд і впорядковане завершення команд. Такий підхід був досить простим, оскільки при виникненні конфліктів у одному з конвеєрів процесора призупинялась робота іншого конвеєра до зняття конфлікту, з тим, щоб не порушувати порядок поступлення та завершення команд. Однак це вимагало великих часових втрат.

Коли в суперскалярному процесорі реалізується стратегія підтримки впорядкованого поступлення команд та невпорядкованого завершення команд, то це дає можливість конвеєрам, в яких не відбулися конфлікти, продовжити опрацювання команд. Це дозволяє підвищити ефективність використання обладнання конвеєра команд. Однак при цьому потрібно забезпечити отримання коректних результатів, оскільки можливі некоректні записи результатів у регістровий файл та в пам'ять.

Іще більше підвищити ефективність використання обладнання конвеєра команд дозволяє реалізація стратегії підтримки невпорядкованого поступлення команд і невпорядкованого завершення команд. За цією стратегією команди подаються на виконання не в порядку їх поступлення в конвеєр, а при їх готовності до опрацювання, тобто коли наявні операнди та вільний функціональний вузол, в якому вони мають бути опрацьовані. Для виконання цієї стратегії необхідно забезпечити коректність результату вико-

нання програми. Для цього використовується спосіб, який називається перевпорядкуванням команд, або відкладенням виконання команд. Для перевпорядкування команд в конвеєр вводиться додаткова буферна пам'ять, яка називається вікном команд. До цієї пам'яті завантажуються всі команди, які пройшли декодування, та, при необхідності запобігання конфліктам команд за даними при запису результатів до регістрового файла, виконується перейменування регістрів (див. п. 5.2.8). Вікно команд забезпечує відкладення передачі команд на виконання до готовності операндів і дозволяє забезпечити потрібну черговість завершення команд.

Буферна пам'ять, яка називається вікном команд, може бути реалізована двома способами - як інтегрована та як розподілена.

Інтегрована буферна пам'ять реалізується на основі асоціативної пам'яті. Вона оперативно виявляє і подає на виконання команди, для виконання яких є всі необхідні операнди та ресурси. Для кожної команди в цій пам'яті виділяється одна комірка. В цій комірці зберігається декодована команда, яка має наступні поля: декодоване поле операції (О), поля операндів (ПО), які вміщують або самі операнди, або адреси їх знаходження, поле, яке вказує місце розміщення результату (ПР), а також поле розрядів достовірності (РД). Для аналізу доступності та виконання розподілу команд до вільних функціональних пристроїв (ФП) в процесорі використовується пристрій диспетчеризації, в якому є регістр зайнятості функціональних пристроїв процесора (рис. 5.23).

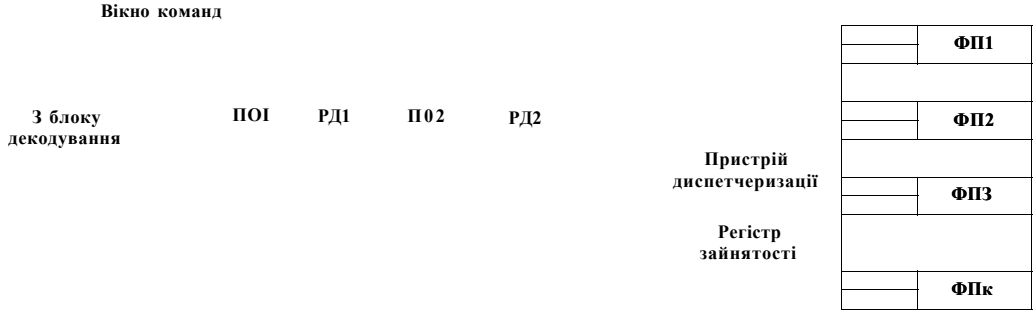


Рис. 5.23. Використання інтегрованої буферної пам'яті (вікна команд) для перевпорядкування команд

Команда зчитується з інтегрованої буферної пам'яті на виконання лише після того, коли в полях операндів ПОІ та П02 присутні значення операндів або їх адрес (тобто в полі розрядів достовірності Рді та РД2 записані одиниці), та коли потрібні для її виконання функціональні пристрої є вільними (тобто в регістрі зайнятості записані одиниці). Результат виконання команди записується до відповідного регістру регістрового файла. Оновлення інформації про зайнятість функціональних пристроїв процесора здійснюється в кожному його циклі.

Якщо вікно команд реалізується як розподілена буферна пам'ять, то на вході кожного функціонального блоку розміщується буфер декодованих команд, який називається блоком резервування (БР). Після вибірки та декодування команди поступають до того блоку резервування, в якому вони будуть виконуватися (рис. 5.24). Робота кожного

блоку резервування є аналогічною роботі інтегрованої буферної пам'яті, тобто команда поступає на виконання при готовності операндів та незайнятості функціонального пристрою.

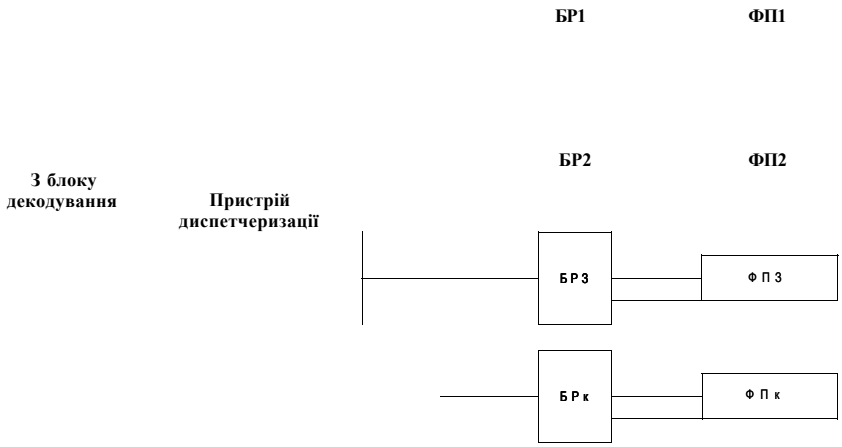


Рис. 5.24. Використання розподіленої буферної пам'яті (вікна команд) для перевпорядкування команд

Оскільки метод резервування орієнтований на подання на опрацювання одночасно декількох команд, використання розподіленої буферної пам'яті (вікна команд) для забезпечення перевпорядкуванням команд є значно простішим порівняно з використанням багатопортової інтегрованої буферної пам'яті. Подібну розподілену буферну пам'ять вперше запропонував Р. Томасуло в комп'ютерній системі IBM360/91 у 1967 році, тому описаний метод резервування носить його ім'я.

Важливим питанням для забезпечення виконання стратегії невпорядкованого поступлення команд та невпорядкованого завершення команд є підтримання правильної послідовності виконання команд при декількох паралельно працюючих функціональних пристроях. Вирішення цього питання було знайдене в 1988 році Смітом та Плескуном, які запропонували для цього використати буфер відновлення послідовності. До цього буфера команди записуються в порядку, який відповідає заданому програмою порядку їх зчитування, а зчитування команди з буфера дозволено тільки після завершення її виконання та коли всі попередні команди вже зчитані з буфера.

5.6. *Комп'ютери з довгим форматом команди*

Раніше розглянуті суперскалярні процесори характеризуються високою продуктивністю при забезпеченні безконфліктного виконання команд. Але це вимагає значних додаткових затрат обладнання. На рис. 5. 25 показано кристал суперскалярного процесора MIPS R10000, на якому позначені вузли процесора та виділено апаратні засоби пристрою керування, які забезпечують невпорядковане виконання команд. Видно, що ці засоби зайняли більше ЗО відсотків площі кристала.

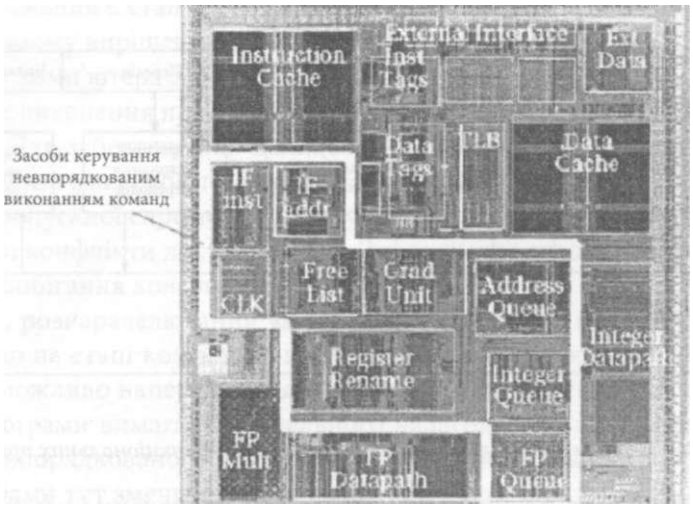


Рис. 5.25. Кристал суперскалярного процесора MIPS R10000, на якому позначені вузли процесора та виділено засоби керування невпорядкованим виконанням команд

Розглянемо далі архітектури комп'ютерів, у яких відсутні конфлікти команд. До них, зокрема, належать комп'ютери з довгим форматом команди

Архітектура комп'ютерів з довгим форматом команди (КДФК), англійський еквівалент VLIW (Very Long Instruction Word), бере свій початок від паралельного мікрокоду, що застосовувався в комп ютерах CDC6600 і IBM 360/91. У 70-х роках багато комп'ютерних систем оснащувалися додатковими векторними процесорами обробки сигналів, що використовували довгий формат команди. Зокрема, до таких процесорів належали процесори AP-120B, AP-190L та інші фірми FPS.

Першими універсальними комп'ютерами з архітектурою КДФК стали міні-суперкомп ютери, випущені на початку 1980-х років компаніями MultiFlow, Culler і Cydrome, але вони не мали комерційного успіху. Наприклад, комп ютер компанії MultiFlow 7/300 використовував два арифметико-логічні пристрої для цілих чисел, два арифметико-логічні пристрої для чисел з рухомою комою і блок логічного галуження. Його 256-розрядний довгий формат команди містить поля для восьми 32-розрядних команд. В цих комп'ютерах були використані планувальник обчислень і програмна конвеєризація, які є основою технології компілятора КДФК

Архітектура КДФК передбачає наявність багатьох незалежних функціональних пристроїв. Для забезпечення паралельного виконання декілька команд пакуються в один пакет, який в подальшому будемо називати в язанкою команд, та поступають на виконання. В'язанка команд може включати, наприклад, команди над числами з фіксованою та рухомою комою, команду звернення до пам яті та команду переходу, як це показано на рис. 5.26. Така в'язанка команд матиме набір полів команд для кожного функціонального пристрою, що призводить до кратного збільшення її довжини порівняно з однією командою. Як видно з рисунка, кожне з цих полів керує відповідним функціональним пристроєм, а обмін даними між пристроями здійснюється через інтегрований багатопортовий регістровий файл.

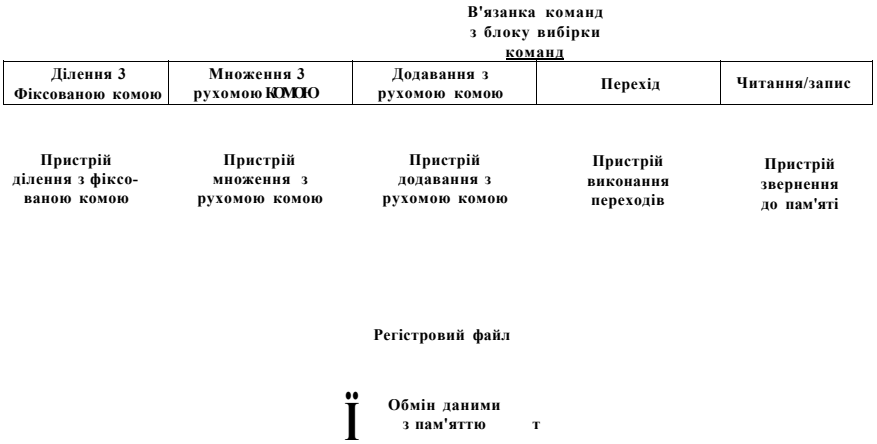


Рис. 5.26. Виконання в'язанки команд в незалежних функціональних пристроях

Завдання ефективного планування паралельного виконання команд в комп'ютері з довгим форматом команди повністю лягає на компілятор. Компілятор з послідовності команд початкової програми генерує в'язанку команд шляхом проглядання програми як в межах, так і за межами лінійних ділянок програми без галужень. Для забезпечення коректності виконання операцій компілятор має бути здатним визначати можливість появи конфліктів при виконанні команд. У певних ситуаціях, коли компілятор не може зробити заключення, що конфлікт не відбудеться, наприклад, у випадку звернення до масиву, коли індекс обчислюється під час виконання програми, операції не можуть плануватися для паралельного виконання. Відповідно це призводить до зниження продуктивності.

Компілятор визначає ділянку програми без циклів, яка стає кандидатом для формування в'язанки команд. Для збільшення розміру тіла циклу широко використовується методика розкручування циклів, що призводить до утворення великих фрагментів програми, що не містять зворотних дуг.

Якщо виконанню підлягає програма, що має тільки переходи вперед, компілятор робить евристичний прогноз вибору умовних гілок. Шлях, що має найбільшу вірогідність виконання (його називають трасою), використовується для оптимізації, що проводиться з врахуванням конфліктів за даними між командами і обмежень за апаратними ресурсами. Під час планування генерується в'язанка команд. Всі команди, які входять до в'язанки команд, видаються одночасно і виконуються паралельно. Після обробки першої траси планується наступний шлях, що має найбільшу вірогідність виконання (попередня траса більше не розглядається). Процес пакування команд послідовної програми у в'язанки команд продовжується до тих пір, поки не буде оптимізована вся програма.

Ключовою умовою досягнення ефективної роботи комп'ютера з довгим форматом команди є коректний прогноз вибору умовних гілок. Відмічено, наприклад, що прогноз умовних гілок для наукових програм часто виявляється точним. Повернення назад є у всіх ітераціях циклу, за винятком останньої. Таким чином, прогноз буде коректний в більшості випадків. Інші умовні гілки, наприклад, гілка обробки переповнення і перевірки граничних умов (вихід за межі масиву), також є надійно передбачуваними.

З погляду архітектурних ідей комп'ютер з довгим форматом команди можна розглядати як розширення архітектури комп'ютера з простою системою команд, оскільки тут

також довжина команди є сталою, апаратні ресурси плануються статично, та віддається перевага програмному вирішенню конфліктів.

В архітектурі комп'ютера з довгим форматом команди, крім того, принцип заміни керування під час виконання програми на планування під час компіляції поширений і на систему пам яті. Для забезпечення зайнятості конвеєрних функціональних пристроїв тут необхідна пам ять з високою пропускною спроможністю. Одним із сучасних підходів до збільшення пропускної спроможності пам яті є використання розшарування пам'яті, причому можливі конфлікти доступу до пам'яті визначає спеціальний модуль компілятора модуль запобігання конфліктам

Таким чином, розпаралелювання у комп'ютерах з довгим форматом команди виконується виключно на етапі компіляції під час формування в язанок команд, тобто статично. Так як неможливо наперед передбачити розв'язання усіх видів залежностей, відкомпільовані програми вимагають ретельного налагоджування. Негативно впливають також ефекти невпорядкованого завершення виконання команд. Саме тому надійність виконання програми тут зменшена порівняно з суперскалярним варіантом архітектури комп'ютера. Тобто, цей підхід дозволяє досягти максимуму продуктивності, але є прийнятним лише у певних застосуваннях комп'ютерних засобів. Перевага КДФК в потенційній продуктивності найбільш відчутна в серверних задачах, де паралельно опрацьовують декілька процесів (ниток), в наукових задачах, задачах тривимірної візуалізації та в задачах обробки сигналів (де, зокрема, застосовуються процесори ADSP21XX фірми Analog Devices та TMS320C6X фірми Texas Instruments, які належать до вказаної архітектури).

Архітектуру КДФК запроваджено у новітніх процесорах Alpha фірми DEC та IA-64 фірми Intel. Останній процесор оптимізовано для виконання серверних задач Як приклад на рис. 5.27 приведено структуру ядра процесора TMS320C6X фірми Texas Instruments, в якому використано довгий формат команди. В'язанка команд цього процесора складається з восьми 32-розрядних команд. Тут використано два тракти обробки даних A та B, кожний з яких має по 4 функціональні блоки та свій багатопортовий регістровий файл.

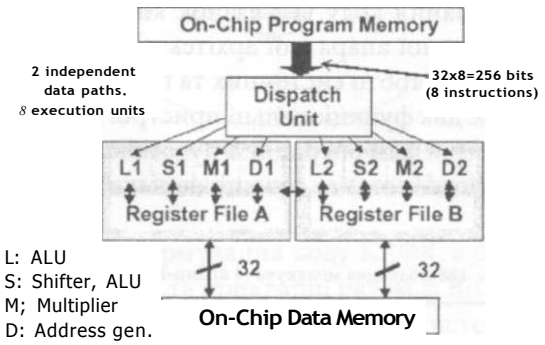


Рис. 5.27. Структура ядра процесора TMS320C6X фірми Texas Instruments

Тракти обробки даних A і B є однотипними та мають в своєму складі наступні функціональні пристрої: L - АЛП, S - пристрій зсуву та АЛП, M - перемножувач, D - формувач адрес пам'яті. В'язанка команд зчитується з пам'яті команд (On-Chip Program Memory) та поступає до блоку диспетчеризації (Dispatch Unit), який здійснює розподіл команд у відповідні функціональні пристрої.

5.7. *Комп'ютери з комбінованою архітектурою*

При реалізації архітектури КДФК виникають серйозні проблеми. Компілятор цього комп'ютера повинен в деталях враховувати внутрішні особливості процесора, аж до організації роботи його функціональних пристроїв. Як наслідок, при випуску нової версії комп'ютера з більшою кількістю функціональних пристроїв доводиться радикально переписувати і компілятор. Інша проблема - це за своєю суттю статична природа оптимізації, яку забезпечує компілятор КДФК. Важко передбачити, як, наприклад, поведеться компілятор, коли зіткнеться під час компіляції з непередбаченими динамічними ситуаціями, такими як очікування введення/виведення. Тому розробники сучасних швидкодіючих комп'ютерів починають відходити від чистої архітектури КДФК

У 2000 році корпорація Трансмета випустила процесор Crusoe, сумісний із системою команд процесора x86, який характеризується високою продуктивністю та зниженою споживаною потужністю. Покращання характеристик досягнуто шляхом заміни відчутної частки апаратури процесора допоміжною, але обов'язковою, програмною оболонкою Процесор Crusoe складається з апаратного ядра та програмної оболонки, яка формує програмний код для цього ядра. Ядром є нескладний КДФК, здатний виконувати до чотирьох простих команд в кожному такті. Доповнення апаратного ядра програмною оболонкою для формування програмного коду створює ефект присутності повноцінного набору апаратних засобів архітектури x86. Програмну оболонку формування коду називають Code Morphing™ тому, що вона динамічно відбиває ("morphs") зовнішні (складні) команди процесора x86 у внутрішні (спрощені) в'язанки команд апаратного ядра

Технологія Code Morphing™ фірми Трансмета змінює традиційні методи проектування процесорів. Практична апробація того, що серійні процесори можна реалізувати як гібрид програмних і апаратних засобів, суттєво розширила можливості проектувальників по вибору та впровадженню оптимальних рішень. По перше, паралельна робота проектувальників апаратного ядра процесора та системних програмістів, які розробляють програмну оболонку, сприяє скороченню термінів розробки. По друге, зовнішня програмна оболонка формування коду із в язанок команд для ядра процесора забезпечує незалежний від прихованої апаратної архітектури розвиток програмних засобів комп'ютерної системи в цілому тобто системних та прикладних програм

Ядро процесора включає два функціональні пристрої для операцій з фіксованою комою, функціональний пристрій для операцій з рухомою комою, функціональний пристрій звернення до пам'яті (load/store) та функціональний пристрій виконання умовних переходів (рис. 5.28).

Рис. 5.28. Загальна структура ядра процесора Crusoe

Тут ADD - операція додавання з фіксованою комою, FADD - операцією додавання з рухомою комою, LD - завантаження подвійного слова, а BRCC - умовний перехід за деякою умовою (Conditional Code Branch).

В'язанка команд процесора Crusoe отримала назву молекули. Вона має довжину 64/128 бітів та вміщує чотири команди (чотири атоми). Усі атоми в межах молекули виконуються паралельно. Формат молекули прямо визначає зв'язок атома із конкретним функціональним пристроєм, що значно і спрощує, і прискорює апаратні засоби. Самі молекули процесор опрацьовує із збереженням черговості, тобто без використання складних апаратних засобів невпорядкованого опрацювання. Для забезпечення ефективного використання процесора мінімізується кількість незаповнених атомів, а у випадках несумісності атомів за паралельністю, на місце відсутнього атома вставляється пуста операція.

Програма Code Morphing є фундаментальною системою динамічної трансляції, що транслює команди однієї архітектури (в даному випадку КССК процесора x86) в команди іншої архітектури (КДФК). Програму Code Morphing розміщено в ПЗП процесора і вона є першою програмою, що виконується при завантаженні процесора. Рис. 5.29 ілюструє зв'язок між кодом процесора x86, програмою Code Morphing та ядром процесора Crusoe.

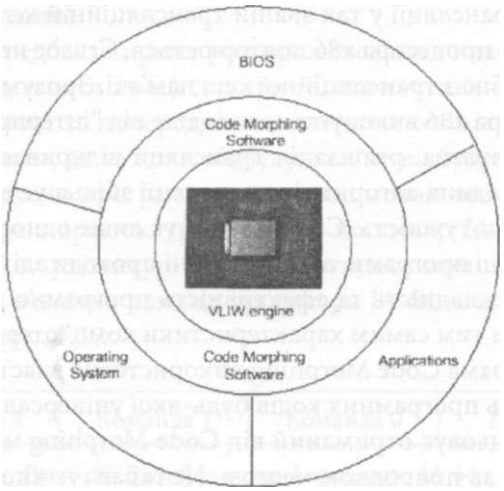


Рис. 5.29. КДФК (VLIW engine), Code Morphing, BIOS, операційна система та прикладні програми в кодах процесора x86

Завдяки тому, що програма формування коду КДФК, а саме, Code Morphing, ізолює програми процесора x86 (системні та прикладні разом із BIOS і операційною системою) від апаратури процесора Crusoe із притаманною йому системою команд КДФК, цю систему команд можна змінювати без жодного впливу на систему команд процесора x86. Але при зміні архітектури КДФК певних змін зазнає і програма Code Morphing. Проте зміни до програми формування коду Code Morphing треба виконувати лише один раз на кожну зміну архітектури ядра.

Приховування реальної архітектури рівня машинних команд за шаром фірмової програми Code Morphing дозволяє уникнути проблем, притаманних чистим КДФК. Чисті КДФК змушують розробника компілятора враховувати усі деталі роботи конвеєра. При

цьому найменші зміни у конвеєрі вимагають перепроектування компілятора (не звичайний, а оптимізуючий), що є надто складною ресурсомісткою та кропіткою працею. Зазначеної проблеми для процесора Crusoe просто не існує. З погляду системних програм Crusoe виглядає як стандартний процесор x86.

Описана технологія віртуалізації архітектури комп'ютера зі складною системою команд є принципово новим і ефективним методом декомпозиції та вирішення складних завдань проектування сучасних комп'ютерів. Запропонований фірмою Трансмета програмно-апаратний метод проектування забезпечив вдалий розподіл функцій процесора між його програмною та апаратною частинами, і, тим самим, дозволив знайти компроміс між вартістю та складністю, між продуктивністю та споживаною потужністю.

Програма Code Morphing формування коду КДФК транслює групи команд процесора x86, а не кожну відокремлену команду, як у суперскалярному процесорі. Зрозуміло, що опрацювання групи команд розширює діапазон дій програмного транслятора, що дає йому можливість враховувати семантику фрагмента коду, разом із притаманною фрагментові кореляцією поміж сусідніми командами. Саме це обумовлює перевагу трансляційної технологій Crusoe. Більше того, суперскалярний процесор транслює команду кожного разу, як її виконує. Crusoe виконує трансляцію команди одноразово, зберігаючи результат трансляції у так званій трансляційній кеш пам'яті. Коли виконання фрагмента програми процесора x86 повторюється, Crusoe не виконує повторну трансляцію, а забирає потрібне з трансляційної кеш пам'яті. Зрозуміло, що "компільований" фрагмент коду процесора x86 виконується швидше від "інтерпретованого".

Програмна, а не апаратна, реалізація трансляції відкриває нові можливості. Адже апаратна реалізація складних алгоритмів трансляції збільшує кількість транзисторів на кристалі та споживану потужність. Crusoe виконує лише одноразову повільну трансляцію при першому проході програми, а всі наступні проходи здійснює швидко, що дозволяє значно підвищити складність та ефективність програмно реалізованих алгоритмів трансляції, підвищивши тим самим характеристики комп'ютерної системи.

Зрозуміло, що програма Code Morphing використовує властивість локалізації (locality of reference) вибирань програмних кодів будь-якої універсальної машини. Апаратура процесора Crusoe опрацьовує отриманий від Code Morphing молекулярний код не хаотично, а впорядковано, за природною чергою. Це гарантує впорядковане опрацювання оригінальних команд процесора x86. Саме молекули однозначно визначають паралелізм рівня машинних команд, тому апаратна реалізація процедури паралельного опрацювання спрощується.

5.8. Комп'ютери з явним паралелізмом виконання команд

Як ми вже побачили з матеріалу даного розділу, планування порядку обчислень є досить важким завданням, яке доводиться вирішувати при проектуванні сучасного комп'ютера. У суперскалярній архітектурі для розпізнавання залежностей між командами застосовуються досить складні апаратні рішення (наприклад, в P6 і наступних процесорах фірми Intel для цього використовується буфер перевпорядковування команд - Reorder Buffer). Проте розміри такого апаратного планувальника при збільшенні кількості функціональних пристроїв зростають в геометричній прогресії. Тому суперскалярні

проекти зупинилися на відмітці 5-6 оброблюваних за цикл команд. Навіть КДФК теж далеко не завжди можуть забезпечити повне заповненням в'язанок команд - реальне завантаження близько 6-7 команд в такті.

Вище зазначені проблеми призначена вирішувати нова архітектура із назвою EPIC (Explicitly Parallel Instruction Set Computing).

Класичний процесор, переважно, не спроможний визначити взаємовплив (залежності даних, керування і структури) поміж віддаленими командами первинного (поки що не розпаралеленого) потоку команд. Адже процесор аналізує лише той фрагмент потоку команд, що розташований безпосередньо у процесорному апаратному, тому і невеликому, буфері команд. З іншого боку, програма-компілятор виконуваного коду має значно ширше поле зору та практично необмежений час (ще до виконання програми процесором), аби наперед проаналізувати первинний програмний код на предмет виявлення вказаних вище залежностей та оптимізувати цей код під потрібну архітектуру. В цілому, тут ми маємо ситуацію, коли бажано все, що можна, підготувати заздалегідь (зрозуміло, статично, під час компіляції), а не приймати складних апаратних динамічних рішень в реальному часі (при виконанні процесором вже остаточно скомпільованої програми, із притаманними швидкими рішеннями, помилками та з відповідними часовими витратами на їх виправлення).

Архітектура EPIC передбачає пряме розпаралелювання виконання програми, тобто компілятор мусить повідомляти процесор про те, яка частина коду може виконуватися паралельно. Оптимізований за попереднім означенням компілятор EPIC аналізує програмний код, аби визначати, де та коли відбудуться/не відбудуться умовні переходи та знайти і позначити ті частини програмного коду, які можна виконувати паралельно.

Прикладом процесора з архітектурою EPIC є процесор IA-64 фірми Intel. В цьому процесорі команди, як і в архітектурі КДСК, пакуються компілятором в 128-розрядні в'язанки команд. Кожна в'язанка команд процесора IA-64 містить три команди та шаблон, як це показано на рис. 5.30.

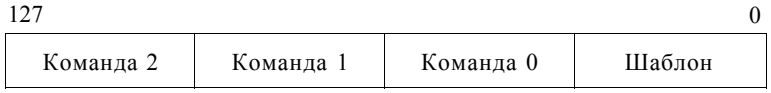


Рис. 5.30. В'язанка команд процесора IA-64

В шаблоні вказується залежність між командами в одній в'язанці та між в'язанками, тобто вона вказує, чи можна одночасно виконувати і-ту (i= 0,1,2) команду в'язанки m одночасно з j-ю (j= 0,1,2) командою в'язанки п. Кожній в'язанці в процесорі виділяється три функціональних пристрої, тобто кожній команді виділяється один пристрій. Вміст поля шаблону встановлюється або при генеруванні коду компілятором, або безпосередньо системним програмістом, що пише мовою асемблер. Процес генерації коду виконують так, аби гарантовано позбутися конфліктів типу RAW, WAW в межах командної групи.

Кожна з трьох команд в'язанки має формат, приведений на рис. 5.31.

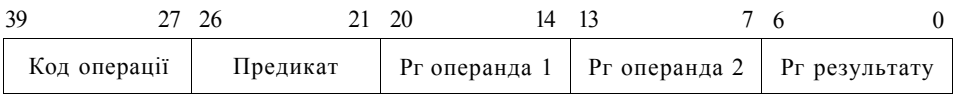


Рис. 5.31. Формат кожної з трьох команд в'язанки

До складу команди входять наступні поля: коду операції, предиката, номери регістрів двох операндів та регістра результату. Розрядності кожного поля вказані на рисунку.

Поле предиката вказує номер регістра предиката, яких в процесорі є 64. Предикація - це спосіб обробки умовних переходів. В процесорі IA-64 виконуються обидві вітки переходу. Суть способу предикації полягає в наступному. Командам різних гілок одного умовного переходу виділяються різні регістри предиката. Ці команди виконуються на функціональних пристроях процесора, а їх результати записуються до пам'яті тільки після визначення вмісту регістрів предиката, тобто після обчислення умови переходу. Вмісту регістрів вітки переходу, якій відповідає умова переходу, присвоюється значення 1, а вмісту регістрів вітки переходу, якій не відповідає умова переходу, присвоюється значення 0. Процесор перевіряє вміст регістрів предикату і записує в пам'ять результати тільки тих команд, які вказують на регістри предикатів з одиничним вмістом.

Архітектура IA-64 підвищує рівень паралельного виконання команд за рахунок того, що вона дозволяє на рівні мови асемблер прямо вказувати на паралелізм, реалізує в'язанки, кожна з яких містить три виконувані команди та містить множину надлишкових програмно-недосяжних регістрів, що дублюють операнди поточних команд, запобігаючи тим самим завчасному перезапису цих операндів.

5.9. *Короткий зміст розділу*

Розглянуті конфлікти в конвеєрі команд та методи їх усунення, оскільки вони знижують продуктивність конвеєра, яка могла б бути досягнута в ньому в ідеальному випадку. Більше того, конфлікти можуть звести нанівець всі затрати на створення конвеєра команд. Проведено аналіз методів запобігання трьом класам конфліктів: структурних, які виникають з причини браку ресурсів, коли апаратні засоби не можуть підтримувати всі можливі комбінації команд в режимі одночасного виконання з перекриттям, конфліктів за даними, що виникають у разі, коли виконання наступної команди залежить від результату виконання попередньої команди, та конфліктів керування, які виникають при конвеєризації команд передачі керування, які змінюють значення лічильника команд. Наведено приклади структур конвеєрних процесорів, у яких зменшено ймовірність виникнення конфліктів.

Розглянуто особливості запобігання конфліктам в суперскалярних процесорах, які є наступним кроком в побудові високопродуктивних процесорів. Суперскалярний процесор має кілька функціональних блоків і виконує кілька команд за один такт, тобто в такому процесорі одна команда виконується менше як за один такт. Прикладами суперскалярних процесорів є PowerPC фірми IBM, UltraSparc фірми Sun, Alpha фірми DEC. Але методи запобігання конфліктам в таких процесорах є ще складнішими, ніж у конвеєрних процесорах, що вимагає відповідного ускладнення апаратних засобів.

Були розглянуті архітектури комп'ютерів, в яких відсутні конфлікти команд, а саме: комп'ютери з довгим форматом команди, а також комбіновані архітектури, в яких поєднано архітектури КПСК та КДФК.

Проблеми забезпечення динамічного планування виконання команд привели розробників до архітектури комп'ютера з явним паралелізмом EPIC, прикладом якої стала розробка IA-64 фірми Intel. Комп'ютери цієї архітектури опрацьовують паралельно в'язанку команд, яка вказує декілька операцій, що можуть виконуватися паралельно.

Система команд цієї архітектури тісно пов'язана з будовою компілятора, оскільки планування паралельного виконання команд тут покладено на компілятор, який здійснює цю роботу перед виконанням програми в комп'ютері.

Для зменшення впливу умовних переходів на продуктивність конвеєра в процесорі IA-64 введено предикатні команди. В цьому процесорі всі команди виконуються, але результати їх виконання записуються до регістрового файла лише тоді, коли розряд предиката рівний 1. Результатом є те, що не потрібно зупиняти конвеєр до вияснення умови переходу, хоча виконується більша кількість команд.

5. 7 0. *Література для подальшого читання*

Конфлікти в конвеєрі команд та методи їх усунення розглянуті в роботах [7, 8,13, 14, 16,18-21]. В роботах [3,4,30-33] проведено аналіз методів запобігання трьох класів конфліктів: структурних, конфліктів за даними та конфліктів керування. Опис симулятора WinDLX є на web-сторінці www.

В роботах [9, 10, 22-26] розглянуто особливості запобігання конфліктам в суперскалярних процесорах. Для аналізу особливостей реалізації засобів запобігання конфліктам в суперскалярних процесорах PowerPC фірми IBM, UltraSparc фірми Sun, Alpha фірми DEC та інших доцільно пошукати їх описи на web-сторінках цих фірм. Використання розподіленої буферної пам'яті (вікна команд) для перевпорядкування команд запропоновано в роботах [1, 27].

Обмеження паралелізму рівня команд проаналізовано в [28, 29]. В роботах [3, 5, 6, 12, 15] детально розглянуті архітектури комп'ютерів, у яких відсутні конфлікти команд, а саме комп'ютерів з довгим форматом команди. Зокрема в роботі [2] розглянуті питання побудови перших процесорів обробки сигналів AP-120B фірми FPS з архітектурою КДФК. Принципи побудови компіляторів КДФК^можна знайти в [4, 11, 17].

Інформацію про комбіновані архітектури, в яких поєднано архітектури КПСК та КДФК, можна знайти в [5].

Архітектура комп'ютера з явним паралелізмом EPIC описана в *[7].*

5.11. Література до розділу 5

1. Anderson, D. W., F. J. Sparacio, and R. M. Tomasulo [1967]. "The IBM 360 Model 91: Processor philosophy and instruction handling", IBM J. Research and Development 11:1 (January), 8-24.

2. Charlesworth, A. E. [1981]. "An approach to scientific array processing: The architecture design of the AP-120B/FPS-164 family", Computer 14:12 (December), 12-30.

3. COLWELL, R. P., R. P. NIX, J. J. O'DONNELL, D. B. PAPWORTH, AND P. K. RODMAN [1987]. "A VLIW architecture for a trace scheduling compiler", Proc. Second Conf. on Architectural Support for Programming Languages and Operating Systems, IEEE/ACM (March), Palo Alto, Calif., 180-192.

4. Ellis, J. R. [1986]. Bulldog: A Compiler for VLIW Architectures, MIT Press, Cambridge, Mass.

5. FISHER, J. A. [1981]. "Trace scheduling: A technique for global microcode compaction", IEEE Trans, on Computers 30:7 (July), 478^t90.

6. FISHER, J. A. [1983]. "Very long instruction word architectures and ELI-512", Proc. Tenth Symposium on Computer Architecture (June), Stockholm, 140-150.

7. Fisher, J. A. and S. M. Freudenberger [1992]. "Predicting conditional branches from previous runs of a program", Proc. Fifth Conf. on Architectural Support for Programming Languages and Operating Systems, IEEE/ACM (October), Boston, 85-95.

8. Hwu, W.-M. and Y. Patt [1986]. "HPSm, a high performance restricted data flow architecture having minimum functionality", Proc. 13th Symposium on Computer Architecture (June), Tokyo, 297-307.

9. Johnson, M. [1990]. Superscalar Microprocessor Design, Prentice Hall, Englewood Cliffs, N.J.

10. JOUPPI, N. P. AND D. W. WALL [1989]. "Available instruction-level parallelism for superscalar and superpipelined processors", Proc. Third Conf. on Architectural Support for Programming Languages and Operating Systems, IEEE/ACM (April), Boston, 272-282.

11. Lam, M. [1988]. "Software pipelining: An effective scheduling technique for VLIW processors", SIGPLAN Conf. on Programming Language Design and Implementation, ACM (June), Atlanta, Ga., 318-328.

12. Mahlke, S. A., W. Y. Chen, W.-M. Hwu, B. R. Rau, and M. S. Schlansker [1992]. "Sentinel scheduling for VLIE and superscalar processors", Proc. Fifth Conf. on Architectural Support for Programming Languages and Operating Systems (October), Boston, IEEE/ACM, 238-247.

13. McFarling, S. [1993] "Combining branch predictors", WRL Technical Note TN-36 (June), Digital Western Research Laboratory, Palo Alto, Calif.

14. McFarling, S. and J. Hennessy [1986]. "Reducing the cost of branches", Proc. 13th Symposium on Computer Architecture (June), Tokyo, 396-403.

15. N1COLAU, A. AND J. A. Fisher [1984]. "Measuring the parallelism available for very long instruction word architectures", IEEE Trans, on Computers C-33:ll (November), 968-976.

16. Pan, S.-T, K. So, and J. T. Rameh [1992]. "Improving the accuracy of dynamic branch prediction using branch correlation", Proc. Fifth Conf. on Architectural Support for Programming Languages and Operating Systems, IEEE/ACM (October), Boston, 76-84.

17. RAU, B. R., C D. GLAESER, AND R. L. P1CARD [1982]. "Efficient code generation for horizontal architectures: Compiler techniques and architectural support", Proc. Ninth Symposium on Computer Architecture (April), 131-139.

18. Riseman, E. M. and C. C Foster [1972]. "Percolation of code to enhance parallel dispatching and execution", IEEE Trans, on Computers C-2L12 (December), 1411-1415.

19. SMITH, A. and J. LEE [1984]. "Branch prediction strategies and branch-target buffer design", Computer 17:1 (January), 6-22.

20. Smith, J. E. [1981]. "A study of branch prediction strategies", Proc. Eighth Symposium on Computer Architecture (May), Minneapolis, 135-148.

21. Smith, J. E. and A. R. Pleszkun [1988]. "Implementing precise interrupts in pipelined processors", IEEE Trans, on Computers 37:5 (May), 562-573. This paper is based on an earlier paper that appeared in Proc. 12th Symposium on Computer Architecture, June 1988.

22. Smith, M. D., M. Horowitz, and M. S. Lam [1992]. "Efficient superscalar performance through boosting", Proc. Fifth Conf. on Architectural Support for Programming Languages and Operating Systems (October), Boston, IEEE/ACM, 248-259.

23. Smith, M. D., M. Johnson, and M. A. Horowitz [1989]. "Limits on multiple instruction issue".

24. SOHI, G. S. [1990]. "Instruction issue logic for high-performance, interruptible, multiple functional unit, pipelined computers", IEEE Trans, on Computers 39:3 (March), 349-359.

25. SOHI, G. S. AND S. Vajapeyam [1989]. "Tradeoffs in instruction format design for horizontal architectures", Proc. Third Conf. on Architectural Support for Programming languages and Operating Systems, IEEE/ACM (April), Boston, 15-25.

26. THORLIN, J. F. [1967]. "Code generation for PIE (parallel instruction execution) computers", Proc. Spring Joint Computer Conf. 27.

27. TOMASULO, R. M. [1967]. "An efficient algorithm for exploiting multiple arithmetic units", IBM J. Research and Development 11:1 (January), 25-33.

28. WALL, D. W. [1991]. "Limits of instruction-level parallelism", Proc. Fourth Conf. on Architectural Support for Programming Languages and Operating Systems (April), Santa Clara, Calif, IEEE/ACM, 248-259.

29. Wall, D. W. [1993]. Limits of Instruction-Level Parallelism, Research Rep. 93/6, Western Research Laboratory, Digital Equipment Corp. (November).

30. WEISS, S. and J. E. Smith [1984]. "Instruction issue logic for pipelined supercomputers", Proc. 11th Symposium on Computer Architecture (June), Ann Arbor, Mich., 110-118.

31. WEISS, S. and J. E. SMITH [1987]. "A study of scalar compilation techniques for pipelined supercomputers", Proc. Second Conf. on Architectural Support for Programming Languages and Operating Systems (March), IEEE/ACM, Palo Alto, Calif., 105-109.

32. Yeh, T. and Y. N. Patt [1992]. "Alternative implementations of two-level adaptive branch prediction", Proc. 19th Symposium on Computer Architecture (May), Gold Coast, Australia, 124-134.

33. YEH, T. AND Y. N. Patt [1993]. "A comparison of dynamic branch predictors that use two levels of branch history", Proc. 20th Symposium on Computer Architecture (May), San Diego, 257-266.

5.12. Питання до розділу 5

1. Назвіть причини необхідності забезпечення ефективного виконання команд в процесорі.
2. Назвіть три класи конфліктів у конвеєрі команд та причини їх появи.
3. Які є дві групи структурних конфліктів?
4. Наведіть приклад структурних конфліктів, які виникають через потребу порушення тактової частоти роботи конвеєра.
5. Наведіть приклад структурних конфліктів, які виникають у зв'язку з необхідністю очікування на звільнення ресурсів комп'ютера.
6. Чому розробники допускають наявність структурних конфліктів?
7. Яка причина створення процесорів з неконвеєрними функціональними пристроями?
8. На який час потрібно призупинити роботу конвеєра команд при появі структурних конфліктів?
9. Які є способи вирішення структурних конфліктів?
10. Коли виникає конфлікт за даними?
11. Назвіть три можливі конфлікти заданими.
12. Поясніть суть конфлікту "читання після запису".
13. Поясніть суть конфлікту "запис після читання".
14. Поясніть суть конфлікту "запис після запису".
15. Які можливі конфлікти за даними?
16. Які є методи зменшення впливу залежностей між даними на роботу конвеєра команд?
17. Що дає призупинення роботи конвеєра при виявленні конфлікту за даними?
18. Що дає застосування випереджувального пересилання при виявленні конфлікту за даними?
19. Як реалізується в конвеєрі команд випереджувальне пересилання?
20. Чи завжди є можливим випереджувальне пересилання?
21. Приведіть приклади можливих та неможливих випереджувальних пересилань.
22. Що роблять, оптимізуючи компілятори, щоб не допустити конфліктів за даними?
23. Які є ознаки наявності конфліктів за даними?
24. Для яких частин програми є ефективною статична диспетчеризація послідовності команд під час компіляції?
25. Як здійснюється динамічна диспетчеризація послідовності команд у програмі під час компіляції?
26. Поясніть суть методу перейменування регістрів.
27. Які є типи конфліктів керування?
28. Назвіть способи зниження втрат на вибірку команд переходу.
29. Поясніть суть способу обчислення виконавчої адреси команди переходу в ярусі декодування команди.

30. Поясніть суть способу використання буфера адрес переходів.

31. Поясніть суть способу використання буфера команд переходів.

32. Поясніть суть способу використання буфера циклу.

33. Назвіть способи зниження втрат на виконання команд умовного переходу.

34. Поясніть суть способу введення буфера попередньої вибірки з метою зниження втрат на виконання команд умовного переходу.

35. Поясніть суть способу дублювання початкових ярусів конвеєра з метою зниження втрат на виконання команд умовного переходу.

36. Поясніть суть способу затримки переходу з метою зниження втрат на виконання команд умовного переходу.

37. Поясніть суть способу статичного передбачення переходу з метою зниження втрат на виконання команд умовного переходу.

38. Назвіть методи статичного передбачення умовного переходу.

39. Поясніть суть методу повернення, який застосовується при статичному передбаченні умовного переходу.

40. Поясніть суть методу профілювання, який застосовується при статичному передбаченні умовного переходу.

41. Поясніть суть методу статичного передбачення умовного переходу, за яким результат переходу визначається кодом операції команди переходу.

42. Поясніть суть методу статичного передбачення умовного переходу, за яким результат переходу визначається напрямом переходу.

43. Поясніть суть динамічного передбачення переходу.

44. Що таке таблиця історії переходів? Як вона реалізується?

45. Наведіть однорівневу схему передбачення переходу з формуванням адреси таблиці історії переходів в програмному лічильнику.

46. Наведіть однорівневу схему передбачення переходу з формуванням адреси таблиці історії переходів у регістрі глобальної історії.

47. Наведіть однорівневу схему передбачення переходу з комбінованим формуванням адреси таблиці історії переходів в програмному лічильнику та в регістрі глобальної історії.

48. Наведіть дворівневу схему передбачення переходу з використанням таблиці локальної історії.

49. Наведіть структуру гібридної схеми передбачення переходу.

50. Проаналізуйте тотожність та розбіжність КДФК і суперскалярної архітектур.

51. Визначте місце суперскалярних і КДФК архітектур в ієрархії сучасних комп'ютерів.

52. Визначте та поясніть основні чинники, що обмежують ефективність КДФК архітектури.

53. Наведіть основні ідеї, покладені в основу архітектури EPIC.

Розділ 6

cАморитміьІшкона^ а&ра&шдотік

Операції обробки даних ініціюються відповідними командами обробки даних. До числа цих операцій входять:

• логічні операції (логічне множення, логічне додавання, інверсія і т. д.) над розрядами слів, скалярами та векторами;

• операції зсуву (праворуч, ліворуч) над скалярами та векторами;

• операції відношення: менше, більше, рівне, менше-рівне, більше-рівне;

• арифметичні операції (додавання, віднімання, множення та ділення) над одиночними даними та векторами даних;

• операції обчислення елементарних функцій типу exp X, In X, Sin X, Cos X, arctg y/x, Sh X, Ch X, піднесення до степеня A^{m};

• операції перетворення даних (перетворення із формату з фіксованою в формат з рухомою комою і навпаки, перетворення з двійково-десяткового коду в двійковий і навпаки і т. д.);

• операції реорганізації масивів і визначення їх параметрів: сортування, пошук максимуму або мінімуму, вибір заданого масиву, зсув елементів масиву, стиск масиву;

• операції обробки символів та стрічок символів: пошук символу, зсув, заміна символів у стрічці, пакування стрічок символів, порівняння стрічок символів.

В останніх комп'ютерах у зв'язку з широким використанням засобів телекомунікацій та мультимедіа до складу основних операцій добавилися складні операції типу кодування, компресії, шифрування тощо.

В даному розділі розглянемо основні алгоритми виконання вищеназваних операцій, не вникаючи в питання їх реалізації в комп'ютері.

6.1. Логічні операції

До складу команд обробки даних входить велика кількість команд, які ініціюють логічні операції. До їх числа входять операції булевої алгебри: логічне множення, додавання, додавання по модулю два, інверсія і т. д. При цьому логічні операції можуть виконуватись над окремими розрядами слова, над одиночними даними, а також над векторами даних.

Логічні операції передбачають побітову обробку даних. Коли говорять про логічну операцію над парою слів, то мається на увазі, що проводяться окремі операції над кожною парою біт, які входять до цих слів.

Розділ 6

Операції обробки даних ініціюються відповідними командами обробки даних. До числа цих операцій входять:

• логічні операції (логічне множення, логічне додавання, інверсія і т. д.) над розрядами слів, скалярами та векторами;

• операції зсуву (праворуч, ліворуч) над скалярами та векторами;

• операції відношення: менше, більше, рівне, менше-рівне, більше-рівне;

• арифметичні операції (додавання, віднімання, множення та ділення) над одиночними даними та векторами даних;

• операції обчислення елементарних функцій типу exp X, In X, Sin X, Cos X, arctg y/x, Sh X, Ch X, піднесення до степеня A^m;

• операції перетворення даних (перетворення із формату з фіксованою в формат з рухомою комою і навпаки, перетворення з двійково-десяткового коду в двійковий і навпаки і т. д.);

• операції реорганізації масивів і визначення їх параметрів: сортування, пошук максимуму або мінімуму, вибір заданого масиву, зсув елементів масиву, стиск масиву;

• операції обробки символів та стрічок символів: пошук символу, зсув, заміна символів у стрічці, пакування стрічок символів, порівняння стрічок символів.

В останніх комп'ютерах у зв'язку з широким використанням засобів телекомунікацій та мультимедіа до складу основних операцій добавилися складні операції типу кодування, компресії, шифрування тощо.

В даному розділі розглянемо основні алгоритми виконання вищеназваних операцій, не вникаючи в питання їх реалізації в комп'ютері.

6.1. *Логічні операції*

До складу команд обробки даних входить велика кількість команд, які ініціюють логічні операції. До їх числа входять операції булевої алгебри: логічне множення, додавання, додавання по модулю два, інверсія і т. д. При цьому логічні операції можуть виконуватись над окремими розрядами слова, над одиночними даними, а також над векторами даних.

Логічні операції передбачають побітову обробку даних. Коли говорять про логічну операцію над парою слів, то мається на увазі, що проводяться окремі операції над кожною парою біт, які входять до цих слів.

6.1.1. Операція заперечення

Операція заперечення (інверсія, НЕ, NOT) є операцією над одним операндом і означає, що біти із значенням "0" набувають значення "1" і навпаки. Для відображення дії логічної операції часто використовують так звані таблиці істинності. Табл. 6.1 є таблицею істинності для операції заперечення.

Таблиця 6.1

біт операнда	біт результату
0	1
1	0

Приклади:
NOT (1000 10100010 1100) = 0111 0101 1101 0011.
NOTQ11O 1011 1010 0111) = 0001 01000101 1000.

6.1.2. Логічне І

Ця операція (загальноприйняте позначення І, AND) передбачає наявність як мінімум двох операндів, назвемо їх X та Y. Вона виконує порозрядну кон'юнкцію змінних, тобто отримання одиниці лише тоді, коли всі вхідні змінні рівні одиниці. Відобразимо значення функції наступною таблицею істинності (табл. 6.2.)

Таблиця 6.2

бітX	бітУ	біт результату
0	0	0
0	1	0
1	0	0
1	1	1

Приклади виконання операції логічного множення приведено на рис. 6.1.

```
  1101 0011 0010 1110          1100 0101 0010 1100
  0011 0010 1111 0000          1100 1101 1111 0010
= 0001 0010 0010 0000        = 1100 0101 0010 0000
```

Рис. 6.1. Приклади виконання операції логічного множення

6.1.3. Логічн" АБО

Ця операція (загальноприйняте позначення АБО, ОЯ) також передбачає наявність як мінімум двох операндів X та У. Вона виконує порозрядну диз'юнкцію змінних, тобто отримання одиниці тоді, коли хоча б одна вхідна змінна рівна одиниці. Відобразимо значення функції наступною таблицею істинності (табл. 6.3).

Таблиця 6.3

бітX	біт Y	біт результату
0	0	0
0	1	1

1	0	1
1	1	1

Приклади виконання операції логічного додавання приведено на рис. 6.2.

Рис. 6.2. Приклади виконання операції логічного додавання

6.1.4. Виключне АБО

Також цю операцію ще називають додавання за модулем два (ХОЯ), оскільки вона виконує порозрядне додавання вхідних змінних за модулем два. Ця операція виконується як мінімум над двома операндами X та У. Відобразимо значення функції наступною таблицею істинності (табл. 6.4).

Таблиця 6.4

бітХ	біт У	біт результату
0	0	0
0	1	1
1	0	1
1	1	0

Приклади виконання операції логічного додавання за модулем два наведено на рис. 6.3.

Рис. 6.3. Приклади виконання операції логічного додавання за модулем два

6.2. *Операції зсуву*

6.2.1. Логічні зсуви

При виконанні логічного зсуву праворуч або ліворуч всі розряди слова зсуваються на один розряд у відповідну сторону, в перший розряд записується нуль, а останній розряд випадає (рис. 6.4). Зсуви досить часто використовуються як складові операції при виконанні багатьох алгоритмів обробки даних. Для формату представлення даних без знаків зсув на один розряд ліворуч еквівалентний множенню на два, а на один розряд праворуч - відповідно цілочисельному діленню на два.

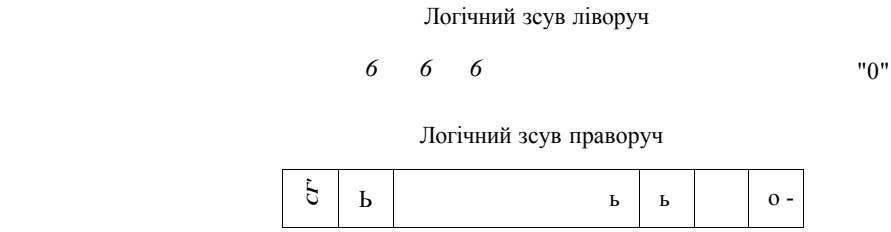


Рис. 6.4. Логічний зсув ліворуч та праворуч

Команда логічного зсуву має наступні поля: код логічної операції зсуву праворуч або ліворуч, адреса операнда та код зсуву, який вказує величину зсуву, тобто кількість розрядів, на які має бути проведений зсув. Якщо операнд позначити через X, а код зсуву через У, то результат *X* виконання операції буде рівним $X = X\ 2^{+/y}$, де знак "+" відповідає зсуву ліворуч, а знак "-" - праворуч.

6.2.2. *Арифметичні зсуви*

Арифметичні зсуви дуже подібні до попередніх (логічних), але вони мають таку особливість, як розмноження знака при зсуві праворуч та збереження знака при зсуві ліворуч (рис. 6.5).

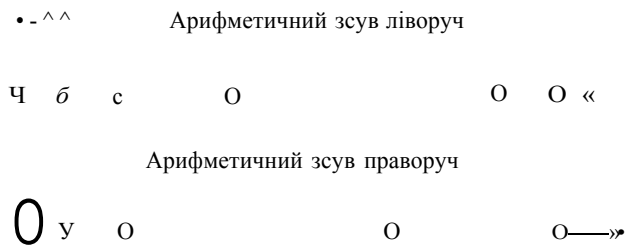


Рис. 6.5. Арифметичний зсув ліворуч та праворуч

Такі зсуви також мають зміст множення і ділення на два для формату представлення даних із знаками в оберненому та доповняльному кодах. Вони дозволяють після операції зсуву ліворуч зберегти знак представленого числа та після операції зсуву праворуч зберегти коректний результат ділення на два.

Команда арифметичного зсуву має наступні поля: код операції арифметичного зсуву праворуч або ліворуч, адреса операнда та код зсуву, який вказує величину зсуву, тобто кількість розрядів, на які має бути проведений зсув.

6.2.3. *Циклічні зсуви*

Циклічний зсув передбачає, що розряди, які витісняються з одного боку операнда, дописуються з іншого його боку (рис. 6.6).

Рис. 6.6. Циклічний зсув ліворуч та праворуч

Команда циклічного зсуву має наступні поля: код операції циклічного зсуву праворуч або ліворуч, адреса операнда та код зсуву, який вказує величину зсуву, тобто кількість розрядів, на які має бути проведений зсув.

6.3. Операції відношення

6.3.1. Порівняння двійкових кодів на збіжність

Операція порівняння двійкових кодів на збіжність дає логічний результат, рівний одиниці, якщо коди збігаються. В інших випадках значення результату рівне нулю. Виконувана функція визначається наступним булевим рівнянням:

$$Z = \bigwedge_{i=0}^{n} ((X_i \text{ AND } Y_i) \text{ OR } (\text{NOT}(X_i) \text{ AND NOT}(Y_i))),$$

де i = 0, 1 ... n - номери розрядів чисел X та Y, які порівнюються, Z - розряд результату.

6.3.2. *Визначення старшинства двійкових кодів*

Операції визначення старшинства двійкових кодів, тобто визначення, яке з двох чисел є меншим, більшим, меншим-рівним, більшим-рівним, виконуються з використанням двійкового віднімання.

Якщо це числа без знаків, то при відніманні першого числа від другого отриманий результат може бути більшим нуля, рівний нулю, або меншим нуля. Тоді, якщо отриманий результат є більшим нуля, то перше число є більшим від другого, якщо отриманий результат рівний нулю, то числа рівні, а якщо отриманий результат є меншим нуля, то друге число є більшим від першого.

Якщо це числа із однаковими знаками, то при відніманні першого числа від другого отриманий результат може бути додатнім, рівним нулю, або від'ємним. Тоді якщо отриманий результат рівний нулю то числа рівні, при від'ємному результаті перше число є меншим другого, а друге відповідно більшим, а при додатному результаті перше число є більшим, а друге відповідно меншим від першого. Якщо ж це числа із різними знаками, то більшим є додатне число.

6.4. Арифметичні операції

Команди обробки даних також ініціюють виконання арифметичних операцій. Це операції додавання, віднімання, множення та ділення над числами, представленими в форматах з фіксованою та рухомою комою. При цьому арифметичні операції також можуть виконуватись як над скалярними даними, так і над векторами.

До арифметичних належать також наступні операції над одиночними операндами:

- взяття абсолютної величини від операнда;
- інверсія знака операнда;
- прирощення операнда на одиницю;
- зменшення операнда на одиницю.

Методи реалізації арифметичних операцій в комп'ютерах вибирають з врахуванням потрібної швидкодії, формату представлення чисел, системи числення та інших техніко-економічних і системних факторів. Критерієм вибору методу зазвичай є досягнення мінімальних затрат обладнання, максимальної швидкодії або оптимального співвідношення затрат обладнання і швидкодії. Існує велика кількість алгоритмів виконання арифметичних операцій в комп'ютері. В даному розділі розглянемо лише основні алгоритми, а питання їх модифікації та реалізації будуть розглянуті в наступних розділах.

6.4.1. Додавання двійкових чисел без знаків

Додавання в двійковій системі числення відбувається подібно до звичайного додавання в десятковій системі. Додаються два розряди, які розміщені у числі на одній позиції. При виникненні переносу він передається в старший розряд і там додається. В загальному додавання можна описати наступними формулами:

$S_i = X_i \text{ XOR } y_i \text{ XOR } c_i$

$c_{i+1} = (x_i \text{ AND } y_i) \text{ OR } (x_i \text{ AND } c_i) \text{ OR } (y_i \text{ AND } c_i)$

$c_0 = 0$

де: S_i - і-тий розряд суми, x_i, y_i - і-ті розряди першого та другого доданків відповідно, c_i - розряд переносу.

Ці формули отримані з наступної таблиці істинності (табл. 6.5).

Таблиця 6.5

c_i	X_i	y_i	S_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Якщо результат, який мав би бути отриманий після додавання, є більшим, ніж може вміститися в даній розрядній сітці, то старший розряд втрачається і така ситуація називається втратою результату через переповнення.

Приклади виконання операції додавання двійкових чисел без знаків приведено на рис. 6.7.

0 1 1 0 1 0 1 1	1 0 1 1 0 0 1 1
0 1 0 0 0 0 0 1	1 1 0 1 0 1 1 0
1 0 1 0 1 1 0 0	1 0 0 0 1 0 0 1
	Переповнення

Рис. 6.7. Приклади виконання операції додавання двійкових чисел без знаків

Позначимо оператор виконання операції однорозрядного двійкового додавання відповідно до табл. 6.5 знаком суми, як це показано на рис. 6.8.

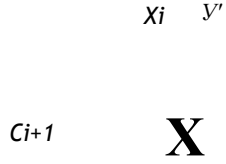


Рис. 6.8. Оператор однорозрядного двійкового додавання

Тут 8. - і-й розряд суми, х, у. - і-ті розряди першого та другого доданків відповідно, с. та с - відповідно і-й та (і+1)-й розряди переносу. Тоді граф алгоритму п-розрядного додавання буде мати вигляд, показаний на рис. 6.9. Показаний на рис. 6.9. алгоритм називається алгоритмом додавання двійкових чисел з послідовним переносом, оскільки перенос проходить через всі оператори однорозрядного двійкового додавання.

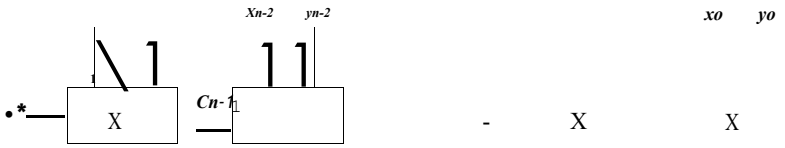


Рис. 6.9. Граф алгоритму п-розрядного додавання двійкових чисел з послідовним переносом

Вхідними даними тут є двійкові числа х ,, х _, х₀ та у , у , у₀ , а вихідним - двійкове число Б ,, $ 5₀, яке є сумою вхідних чисел. Показаний алгоритм додавання є доволі простим. Недоліком представленого алгоритму є його суттєва часова складність, тобто велика кількість операцій, які знаходяться на критичному шляху, оскільки

для отримання останнього розряду суми перенос повинен бути сформованим всіма операторами однорозрядного двійкового додавання. Для зменшення кількості операцій, які знаходяться на критичному шляху, створено ряд алгоритмів додавання, в яких скорочено кількість послідовних операцій при формуванні переносу. Це здійснюється шляхом паралельного формування переносів для всіх розрядів (так званий прискорений перенос), або паралельного формування переносів для груп розрядів (так званий частково-прискорений перенос), або використання алгоритму за методом вибору переносу.

6.4.2. *Додавання двійкових чисел із знаками*

Як ми вже бачили в попередньому розділі, існує чотири методи представлення п-розрядних чисел із знаками: в прямому, оберненому та доповняльному кодах, а також із зміщенням. В прямому коді старший розряд представляє знак числа, а наступні п-1 розряди представляють модуль числа. В оберненому та доповняльному кодах додатні числа мають те ж саме представлення, що і в прямому. Представлення ж від'ємних чисел тут є іншим. В оберненому коді від'ємні числа представляються шляхом інверсії їх розрядів, а в доповняльному коді крім того до молодшого розряду оберненого коду додається одиниця. В представленні із зміщенням всі числа, як додатні так і від'ємні, додаються до зміщення і отримані суми представляються як звичайні числа без знаків. Так від'ємне число к буде представлене як к + Ь > − 0, де Ь - зміщення. Типовим значенням зміщення вибирається число 2^{n} - 1.

Приклад: використовуючи чотирирозрядну сітку (п = 4), представимо в описаних вище кодах число к = - 3 (або в двійковій системі к = - 01Ц). В прямому коді к = 1011_2, причому старший розряд є знаковий. В оберненому коді к =1100_2 , а в доповняльному коді к = 1101_2- Для системи із зміщенням, коли зміщення Ь=2" - 1=8, маємо к = 0101_2.

Додавання чисел, представлених в прямому коді, вимагає проведення початкового аналізу знаків чисел. Якщо знаки однакові, то модулі чисел додаються, а результату присвоюється їх знак до проведення додавання. Якщо ж їх знаки різні, то модулі чисел віднімаються, а результату присвоюється знак більшого за модулем числа, або знак "+", якщо модулі чисел є рівними.

Додавання чисел, представлених в оберненому та прямому кодах, не залежить від їх знаків і проводиться так само як додавання додатних чисел в прямому коді з тією різницею, що при додаванні чисел, представлених в оберненому коді, необхідно перенос з старшого розряду подавати на вхід переносу молодшого розряду. Представлення в доповняльному коді використовується значно ширше, ніж в оберненому, оскільки при додаванні чисел тут перенос із старшого розряду просто ігнорується. Наприклад, додавши 5+(-2) в доповняльному коді маємо $0101_2 + 1110_2 = 0011_2$. Тобто отриманий правильний результат 3 при ігноруванні переносом із старшого розряду. Якщо ж додати ті ж самі числа в оберненому коді отримаємо $0101_2 + 1101_2 + 1 = 0011_2$. Тобто також отриманий правильний результат 3 при врахуванні переносу із старшого розряду. Цей перенос називається циклічним. Додавання до отриманої суми одиниці циклічного переносу не викликає повторного циклічного переносу. Наведений приклад наглядно ілюструє рис. 6.10.

Таблиця 6.6

ь.	х.	Уі	Б	
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	1

Де: 5 - і-й розряд різниці, х, у. - і-ті розряди зменшуваного та від'ємника відповідно, Ь. - розряд запозичення.

Приклади виконання операції віднімання двійкових чисел без знаків приведено на рис. 6.12.

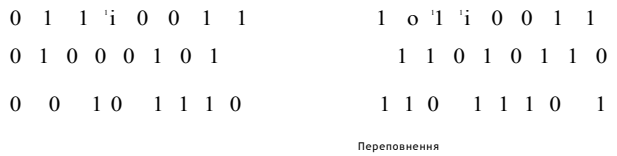


Рис. 6.12. Приклади виконання операції додавання двійкових чисел без знаків

Коли числа представлені в оберненому або доповняльному кодах, то можна використати інший метод: потрібно змінити знак від'ємника, всі його розряди потрібно інвертувати, а для доповняльного коду, крім того, збільшити від'ємник на одиницю молодшого розряду, і тоді просто додати до зменшуваного отримане число *Б = х + (ЬЮТ(у) + 1 м.р.).*

6.4.4. Множення двійкових чисел

Множення може проводитись в прямому, оберненому та доповняльному кодах. Знак результату операції множення можна визначати окремо. Для цього використовується операція ХСЖ над знаковими розрядами співмножників відповідно до табл. 6.7.

Таблиця 6.7

знак Х	знак У	Знак результату
0	0	0
0	1	1
1	0	1
1	1	0

При виконанні множення двох операндів однакової розрядності розрядність результату збільшується вдвічі, порівняно з розрядністю множників.

При виконанні множення операндів, представлених в прямому коді, їх модулі множаться як цілі двійкові числа без знаків, або як дробові числа без знаків, оскільки про-

цедура множення в обох випадках та ж сама. При виконанні множення операндів, представлених в оберненому коді, всі розряди від'ємних чисел потрібно інвертувати, а далі проводити множення так само, як над даними, представленими в прямому коді. Разом з тим, потрібно зауважити, що існують методи прямого множення операндів, представлених в обернених кодах.

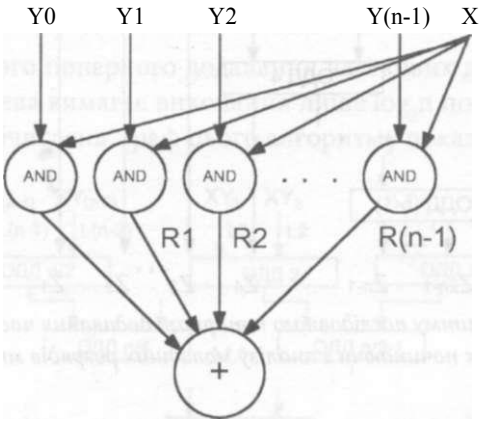
6.4.4.1. Множення цілих двійкових чисел без знаків

Якщо позначити множене буквою Х, а множник буквою У, причому представити У у вигляді суми його двійкових розрядів

то результат Ъ множення двох цілих двійкових чисел без знаків визначається з виразу:

$$Z = XY = \quad Yi2' = XY_0 2^{\circ} + XY_x 2^{1} + \ldots + \text{ЛТ}_{,\_,} 2^{1"^{x)}}$$

З наведеного виразу видно, що операція множення двійкових чисел зводиться до операції логічного множення множеного (множене - перший множник) на розряди множника та підсумовування отриманих результатів з їх зсувом на кількість розрядів, рівну відповідному показнику ступеня у виразі. Граф алгоритму множення має вигляд, показаний на рис. 6.13.



I XY

Рис. 6.13. Граф алгоритму множення

Лінійка операторів AND формує п n-розрядних результатів логічного множення множеного на розряди множника, які зсуваються праворуч на R. (i = 1, 2, п-1) розрядів. Ці результати називаються частковими добутками. Оператор із знаком додавання тут означає багатомісну операцію додавання часткових добутків.

6.4.4.2. Багатомісна операція додавання часткових добутків

Алгоритм виконання багатомісної операції додавання часткових добутків залежить від типу використовуваних операторів додавання. Це можуть бути зокрема оператори попарного п-розрядного додавання двох чисел, оператори попарного однорозрядного додавання двох чисел, оператори багатомісного паралельного додавання чисел і т. д. Якщо використовувати оператори попарного п-розрядного додавання двох чисел, тобто двох часткових добутків, то можуть бути запропоновані наступні алгоритми:

- алгоритм послідовного попарного додавання часткових добутків, отриманих починаючи з аналізу молодших розрядів множника;
- алгоритм послідовного попарного додавання часткових добутків, отриманих починаючи з аналізу старших розрядів множника;
- алгоритм паралельного попарного додавання часткових добутків з використанням структури бінарного дерева.

Граф алгоритму послідовного попарного додавання часткових добутків, отриманих починаючи з аналізу молодших розрядів множника, показаний на рис. 6.14.

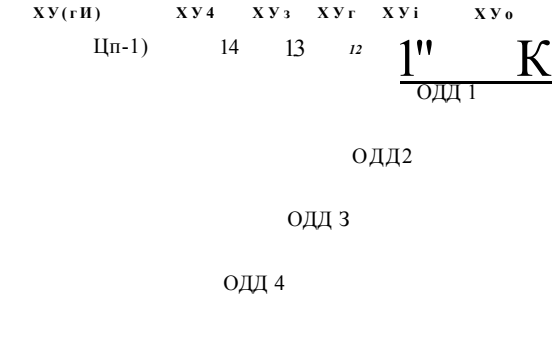


Рис. 6.14. Граф алгоритму послідовного попарного додавання часткових добутків, отриманих починаючи з аналізу молодших розрядів множника

Тут послідовно з'єднані оператори двомісного (попарного) додавання (ОДД), причому часткові добутки подаються для додавання із зміщенням на і (1^, Ц, .. Ь(п,.) розрядів ліворуч. Молодший розряд результату кожного і-го ОДД (ОДД1; ОДД,, ОДД(п) є відповідним розрядом добутку *Ь.,* а нульовий розряд добутку є рівним молодшому розряду першого часткового добутку. Розряди добутку від X_{2n} -го до *-то* отримуються з виходів (п-І)-го оператора ОДД, починаючи з другого виходу.

Подібним до розглянутого є алгоритм послідовного попарного додавання часткових добутків, отриманих починаючи з аналізу старших розрядів множника, граф якого показано на рис. 6.15.

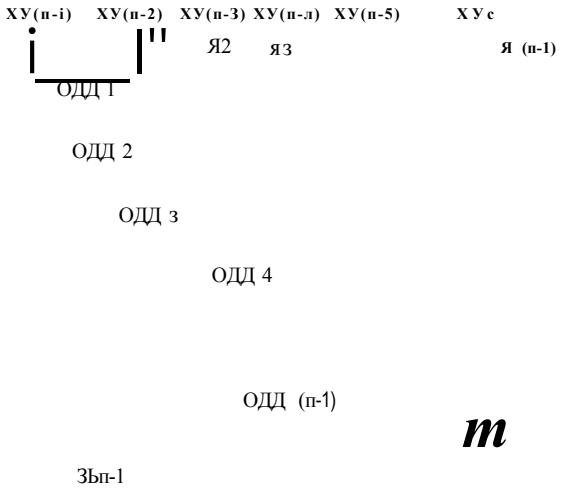


Рис. 6.15. Граф алгоритму послідовного попарного додавання часткових добутків, отриманих починаючи з аналізу старших розрядів множника

Різниця в тому, що тут часткові добутки зміщуються на і (1^, $Я_2$, ...Рч., $Я_{(п}$) розрядів праворуч, а це вимагає розширення на один біт розрядності кожного і-го ОДД в порівнянні з (і-І)-м.

Обидва розглянуті алгоритми вимагають виконання послідовно п-1 додавання, тобто послідовного виконання ОДД, причому в другому алгоритмі ОДД є обчислювально складнішими.

Алгоритм паралельного попарного додавання часткових добутків з використанням структури бінарного дерева вимагає виконання лише \ogri послідовних додавань за рахунок розпаралелення обчислень. Граф цього алгоритму показано на рис. 6.16.

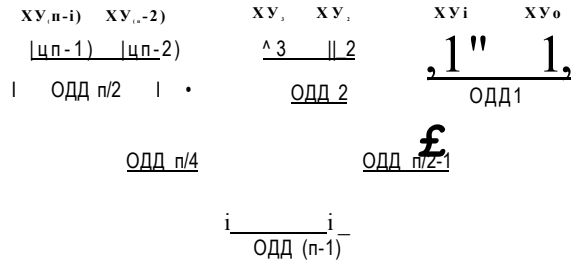


Рис. 6.16. Граф алгоритму паралельного попарного додавання часткових добутків з використанням структури бінарного дерева

При цьому на рис. 6.16 часткові множники зміщуються ліворуч, як на рис. 6.14, хоча їх можна змістити і праворуч, як на рис. 6.15.

Найчастіше в якості операторів додавання в алгоритмах виконання багатомісної операції додавання часткових добутків використовується оператор двомісного одноroз-

рядного двійкового додавання. Це дозволяє повніше врахувати особливості порозрядної обробки даних та зменшити обчислювальну складність алгоритмів, тобто кількість виконуваних операцій. Для цього випадку розроблена велика кількість алгоритмів додавання часткових добутків. Нижче розглянуті найуживаніші алгоритми. Найвідомішими серед матричних алгоритмів, які базуються на використанні операторів двомісного однорозрядного двійкового додавання, є алгоритми з горизонтальним та діагональним розповсюдженням переносу та алгоритми Дадда і Уоллеса.

Для наочності обмежимося чотирирозрядними числами. Виконання багатомісної операції додавання часткових добутків для двох чотирирозрядних чисел А та В, де *A = a{i₂ap.₀\ B = Ьфрф₀* можна представити у вигляді, показаному на рис. 6.17.

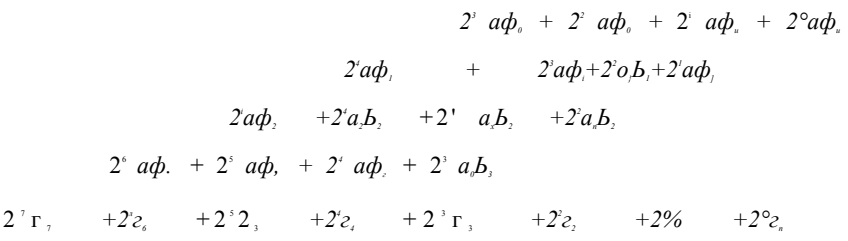


Рис. 6.17. Виконання багатомісної операції додавання часткових добутків для двох чотирирозрядних чисел

Перший рядок даного виразу представляє перший частковий добуток, другий - другий, і так далі. Окремі часткові добутки *ab,* як це було показано раніше на рис. 6.13, отримані за допомогою елементів AND. Додавання часткових добутків зводиться до додавання розрядів часткових добутків з однаковими вагами (по стовпцях) з врахуванням переносів з молодших розрядів.

На рис. 6.18 зображено граф алгоритму виконання багатомісної операції додавання часткових добутків з горизонтальним розповсюдженням переносу при використанні операторів однорозрядного двійкового додавання, який синтезований на основі графа алгоритму послідовного попарного додавання часткових добутків, отриманих починаючи з аналізу молодших розрядів множника, який показаний на рис. 6.14.

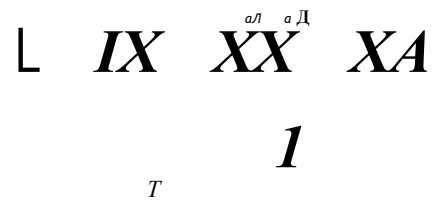


Рис. 6.18. Граф алгоритму паралельного матричного виконання багатомісної операції додавання часткових добутків з горизонтальним розповсюдженням переносу

В наведеному алгоритмі знаком суми з трьома входами позначено оператор виконання повного однорозрядного двійкового додавання, представлений на рис. 6.8, де *ab* - вхідні дані; г. - сума, а знаком суми з двома входами (три крайні зліва оператори) позначено оператор виконання неповного однорозрядного двійкового додавання, коли відсутній вхідний перенос, що спрощує алгоритм додавання. Цей оператор виконує операції відповідно до табл. 6.8. На рис. 6.18 переноси не позначено з метою спрощення сприйняття.

Таблиця 6.8

X	*У,*	*Z*	*Cu,*
0	0	0	0
0	1	1	0
i	0	1	0
i	1	0	1

Його роботу можна описати наступними формулами:

Z = x. XOR y.

c. , = x AND y.

Обчислювальна складність даного алгоритму рівна: n^2 - *n* операцій двомісного однорозрядного двійкового додавання *[*n^2 - *2n* операцій повного та *n* операцій неповного двомісного однорозрядного двійкового додавання). Кількість операторів двомісного однорозрядного двійкового додавання на критичному шляху рівна *3n - 4.*

Виконання багатомісної операції додавання часткових добутків з діагональним розповсюдженням переносу показано на рис. 6.19. Цей алгоритм також синтезований на основі графа алгоритму послідовного попарного додавання часткових добутків, отриманих починаючи з аналізу молодших розрядів множника, який показаний на рис. 6.14.

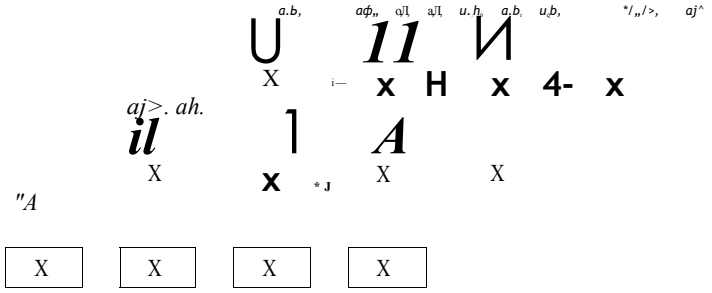


Рис. 6.19. Граф алгоритму паралельного матричного виконання багатомісної операції додавання часткових добутків з діагональним розповсюдженням переносу

Обчислювальна складність даного алгоритму рівна: n^2 - *n* операцій двомісного однорозрядного двійкового додавання *(*n^2 - *2n* операцій повного та *n* операцій неповного двомісного однорозрядного двійкового додавання). Кількість операторів двомісного однорозрядного двійкового додавання на критичному шляху рівна *2n* - 2.

Подібним чином можна побудувати алгоритми виконання багатомісної операції додавання часткових добутків з послідовним та діагональним розповсюдженням переносу

відповідно до графа алгоритму послідовного попарного додавання часткових добутків, отриманих починаючи з аналізу старших розрядів множника, показаного на рис. 6.15.

На рис. 6.20 показано граф алгоритму Дадда виконання багатомісної операції додавання часткових добутків, побудований на основі графа алгоритму паралельного попарного додавання часткових добутків з використанням структури бінарного дерева (рис. 6.16).

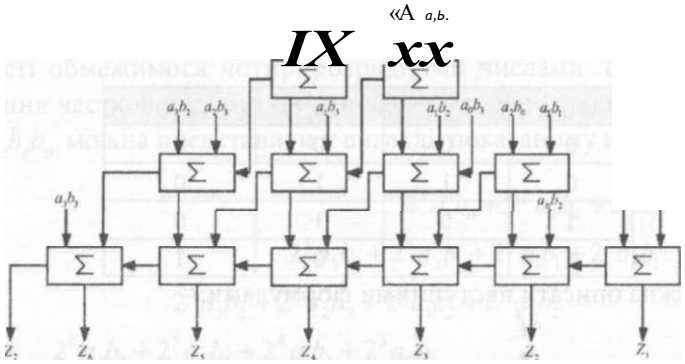


Рис. 6.20. Граф алгоритму Дадда виконання багатомісної операції додавання часткових добутків

Обчислювальна складність даного алгоритму рівна: $n^2 - n$ операцій двомісного однорозрядного двійкового додавання ($n^2 - 2n$ операцій повного та n операцій неповного двомісного однорозрядного двійкового додавання). Кількість операторів двомісного однорозрядного двійкового додавання на критичному шляху рівна $2n - 2$.

6.4.4.3. Множення двійкових чисел із знаками

При множенні чисел, представлених в оберненому коді, найпростіше перевести множене і множник в прямий код та виконати операції в цьому представленні, а при від'ємному результаті провести інвертування його значення.

Існує метод множення двійкових чисел, представлених в доповняльному коді, без перетворення в прямий код. В цьому випадку множення виконується за формулою:

$$2 = XY = X^{\wedge}(Y_{,,} \cdot_{M} - O^{2} \cdot -X\&i \sim YX'' + X(Y_{\cdot} - Y)\mathbf{2'} + \ldots + X(Y_{,\__{j}} - Y_{\__{2}})\mathbf{2}^{(''^{q})}$$

Значення розряду множника, на який здійснюється хмноження на і-му кроці, визначається з виразу:

$$\begin{cases} 0, \text{ якщо } Y_{\cdot} = Y_{\cdot+1}, \\ 1, \text{ якщо } Y'' = 0, Y_{\cdot+1} = 1, \\ -1, \text{ якщо } Y_{\cdot} = 1, Y_{\cdot+1} = 0. \end{cases}$$

При цьому добуток формується відповідно до виразу:

$$2 = XY = X(-Y_{0} + K\mathbf{(2''-2')} + Y_{,}\mathbf{(2' - 2^{2}) + \ldots + ГД2^{''''''} - 2''')} + \ldots + Y_{,,\_,}\{2^{''''^{q}} \mathbf{-2''''''''})\mathbf{)}$$

Результат множення також отримується в доповняльному коді. На основі даного виразу та використання підходу, застосованого в п. 6.4.4.2, можна побудувати графи відповідних алгоритмів множення двійкових чисел в доповняльному коді.

6.4.4.4. Прискорене множення двійкових чисел за методом Бута

Особливістю цього методу є те, що аналізуються відразу два розряди множника. Цей метод широко застосовується. Пришвидшення досягається тим, що при обрахуванні добутку за цим методом зменшується кількість додавань. Суть методу можна виразити наступними формулами для обчислення часткових добутків:

$$\backslash 2 = \begin{cases} \text{якщо } У(,_{\cdot м)}="01"; \\ \text{якщо } Ї_{ІМ} ="10"; \\ \text{якщо } 7_{(\cdot м)}="00","П"; \end{cases}$$

$2_0=0;$ **i = 0, я - 1;** $\Gamma_{и)} = 0;$ $2_п=2 = ХУ,$

де: Х, У, Ъ - множене, множник і добуток відповідно, *7л* - сума часткових добутків на і-му етапі, У(і, і-1) - і-й та і-1 розряди множника, п - кількість розрядів операндів Х та У без врахування знакового розряду.

Можна проілюструвати цей алгоритм за допомогою блок-схеми, показаної на рис. 6.21.

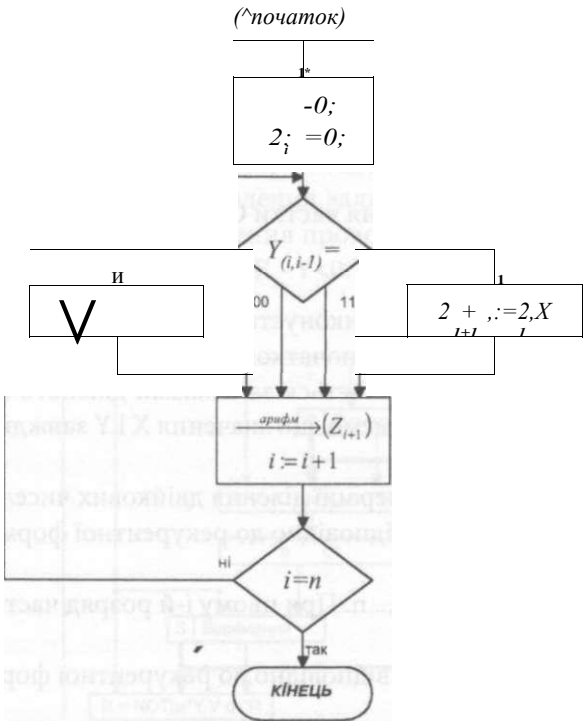


Рис. 6.21. Блок-схема множення двійкових чисел за методом Бута

Тут знаком ^{ирифм} •» позначена операція арифметичного зсуву праворуч.

Тепер розглянемо приклад:

X=0101 0101; Y=01101011.

Результати проміжних обчислень наведені в табл. 6.9.

Таблиця 6.9

i	Z.		операція	
0	0000 0000 0000 0000	0110 101110	$z =^{z}>\sim^{x}$ 2	1101 0101 1000 0000
1	1010 1011 0000 0000	0110 10|11 0	$z_{м} = h$ 2	1110 1010 1100 0000
2	1110 1010 1100 0000	0110 [10110	$= z_{i+}x$ 2	0001 1111 1110 0000
3	0001 1111 1110 0000	011101011 0	$z_{м} =^{z}{}'\sim^{x}$ 2	11100101 0111 0000
4	11100101 0111 0000	0110 1011 [0	$z_{i+i} =^{z}<^{+x}$ 2	0001 1101 0011 1000
5	0001 1101 0011 1000	0110 1011 j 0	$= z,-x$ 2	11100100 0001 1100
6	1110 0100 0001 1100	0110 101110	$z_{м} =^{\wedge}$ 2	1111 0010 0000 1110
7	1111 00100000 1110	0110 101110	$_{м} - z,+x$ 2	0010 0011 1000 0111

Таким чином

0101 0101 .01101011 =00100011 10000111.

6.4.5. *Ділення двійкових чисел*

Ділення X:Y передбачає визначення частки Q і залишку R відповідно до рівності

X = Q * Y + R,

де X - ділене, Y - дільник.

Алгоритм ділення в комп'ютерах виконується як послідовність операцій віднімання дільника Y від часткового залишку Ri, початкове значення якого рівне значенню діленого X. Знаковий розряд частки визначається за знаками діленого і дільника аналогічно операції множення (табл. 6.7). Приймемо, що значення X і Y завжди відповідають вимозі X<Y.

Існує два варіанти виконання операції ділення двійкових чисел:

- зі зсувом залишків ліворуч відповідно до рекурентної формули

$R = 2(R_{i-1} - p_i Y)$,

де $p_i = Sign (R_i - Y)$, $R_0 = X$, i=1,2,...n. При цьому і-й розряд частки визначається з виразу $q_i = NOT (p_i)$;

- зі зсувом дільника праворуч відповідно до рекурентної формули

де $R_0 = X$, i=1,2,...n, q_i - і-й розряд частки, причому $q_i = Sign R_i$.

На n-му кроці ділення в обох випадках визначається значення n-го розряду частки та залишок.

Блок-схема алгоритму ділення за першим варіантом, який має ширше використання, наведена на рис. 6.22.

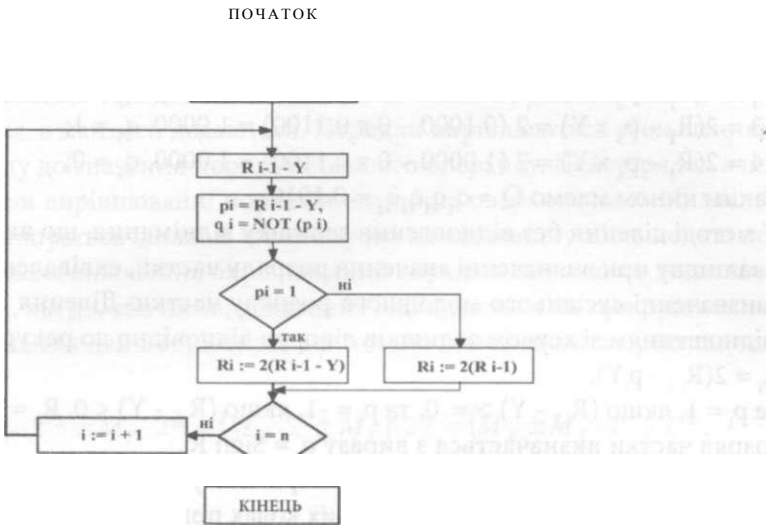


Рис. 6.22. Блок-схема алгоритму ділення

Ділення може бути виконано двома способами: з відновленням і без відновлення залишку.

При діленні з відновленням залишку послідовно віднімається дільник від діленого і проводиться аналіз значення поточного залишку. Якщо після чергового віднімання залишок позитивний, то відповідний розряд частки рівний одиниці. При від'ємному залишку розряд частки рівний нулю. В цьому випадку виконується коригуюче збільшення дільника до поточного залишку (відновлення залишку), він зсувається на один розряд ліворуч і процес повторюється. Обчислення проводяться за формулою першого варіанту, де q. = 1, якщо Я. -У>= 0 і q_i = 0, якщо К\ -У<0. На рис. 6.23 наведено два перші яруси графа, описаного алгоритму ділення.

у х

1

І Б І Віднімання

З Віднімання

І ГЧ = ЫОТр2'УУд1'Р! І

i

62У.

Розглянемо приклад. Нехай $X = 0.1000_2 = 0.5_{10}$ і $Y = 0.1100_2 = 0.75_{10}$. Тут виконана умова X<Y. Поетапні результати виконання ділення відповідно до блок-схеми алгоритму (рис. 6.22) наступні:

$R_0 = X = 0.1000;$

$R_1 = 2(R_0 - p, \times Y) = 2\ (0.1000 - O \times 0.1100) = 1.0000,\ q, = 1;$

$R2 = 2(R, - p_2 \times Y) = 2\ (1.0000 - 1 \times 0.1100) = 0.1000,\ q_2 = 0;$

$R3 = 2\ (R, - p_3 \times Y) = 2\ (0.1000 - 0 \times 0.1100) = 1.0000,\ q_3 = 1;$

$R4 = 2(R\sim - p_4 \times Y) = 2\ (1.0000 - 0 \times 0.1100) = 1.0000,\ q, = 0;$

Таким чином маємо Q = q^^q, = 0.1010.

У методі ділення без відновлення залишку віднімання, що викликало появу від'ємного залишку при визначенні значення розряду частки, еквівалентне двом відніманням при визначенні сусіднього молодшого розряду частки. Ділення виконується за тим же співвідношенням зі зсувом залишків ліворуч відповідно до рекурентної формули

$R = 2(8,, - {}_{P1} У),$

де р = 1, якщо (R - У) > = 0, та р. = -1, якщо (R., - Y) < 0, $R_0 = X$, i=1,2,...п. При цьому і-й розряд частки визначається з виразу q. = Sign R..

Ділення чисел, які представлені в оберненому та доповняльному кодах, можна виконувати за правилами ділення в прямих кодах при відповідному перетворенні операндів в цей код. Розглянемо виконання операції ділення без відновлення залишків чисел, представлених доповняльними кодами. У цьому випадку ділення зводиться до послідовності додавань і віднімань дільника від діленого, а потім часток і часткових залишків і їх зсуву на розряд ліворуч за наступними виразами:

R. = 2R,, - У, при Sign **R.** = Sign Y;

ft, = 2**r**!_, + Y, при Sign R. =/ Sign Y,

для i=l,2,...,n+1 при $2R_0 = X$ та Sign R_Q = Sign X. Тут Sign R. - знак і-го часткового залишку, Sign Y, Sign X - відповідно знаки Y та X.

При цьому розряди частки визначаються наступним чином:

q = 1, якщо Sign R. = Sign Y,

q. = 0, якщо Sign R =/ Sign Y.

Операція виконується за n+1 кроків і дає значення частки в доповняльному коді.

6.4.6. *Арифметичні операції над двійковими числами у форматі з рухомою комою*

Як вже було показано в попередньому розділі, формат з рухомою комою передбачає наявність двох частин числа - порядку (Р) та мантиси (М). Числа X та Y, які мають відповідно мантиси M_x та M і порядки P_x та P , можна представити у вигляді $X = M_x\ 2^{Px}$, $Y = M_y\ 2^{Py}$.

Для забезпечення однозначного і максимально точного представлення чисел прийнято представляти число з рухомою комою в так званому нормалізованому вигляді. Якщо виконується нерівність 1< |M| < 2 (старший двійковий розряд мантиси дорівнює 1), то відповідно до стандарту IEEE-754 вважається, що число представлене в нормалізованому вигляді (в багатьох попередніх комп'ютерах нормалізованим вважалося число, коли 0.5 < |M| < 1). Таким чином, у двійкового нормалізованого числа у форматі з рухомою комою мантиса є двійковим числом, в якому перший розряд рівний одиниці, після якої розміщено дробове число (або лише дробове число, в старшому розряді якого

завжди стоїть 1). Потрібно зауважити, що ці твердження відносяться до двійкових чисел у форматі з рухомою комою, основою порядку яких є число 2. Якщо основою порядку є числа 4, 8, 16 і т. д., то в цьому випадку число є нормалізованим, коли ненульовим є його перший розряд в четвірковій, вісімковій або шістнадцятковій системі числення відповідно. Операція приведення числа до нормалізованого вигляду називається нормалізацією.

При виконанні додавання та віднімання двійкових чисел з рухомою комою порядки операндів вирівнюються, а мантиси додаються. Порядки вирівнюються збільшенням порядку меншого операнду до значення порядку більшого операнду. Цей порядок є порядком результату. Щоб при вирівнюванні порядків величина операнда не змінилася, його мантиса одночасно зменшується шляхом зсуву вправо на відповідну збільшенню порядку кількість розрядів. Після виконання вирівнювання порядків виникають додаткові молодші розряди мантиси, які до, або після, додавання відкидаються чи заокруглюються.

Додавання та віднімання двійкових чисел з рухомою комою можна представити виразом:

$$5 = X \pm У = M_x \cdot 2^{P_x} \pm M_{\circ} \cdot 2^{P_y} = \{м_x \text{ -}2' \pm My\ )\cdot 2^{P_c} = [м_x \pm M_y \cdot 2'''){-}2^{P}\backslash$$

Наведемо приклад:

M_x= 1.000 1000; P_x=0110; M_y=1.110 1001; P_y=0100.

Оскільки перший операнд більший, порядок числа У приводиться до порядку числа X: M_y=0.011 1010; P_y=0110.

Далі мантиси додаються:

$M_s = M_x + M_y$ = 1.000 1000+0.011 1010=10.0000010; P_s=0110.

Тепер проводиться нормалізація: M_s=1.000 0001; P_s=0111.

Множення чисел у форматі з рухомою комою описується наступним співвідношенням:

$$\boldsymbol{B = X \cdot У = M_x \cdot 2^{P_x} \cdot My \cdot ї''\backslash = M_x \cdot My \cdot 2^{P_{x+P_c}}}$$

При виконанні множення порядки операндів додаються, а мантиси перемножуються. Після перемножування мантис виникають додаткові молодші розряди мантиси, які відкидаються або заокруглюються.

Розглянемо приклад:

M_x= 1.000 1000; P_x=0110; M_y=1.110 1001; P_y=0100.

Тоді $M_{\circ}$= 1.000 1000- 1.110 1001=0.111 1011 1100 1000; $P_{\circ}$=0110+0100=1010.

Після нормалізації: $M_{\circ}$= 1.111 0111; $P_{„}$= 1001.

Ділення чисел у форматі з рухомою комою описується наступним співвідношенням:

$$- \quad \frac{X}{У} \quad \frac{M_y \cdot 2^{P_x}}{My 2^{P_c}} \quad \frac{M_x}{M_y} \quad {}^{\wedge}p_{\__P}$$

При виконанні ділення порядки операндів віднімаються, а мантиси діляться.

Розглянемо приклад:

M_x=1.111 1100; P_x=0111; M_y=1.100 0000; P_y=0011.

Тоді $M_{\circ}$=1.111 1100/ 1.100 0000=0.10101; $P_{\circ}$=0111-0011=0100.

Після нормалізації: M_x=1.010 1000; $P_{\circ}$=0101.

6.5. *Операції обчислення елементарних функцій*

Значна частина задач обробки даних розв'язується з використанням операцій обчислення тригонометричних і гіперболічних функцій, функцій типу exp **X, III X, X**² й інших, операцій повороту вектора і перетворення координат з однієї системи в іншу, наприклад, з декартової в полярну. При цьому в деяких областях, наприклад, в радіонавігації, сейсморозвідці, гідроакустиці, вказані операції займають основну частина обчислень. Вирішення таких задач вимагає значних витрат часу Тому спочатку в спеціалізованих, а останнім часом і в універсальних комп'ютерах, до складу системи команд вводять операції обчислення елементарних функцій

В процесі розробки комп'ютерів та математичних методів обчислень було створено велику кількість методів обчислення елементарних функцій. Найчастіше в універсальних комп'ютерах використовується обчислення елементарних функцій за допомогою різного виду апроксимуючих виразів та аналітичних ітераційних методів

6.5.ї. *Розклад функції в ряд та використання ітеративних обчислень*

Для обчислення елементарних функцій часто використовується їх розклад в нескінченний ряд, який містить лише елементарні арифметичні та логічні операції, або використовуються ітеративні співвідношення. Кількість членів ряду чи ітерацій вибирається залежно від необхідної точності, яка до того ж обмежується розрядною сіткою.

Для прикладу можна навести обчислення квадратного кореня за допомогою ітеративної формули Ньютона

$$A_{и} = (MAлA)/2,$$

де: А. - і-те наближення. A_0 можна вибрати довільним, N - число, з якого треба добути корінь.

Щоб перевірити правильність попробуємо добути корінь з 36:

Ац = \ ***(приймаємо випадкове число)***

$$A, = +i)/2 = 18.5$$

$$^{Ai-}(^{3}\%8\ 5^{+}\overline{18}\_5\Big)\Big/^{2} = Ю.2229$$

$$A_{j} = (Зб^{}_{222д}+10.2229)/2 = 6.8722$$

$$A, = (iy_{и8\ ?\ 2\ 2}+ 6.8722)/2 = 6.0553$$

$$^{Ai\ -}(^{3}7\ б\ \ 0553^{+6}-^{0553})/^{2-6}-^{0002}$$

$$A_{6} = (з^{}/_{0002}+6.0002)/2 = 6.0000$$

6.5.2. *Обчислення елементарних функцій методом "цифра за цифрою"*

Протягом останніх десятиліть багато робіт було присвячено ефективному ітераційному алгоритму обчислення елементарних функцій, який найчастіше називають методом "цифра за цифрою". Обчислення елементарних функцій методом "цифра за цифрою" зводиться до виконання двох етапів. На першому етапі аргумент представляється або у вигляді суми п доданків 1п(1+£ а'), або £ *гхсЩ* а', або £ аггЬ а', або в вигляді добутку п співмножників (1- £ а'). Тут і - номер ітерації. На цьому етапі визначаються значення £,

які можуть бути рівні 0,1 або +1, -1. У першому випадку говорять про ітераційний процес із знакопостійними приростами, в другому - із знакозмінними. На другому етапі, на підставі знайдених на першому етапі значень, визначається величина елементарної функції шляхом додавання або множення констант. Одночасне виконання двох описаних етапів методу "цифра за цифрою" називають методом Волдера, а послідовне - методом Меджіта. Відміна методів Волдера і Меджіта, що полягає у величинах ітераційних кроків, значеннях констант і послідовності виконання етапів, істотним чином впливає на точність, швидкодію і структуру операційного пристрою. Детальний аналіз показав, що метод Меджіта має вищу точність, ніж метод Волдера, проте швидкодія його реалізації нижча.

З урахуванням отриманих Вальтером результатів по уніфікації методу і шляхом об'єднання одержаних для різних функцій алгоритмів, єдиний обчислювальний алгоритм "цифра за цифрою" в двійковій позиційній системі числення можна представити в наступному вигляді:

$X_{i+1} = X - p f . Y . 2^{M}$

$y_{i+i} = y + i x . г - ^{1} -'$

$z_{i+1} = z - f . c$

$f/ = -RSignY. + (l-R)SignZ.$

$C =](l+p)/2[arctg2^{M} + (l+|p|)arth2^{11} +](1-p)/2[\ 2^{M}\ ;$

i = 0, 1,2, 3, 3, 4,...,п-1.

Тут]h[- ціла частина від h; |p| - модуль р; f. - двійкові оператори, що приймають значення +1 або -1 залежно від знаку У. або Z, і - номер ітерації, причому, ітерації З, 12,... к, 3(к-1),... повторюються двічі. Параметр р визначає тип обчислюваних функцій: р = 0 - лінійні, р = 1 - тригонометричні, р= -1 - гіперболічні; R - параметр, що визначає від чого залежить значення {.: якщо R = 0, то f. = sign Z., якщо R = 1, то f = signY.; С - константи. В табл. 6.10 представлені функціональні можливості розглянутого алгоритму при різних значеннях параметрів р, R і різних початкових умовах X , Y , Z . Тут К. і К, - коефіцієнти деформації відповідно тригонометричного і гіперболічного векторів, рівні: К = (П (1+2~$^{2(i+1)}$))$^{1/2}$; Кг = (П (1-2$^{2(i+1)}$))$^{1/2}$, де знаком П позначено добуток чисел, взятих в дужки, при і = 0, 1, 2 .., п-1.

Таблиця 6.10

$x_{,,}$	$x_{,,}$		p	R	X	Y	z
X	V 1	7	1	0	K(XCosZ - YSinZ)	K(YCosZ + XSinZ)	0
X	Y	z	-1	0	K(XChZ - YShZ)	K(YChZ + XSZ)	0
l/K	l/K	z	-1	0	ExpZ	ExpZ	0
X	Y	z	0	0	X	X+YZ	0
l/K	0	z	1	0	CosZ	SinZ	• 0
l/K	0	z	-1	0	ChZ	ShZ	0
l/K	-1/K	z	-1	0	ExpZ	-Exp(-Z)	0
X	Y	z	0	1	X	0	Y/X+Z
X	Y	0	1	1	$K(X^2+Y^2)^{"2}$	0	ArctgY/X
X	Y	z	-1	1	$K(X^2+Y^2)^{"2}$	0	ArthY/X+Z
X-1	X-1	0	-1	1	$2K(X)^{"2}$	0	*VAgX*

Алгоритми обчислення окремих елементарних функцій методом "цифра за цифрою" мають меншу обчислювальну складність порівняно з уніфікованим алгоритмом, тому часто їх використання є доцільнішим.

6.5.3. *Табличний метод обчислення елементарних функцій*

Ідея цього методу доволі проста: формується таблиця з наперед обчисленими значеннями функції для всіх значень аргументу. Тоді, коли виникає необхідність обчислення якоїсь функції, відбувається звертання до таблиці і зчитування з неї результату. Таблиця може мати, наприклад, наступний вигляд (табл. 6.11):

Таблиця 6.11

Аргумент	Значення
0000 0001	1001 0111
0000 0010	0011 0100
1111 1111	0011 1011

Часто застосовують комбінований підхід, коли деякі функції обчислюються табличні, а інші - за допомогою розкладу в ряд чи ітераційних формул уже з використанням функцій, що обчислюються таблично.

6.5.4. *Таблично-алгоритмічний метод обчислення елементарних функцій*

Застосування табличного методу вимагає великої за розміром таблиці при обробці аргументів з великою розрядністю, що ускладнює зберігання значень функції та їх пошук в таблиці. Тому бажано стиснути інформацію, а потім відновити її з мінімальними втратами точності. Вирішення цієї задачі можливе шляхом застосування таблично-алгоритмічного методу обчислення елементарних функцій. Для таблично-алгоритмічного методу характерна наявність арифметичних операцій, необхідних для обчислення поправки, яка додається до знайденого по таблиці значення, що залежить від старшої частини аргументу. Чим вищі вимоги до точності, тим більше арифметичних операцій необхідно виконати. При обчисленні поправки використовуються як загальний, так і частковий підходи. Загальний підхід застосовується для широкого класу функцій, а принципи використання часткового підходу ґрунтуються на застосуванні відомих тотожностей для кожної конкретної елементарної функції. При загальному підході використовується співвідношення:

ДХ) = ДХв + Хп-в) = г В Д + Ф(Хп-в, Хе),

де Хе - число, утворене в старшими розрядами аргументу Х, Хп-в - число, утворене (п-в) молодшими розрядами аргументу Х, Ф(Хп-з, Хе) - поправка.

Як видно, при загальному підході поправка залежить як від молодшої, так і від старшої частини аргументу. При частковому підході поправка в основному залежить тільки від молодшої частини аргументу. Обидва підходи передбачають використання таблично-апроксимаційного і таблично-ітераційного методів обчислень.

Це, зокрема, метод, що ґрунтується на представленні функції у виді суперпозиції двох підфункцій, де поправка задається*у вигляді усередненого значення шуканої функції на кінцях підінтервалів, на які розбивається область визначення функції. Однак застосуван-

ня цього методу доцільне лише в комп'ютерах, де вимоги до точності невисокі. Можливості варіювання обчислювальною складністю та об'ємом таблиць надають методи кусочно-лінійної і кусочно-поліноміальної апроксимації, а також методи, що ґрунтуються на розбивці аргументу X на складові X і X, утворені відповідно його старшими та молодшими розрядами, і розкладанні функції в ряд Тейлора та використанні схеми Горнера.

6.6. *Операції перетворення даних*

6.6.1. Перетворення даних із формату з фіксованою у формат з рухомою комою та навпаки

Для перетворення даних із формату з фіксованою у формат з рухомою комою та навпаки спочатку мають бути визначені параметри форматів представлення даних, наприклад, це перетворення даних із формату з одинарною точністю за стандартом IEEE-754 до 32-розрядного формату з фіксованою комою.

При перетворенні даних із формату з фіксованою у формат з рухомою комою потрібно привести мантису до прийнятого в форматі з рухомою комою діапазону (за стандартом IEEE-754 зробити її нормалізованою), та забезпечити обчислення порядку з тим, щоб не змінилося значення числа. Потрібно відзначити, що будь-яке число в форматі з фіксованою комою може бути представлено в форматі з рухомою комою. При цьому, якщо розрядність цього числа менша або рівна розрядності мантиси, воно буде представлено точно, якщо ж більша - наближено, оскільки частина молодших розрядів буде втрачена.

При перетворенні даних із формату з рухомою у формат з фіксованою комою потрібно привести мантису до прийнятого в форматі з фіксованою комою діапазону при зведенні порядку до нульового значення з тим, щоб не змінилося значення числа.

Розглянемо кілька прикладів.

Нехай потрібно перетворити ціле число $X = 79_{10} = 1001111_2$, яке представлене в двійковій формі в доповняльному 16-розрядному коді, як показано на рис. 6.24, у формат з рухомою комою розрядністю п=16 = т+к, де к=6 - розрядність порядку, а т=10 - розрядність мантиси.

15 14 ... 1 0

0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 1

Рис. 6.24. Ціле число X= 79• = 1001111^ яке представлене в двійковій формі в доповняльному 16-розрядному коді

Значення порядку числа визначається з виразу $E_x = P_x+П$, (Б = 2*' = 32). Тут Р = п-1 - початкове значення порядку, рівне розрядності числа з фіксованою комою без врахування знакового розряду, *В* - зміщення порядку. Тобто початкове значення експоненти рівне $E = 32+15 = 101111_2$. Послідовність перетворення даних із формату з фіксованою у формат з рухомою комою показана в табл. 6.12.

Таблиця 6.12

Номер такту	Мантиса	Експонента
0	0,000000001001111	101111
1	0,000000010011110	101110

2	0,000000100111100	101101
3	0,000001001111000	101100
4	0,000010011110000	101011
5	0,000100111100000	101010
6	0,001001111000000	101001
7	0,010011110000000	101000
8	0,100111100000000	100111

Таким чином, елементи числа з рухомою комою визначені. Його значення в заданих межах представлено на рис. 6.25.

15 14 ... 1 0

0 1 0 0 1 1 1 1 0 0 1 1 1 1 0 0

Рис. 6.25. Перетворене число

Для перевірки правильності результату перетворимо показане на рис. 6.25 число в десяткове:

$$X = +0{,}1001111 \cdot 2^{39\,32} = +1001111{,}0_{(2)} = +79_{(10)}.$$

Нехай потрібно перетворити число $X=-32768_{ш}= -1000000000000000_{2}$ в формат з рухомою комою в тій самій розрядній сітці п = 16 = т+1с, де 1с=6 - розрядність порядку, а т=10 - розрядність мантиси.

Двійковий еквівалент перетворюваного числа в 16-розрядній сітці з фіксованою комою в доповняльному коді має вигляд, показаний на рис. 6.26.

15 14 ... 10

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Рис. 6.26. Перетворюване число

Значення порядку числа визначається з виразу $E_x = P_x+0$, (Б = $2^{к1}$ = 32). Тобто початкове значення експоненти рівне Е = 32+15 = 101111_{2}- Як видно з рис. 6.26, тут достатньо зсунути перетворюване число на один розряд праворуч, і, тим самим, отримати нормалізовану мантису, залишивши на місці знаковий розряд, та додати до початкового значення порядку одиницю. Тобто при від'ємному значенні мантиса зсувається праворуч, а до порядку добавляється одиниця. Таким чином, елементи числа з рухомою комою визначені. Його значення в заданих межах представлено на рис. 6.27.

15 14 ... 10

0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0

Рис. 6.27. Перетворене число

Для перевірки вірності результату перетворимо показане на рис. 6.27 число в десятковe: x

$$X = -0{,}1.2^{48"32} = -\ 1000000000\ \ 000000{,}0_{(2)} = -\ 32768_{(10)}.$$

6.6.2. Перетворення даних з двійково-десяткового коду в двійковий та навпаки

Як ми вже бачили в попередньому розділі, двійково-десяткова система числення - це система, у якій кожну десяткову цифру від 0 до 9 подають 4-розрядним (або більшої розрядності) двійковим еквівалентом. Для виконання перетворення можуть бути використані відповідні таблиці. Розглянемо приклади.

Нехай потрібно перетворити десяткове число 3691_{10} в двійково-десятковий код.

При перетворенні кожну цифру десяткового числа перетворюють на його двійковий 4-розрядний еквівалент.

Десяткове число	3	6	9	1
Двійково-десяткове число	0011	0110	1001	0001

Отже, 3691_{10} = ООП 0110 1001 0001_{2_10}.

Нехай потрібно перетворити двійково-десяткове число 1000000001110010 на десятковий еквівалент.

Кожна тетрада двійково-десяткового числа перетворюється на десятковий еквівалент:

Двійково-десяткове число	1000	0000	0111	0010
Десяткове число	8	0	7	2

Отже, $1000\ 0000\ 0111\ 0010_{2\ 10}$ = 8072_{10}.

Аналогічним чином здійснюється перетворення й інших кодів.

6.7. Операції реорганізації масивів і визначення їх параметрів

Масив - це впорядкований скінчений набір даних одного типу, які зберігаються в послідовно розташованих комірках оперативної пам'яті і мають спільну назву. Масив складається з елементів. Кожний елемент має ім'я та індекси, за якими його можна знайти в масиві. Кількість індексів елемента визначає розмір масиву. У математиці поняттю масив відповідають поняття вектора та матриці.

Характеристикою масиву є його розмір - загальна кількість елементів у масиві. Інша характеристика масиву - його розмірність, оскільки масиви можуть бути як одновимірні, так і багатовимірні, а також розміри в кожному з вимірів.

Над масивами можуть виконуватись наступні операції: сортування, пошук максимуму або мінімуму, вибір заданого масиву, зсув елементів масиву, стиск масиву, визначення параметрів масиву.

Алгоритм сортування - це алгоритм для впорядкування елементів масиву. До алгоритмів сортування належать такі: сортування за методом бульбашки, сортування методом вставок, сортування злиттям, цифрове сортування, порозрядне сортування, двійкове дерево сортування, блокове сортування, сортування методом вибору, сортування методом Шелла та інші.

Розглянемо основні принципи виконання декількох вищеназваних алгоритмів сортування. Завдання сортування полягає в здійсненні перестановки елементів масиву таким чином, щоб впорядкувати їх по зростанню чи спаданню їх значень.

При виконанні сортування за методом бульбашки максимальні елементи ніби бульбашки спливають до початку масиву.

При сортуванні вставкою впорядкування елементів масиву здійснюється шляхом порівняння елемента масиву з усіма іншими та його розміщення відповідно до його значення.

При сортуванні за методом вибору впорядкований масив будується шляхом багаторазового застосування вибору мінімального елемента з масиву, його вилученням з масиву і додаванням в кінці нового масиву, який спочатку має бути порожнім.

Сортування за методом бульбашки, вставкою та за методом вибору виконується за пропорційну квадрату розміру масиву чисел кількість порівнянь.

Завдання пошуку максимуму або мінімуму вирішується шляхом розбиття масиву на підмасиви, аж до масиву із двох елементів, та рекурсивного вибору з них більшого або меншого. На рис. 6.28 приведено приклад послідовності дій при знаходженні максимуму або мінімуму для масиву із 8 чисел.

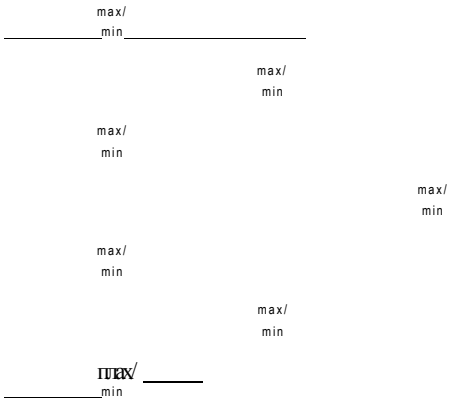


Рис. 6.28. Знаходження максимального і мінімального елементів масиву

Задачі зсуву елементів масиву по заданій координаті, транспонування масиву та визначення параметрів масиву, а також операція стиску (розширення) масиву по заданій координаті, вирішуються шляхом роботи з адресами його елементів у пам'яті.

6.8. *Операції обробки символів та рядків символів*

Є два типи операцій над символами - аналіз, тобто визначення значення символу, і перетворення, тобто зміна значення кодів символів. До основних операцій над символами належать наступні:

• Ідентифікація. Ця операція дозволяє визначити відповідність (невідповідність) значення коду аналізованого символу X коду заданого символу C. Ідентифікація символу X виконується шляхом перевірки збігу коду символу X з кодом символу C, розміщених в основній пам'яті або регістрах, і вказаних відповідними адресами. Результатом операції

є 1, якщо коди символів співпадають, і 0, якщо не співпадають. Операція ідентифікації виконується як операція порівняння двійкових кодів на збіжність, описана в п. 6.4.1.

• Перевірка за маскою. Операція виконується з метою визначення відповідності коду досліджуваного символу X певній ознаці, що вказується кодом символу маски M. Вона передбачає перевірку збігу кодових розрядів символу *X* з кодовими розрядами символу маски M. Це дозволяє класифікувати код символу на приналежність до певної групи символів (наприклад, символи цифр, букв, спеціальних знаків і т. п.).

• Порівняння символів. Порівняння символу X із заданим символом C дозволяє визначити відношення ваг кодів цих символів. Коди символів при цьому зазвичай розглядаються як двійкові цілі числа без знаків (можливі й інші інтерпретації), і порівняння символів виконується як порівняння числових значень їх кодів, в результаті якого визначаються співвідношення $X = C$, $X > C$ і $X < C$. Порівняння символів проводиться як операція визначення старшинства двійкових кодів, описана в п. 6.4.2.

• Запис символу за маскою. Ця операція дозволяє змінювати код символу X згідно з кодом символу маски M шляхом видалення не вказаних (або, навпаки, вказаних) в коді символу маски двійкових розрядів коду символу X. Тобто запис символу за маскою забезпечується вибірковим записом одиничних значень тільки тих кодових розрядів символу X, яким відповідають одиничні значення розрядів коду символу маски M.

• Запис зони байта. Ця операція є окремим випадком операції запису символу за маскою, вживаній при обробці кодів символів байтової структури. Операція дозволяє переслати код лівої (зонової) або правої (цифрової) частини символу X у відповідну частину іншого символу У, тобто чотирьох правих і лівих двійкових розрядів коду X в аналогічні розряди коду У. Інша частина кодів У і X не змінюється.

Рядок представляє послідовність символів, що зберігаються в пам'яті комп'ютера та створюють певну семантичну одиницю оброблюваної інформації. Рядки символів можуть мати фіксовану, змінну або довільну довжину. Одиницею обробки інформації рядків символів є не двійковий розряд, а символ. Тому і всі операції над рядками символів розглядаються як операції над впорядкованими послідовностями символів.

Аналогічно до операцій над символами, операції над рядками символів можна розділити на два види: аналізу, які не змінюють вмісту оброблюваних рядків, і перетворення, за допомогою яких змінюється семантичний або синтаксичний вміст рядків.

До операцій аналізу належать операції пошуку символу та перевірка за маскою. Операція пошуку символу дозволяє визначити наявність і місцезнаходження певного (заданого) символу в рядку. Операція завершується при виявленні першого шуканого символу в рядку, або закінченням посимвольної перевірки цього рядка, якщо шуканий символ відсутній в цьому рядку. Положення виявленого символу визначається порядковим номером символу в рядку або його адресою (при посимвольній системі адресації). Пошук виконується шляхом посимвольного порівняння кодів символів рядка з кодом заданого символу на виконання заданого відношення (=, >, <). Зазвичай передбачаються окремі операції пошуку для кожного з можливих відношень. Коди символів інтерпретуються як двійкові числа без знаків. Залежно від прийнятої системи кодування можливі й інші закони визначення ваг двійкових розрядів кодів символів. Результат виконання операції відображається станом спеціального індикатора і вмістом регістра адреси пам'яті, відповідним адресі останнього звернення до чергового досліджуваного симво-

лу. Кінець досліджуваного рядка визначається після закінчення обробки певного числа символів (при фіксованій довжині слова), після обробки вказаної в команді кількості символів (при змінній посимвольній адресації), або при виявленні ознаки кінця рядка (для слів довільної довжини). Очевидно, що ситуація відсутності шуканого символу в рядку утворює один із станів коду умов або індикаторного регістра. Перевірка за маскою аналогічна операції пошуку символу, тільки виконується пошук не одного символу, а їх послідовності.

До операцій перетворення рядків символів належать операції компонування, редагування, перекодування та перетворення форматів байтів.

За допомогою компонування (запису за маскою) можна виконати об'єднання вмісту двох рядків X і Y за кодом символу маски M шляхом запису символів одного рядка (X) в місце розташування символів іншого рядка (Y) в тих позиціях, які вказані кодом маски. Інші символи першого рядка, другого рядка і код маски не змінюються. Компонування використовується для формування нового рядка з послідовностей символів двох початкових рядків зміною значення деяких символів рядка або зміною послідовності символів в рядку. В операції беруть участь три операнди: змінний рядок Y, рядок даних X і символ (може бути рядок) маски M, послідовність двійкових розрядів якої відповідає символам перших двох операндів зліва направо (або навпаки). Перші два операнди зазвичай задаються їх адресами, а код маски вказується безпосередньо в команді. За цих умов зміні підлягають символи Y, яким відповідає одиничне значення двійкового розряду коду маски M.

Редагування дозволяє видозмінити заданий рядок F (звичайно цей текст представляє число, що зберігається для економії пам'яті в найбільш компактному вигляді) згідно з певним шаблоном S. Необхідність в такій зміні часто виникає при виведенні наборів різних змінних даних на друк згідно з формою їх відображення, що вимагає, наприклад, проставлення спеціальних знаків, записів, пояснень і т. п. Необхідні редакторські дії задаються відповідними керуючими символами, що включаються в необхідній послідовності в рядок-шаблон, відповідно до якого і виконується операція редагування. Достатню гнучкість редагування забезпечують наступні операції над керуючими символами в рядку-шаблоні:

- заміна символів-заповнювачів в тексті шаблону символами рядка даних у порядку їх проходження (зазвичай зліва направо);
- проставлення десяткової крапки (або коми) в заповнюваному числовим рядком контексті шаблону;
- видалення незначущих нулів при заповненні контексту шаблону числовим рядком (заміна їх пропусками);
- вказівка місця початку значущості числових даних в контексті шаблону (для виключення заданого раніше видалення незначущих нулів, починаючи з цього місця);
- задання в рядку шаблону будь-яких послідовностей постійних текстових, числових даних або спеціальних символів;
- вказівка кінця шаблону (якщо формат рядка шаблону не має постійної або змінної довжини).

Редагування цифрового рядка F здійснюється посимвольно зліва направо згідно з вмістом послідовності керуючих символів рядка шаблону S. В результаті виконання one-

рації виходить послідовність символів рядка шаблону Б, в якому символи-заповнювачі (а також символи керування видаленням незначущих нулів і початку значущості, якщо вони є) замінені конкретними значеннями цифр редагованого числового рядка. Саме редаговане число при цьому не змінюється. Шаблон і редаговане число задаються їх адресами, а операція закінчується при виявленні символу-покажчика кінця шаблону при послідовному його огляді під час виконання операції. Зазвичай шаблон створюється для редагування послідовності одного числа, проте це не є обов'язковим, оскільки виконання операції керується повністю контекстом шаблону і єдиною додатковою вимогою для редагування декількох чисел за одним шаблоном є послідовне розташування цих чисел в пам'яті в порядку, відповідному послідовності їх включення в контекст відредагованого рядка згідно з заданим форматом шаблону. Природно, при цьому необхідне багатократне використання символів керування видаленням незначущих нулів і вказівки місця початку значущості. Приймається, що дія попереднього символу, задаючого видалення нулів, анулюється першим подальшим символом вказівки місця початку значущості і будь-яким символом тексту шаблону, окрім символів десяткової крапки (або коми) та символів-заповнювачів. Знак редагованого числа розглядається як чергова цифра цього числа.

Перекодування призначене для виконання переходу від однієї системи кодування символів до іншої, що довільно задається. Передбачається, що належний перекодуванню рядок зберігається в послідовних комірках пам'яті, починаючи з деякої адреси, і задана таблиця перекодування, що зберігається в пам'яті, також починається з деякої адреси. Таблиця перекодування містить перелік належних перекодуванню символів в новій системі кодування. Перекодування виконується заміною чергового символу одного рядка відповідним новим кодом цього символу з таблиці перекодування. Перекодування може супроводжуватися визначенням і відповідною індикацією деяких умов (наприклад, наявність тільки цифрових і алфавітних, або ж тільки визначених символів в рядку, що перекодовується) і завершенням операції при виконанні або невиконанні певних умов. Таким чином, можуть формуватися декілька варіантів операцій перекодування.

Перетворення форматів байтів застосовується для економного зберігання цифрових даних в пам'яті та зручності представлення їх вмісту на зовнішніх документах, що підлягають сприйняттю людиною. Як відомо, для зберігання цифрових даних достатньо 4 двійкових розряди на десяткову цифру, тобто в байті, призначеному для зберігання коду одного алфавітно-цифрового символу, можна розмістити коди двох десяткових цифр, або коди цифри і знаку, що удвічі скорочує об'єм пам'яті, необхідної для зберігання цих даних. Проте при виводі на друк або інші зовнішні носії інформації необхідно привести коди всіх символів до єдиного формату, оскільки інакше неможлива правильна їх інтерпретація з обліком тільки кодів байтів.

Таким чином, при байтовій організації пам'яті для зберігання одиниць числової і символьної інформації застосовують два формати, які умовно називаються упакований - по дві десяткові цифри в байті і зоновий - одна десяткова цифра (або знак) в байті. В останньому випадку двійковий код цифри (або знаку) прийнято розміщувати в правих чотирьох розрядах коду байта і називати цю його частину цифровою, а ліва частина байта при цьому містить спеціальний код зони, використовуваний як ознака цифрових даних, і називається зоновою частиною.

Для перетворення форматів представлення цифрових даних застосовують 4 операції: пакування - перехід від розширеного формату до стислого, розпаковування - зворотна операція пакуванню; пересилку цифр - перезапис цифрових частин байтів і пересилку зон - перезапис зонових частин байтів.

6.9. Короткий зміст розділу

В розділі розкриті основні питання виконання операцій обробки даних: логічних (логічне множення, логічне додавання, інверсія і т. д.), зсуву (праворуч, ліворуч), відношення (менше, більше, рівне, менше-рівне, більше-рівне), арифметичних (додавання, віднімання, множення та ділення), обчислення елементарних функцій, перетворення даних (перетворення із формату з фіксованою в формат з рухомою комою і навпаки, перетворення з двійково-десяткового коду в двійковий і навпаки), реорганізації масивів і визначення їх параметрів (сортування, пошук максимуму або мінімуму, вибір заданого масиву, зсув елементів масиву, стиск масиву), обробки символів та стрічок символів (пошук символу, зсув, заміна символів в стрічці, пакування стрічок символів, порівняння стрічок символів).

Розглянуті основні алгоритми виконання вищеназваних операцій.

6. / 0. Література для подальшого читання

Алгоритми виконання логічних (логічне множення, логічне додавання, інверсія і т. д.), зсуву (праворуч, ліворуч), відношення (менше, більше, рівне, менше-рівне, більше-рівне) та арифметичних (додавання, віднімання, множення та ділення) наведені в багатьох працях, присвячених питанням побудови комп'ютерів. В першу чергу серед них потрібно відзначити роботи [1-5]. В роботах [6-11] описані алгоритми обчислення елементарних функцій за методом "цифра за цифрою". Питанням побудови табличних та таблично-алгоритмічних методів обчислення елементарних функцій присвячені роботи [12-25]. Алгоритми сортування детально описані в роботах [26, 27]. Клас алгоритмів паралельного сортування описаний в роботі [28]. Виконання операцій обробки символів та стрічок символів наведено в [3].

6.11. Література до розділу 6

1. Прикладная теория цифровых автоматов / К. Г. Самофалов, А. М. Романкевич, В. Н. Валуйский, Ю. С. Каневский, М. М. Пиневич. - К.: Вища шк., 1987. - 375 с.

2. Корнейчук В. И., Тарасенко В. П. Основы компютерной арифметики. - К. Корнейчук, 2002. - 176 с.

3. Рабинович З. Л., Раманаускас В. А. Типовые операции в вычислительных машинах. - К.: Техника, 1980. - 264 с.

4. Карцев М. А. Арифметика цифровых машин. - М.: Наука, 1969.

5. Савельев А. Я. Арифметические и логические основы цифровых автоматов. - М.: Наука, 1980. - 255 с.

6. Бойков В. Д., Смолов В. Б. Аппаратурная реализация элементных функций в ЦВМ. - Л. ЛГУ -96 с.

7. Благовещенский Ю. В., Теслер Г. С. Вычисление элементарных функций на ЭВМ. - К.: Техника, 1977. - 208 с.

8. Оранский А. М. Аппаратные методы в цифровой вычислительной технике. - Минск, Из во БГУ, 1977. - 208 с

9. Voider J. E. The CORDIC trigonometric computing technique. - "IRE Trans.", 1959, 3, pp. 330-334.

10. Walther I. S. -In: Proc. Spring Joint Comput. Conf Monthvale, 1971, V.38, N.J.: AFIPS Press, 1971.

11. Meggite I. E. Pseudodivision and Pseudomultiplication process. - IBMJ. Res. Develop., v. 6., 1962,№ 2

12. Смолов В. В., Байков В. Д. Анализ табличных и таблично алгоритмических методов воспроизведения элементарных функций. - Электронное моделирование, 1980, № 1, с. *22-27.*

13. Колубай С. К., Мурашко А. Г Принципы построения процессоров типа "память система поиска" // УСИМ, 1977, № 4, с. 58- 62.

14. Хемел А. Выполнение математических операций с помощью ПЗУ // Экспрессинформация, серия "Вычислительная техника", 1970, № 32, с. 27-29.

15. Смолов В. Б., Байков В. Д. Анализ табличных и таблично алгоритмических методов воспроизведения элементарных функций // Электронное моделирование, 1980, № 1, с. 22-27.

16. Балашов Е. П., Смолов В. Б. и др. К вопросу применения сокращенных таблиц функций для построения высокопроизводительных однородных процессоров // УСИМ, 1975, № 3, с. 99-102

17. Ильин В. А., Попов Ю. А., Дружинина И. И. Об использовании сокращенных таблиц при вычислении элементарных функций // УСИМ, 1979, № 1, с. 58-60.

18. Палагин А. В., Кургаев А. Ф., Кондрачук И. М. К выбору метода вычисления элементарных функций в мини ЭВМ // УСИМ, 1973, № 5, с. 65-69.

19. Потапов В. И., Нестерук В Ф., Флоренсов А Н. Быстродействующие арифметико логические устройства ЦВМ. - Новосибирск, 1978.

20. Потапов В. И., Флоренсов А Н. Таблично аддитивная организация вычисления в ЭВМ функций, принадлежащих к классу дважды непрерывно дифференцируемых // Автоматика и вычислительная техника, 1977, № 6, с. 78-84.

21. Потапов В. И., Флоренсов А Н. Таблично алгоритмическая организация вычислений элементарных функций в ЦВМ. - Изв. вузов, сер. Приборостроение, 1978, т. 21, № 9, с. 63-66.

22. Потапов В. И., Флоренсов А Н. Таблично алгоритмический метод реализации в ЦВМ функции логарифма // УСИМ, 1978, № 4, с. 90-94.

23. Мухопад Ю. Ф., Федченко А И., Лукашенко В М. Таблично функциональные преобразователи с ограниченным числом хранимых констант // УСиМ, 1975, № 4, с. 99-102.

24. Голубков Ю. А., Лебедев А В. Некоторые пути повышения скорости вычисления элементарных функций на ЦВМ. - М., 1962. - 64 с

25. Пелед Ф., Лиу Б. (США). Цифровая обработка сигналов: Теория, проектирование, реализация: Пер с анг. - Киев: Вища школа. Головное изд во, 1979. - 264 с

26. Кнут. Сортировка и поиск.

27. Кун С. Матричные процессоры на СБИС: Пер. с англ. - М.: Мир, 1991. - 672 с

28. Мельник А. А., Илькив В. С. Реализация алгоритмов сортировки. В кн. "Систолические вычислительные структуры". Препринт АН УССР. Ин т ИППММ, № 3, 1987

6.12. *Питання до розділу 6*

1. Назвіть основні операції обробки даних

2. Які основні логічні операції виконуються в комп'ютері? Наведіть таблицю істинності цих операцій

3. Дайте пояснення операцій логічного зсуву

4. Дайте пояснення операцій арифметичного зсуву

5. Дайте пояснення операцій циклічного зсуву

6. Наведіть правило та приклад додавання двійкових чисел без знаків

7. Наведіть правило та приклад віднімання двійкових чисел без знаків

8. Наведіть правило та приклад додавання двійкових чисел, представлених в прямому коді

9. Наведіть правило та приклад додавання двійкових чисел, представлених в оберненому коді

10. Наведіть правило та приклад додавання двійкових чисел, представлених в доповняльному коді

11. Як фіксується переповнення при додаванні двійкових чисел?

12. Приведіть граф алгоритму множення цілих двійкових чисел без знаків

13. Наведіть алгоритми багатомісної операції додавання часткових добутків з використанням операторів паралельного двомісного додавання

14. Наведіть граф алгоритму послідовного попарного додавання часткових добутків, отриманих починаючи з аналізу молодших розрядів множника

15. Наведіть граф алгоритму послідовного попарного додавання часткових добутків, отриманих починаючи з аналізу старших розрядів множника

16. Наведіть граф алгоритму паралельного попарного додавання часткових добутків з використанням структури бінарного дерева

17. Наведіть алгоритми багатомісної операції додавання часткових добутків з використанням операторів двомісного однорозрядного додавання

18. Приведіть граф алгоритму множення двійкових чисел із знаками

19. Поясніть суть та переваги алгоритму множення двійкових чисел за алгоритмом Бута

20. Приведіть та поясніть алгоритми ділення з відновленням та без відновлення залишку. Яка між ними різниця?

21. Як привести число з рухомою комою до нормалізованого вигляду?

22. Приведіть та поясніть алгоритми додавання та віднімання чисел з рухомою комою

23. Приведіть та поясніть алгоритми множення чисел з рухомою комою

24. Приведіть та поясніть алгоритми ділення чисел з рухомою комою

25. Поясніть як виконується операція порівняння двійкових кодів на збіжність та визначення їх старшинства

26. Які є методи обчислення елементарних функцій?

27. Дайте пояснення методу обчислення елементарних функцій шляхом розкладу в ряд

28. Дайте пояснення методу обчислення елементарних функцій шляхом використання ітераційних обчислень

29. Поясніть алгоритм обчислення елементарних функцій методом „цифра за цифрою"

30. Поясніть суть табличного методу обчислення елементарних функцій

31. Поясніть суть таблично алгоритмічного методу обчислення елементарних функцій

32. Опишіть алгоритм перетворення з формату з фіксованою комою до формату з рухомою комою

33. Опишіть алгоритм перетворення з формату з рухомою комою до формату з фіксованою комою

34. Як перетворити двійково десятковий код в двійковий і навпаки?

35. Що таке масив?

36. Назвіть характеристики масиву

37. Назвіть алгоритми сортування чисел

38. В чому полягає задача сортування чисел?

39. Опишіть суть алгоритму сортування методом бульки

40. Опишіть суть алгоритму сортування методом вставки

41. Опишіть суть алгоритму сортування методом вибору елементів масиву

42. Приведіть алгоритм знаходження максимального і мінімального елементів масиву

43. Приведіть перелік основних операцій над символами

44. Приведіть перелік основних операцій над рядками символів

Розділ 7

сАрифметико/лтічтлй/пристрій

7.1. Функції арифметико-логічного пристрою

Арифметико-логічний пристрій (АЛП) призначений для виконання арифметичних, логічних та інших операцій обробки даних над операндами, які представляють собою двійкові числа з фіксованою та рухомою комою, двійково-десяткові числа, команди, адреси, логічні коди, алфавітно-цифрові коди. АЛП є одним з основних вузлів процесора. Інтерфейс АЛП, тобто його зв'язки з іншими вузлами процесора, показано на рис. 7.1.

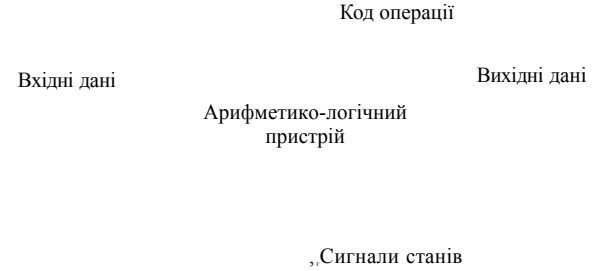


Рис. 7.1. Інтерфейс АЛП

Вхідні дані поступають в АЛП з регістрового файлу процесора або з основної пам'яті (залежно від типу архітектури комп'ютера), до яких записуються і вихідні дані. Код операції поступає з поля коду операції виконуваної команди, яка зберігається в регістрі команди РгК, а сигнали станів АЛП повідомляють пристрій керування про стан ходу виконання операцій та фіксуються в регістрі слова стану програми регістрової пам'яті процесора.

Арифметико-логічний пристрій процесора - це комбінаційна схема (КС) без внутрішньої пам'яті, яка здатна виконувати набір елементарних операцій та деяку множину складних операцій, які ініціюються командами обробки даних з системи команд комп'ютера. В потужних комп'ютерах, а останнім часом і в багатьох однокристальних комп'ютерах, використовуються багатоблокові АЛП з внутрішньою регістровою пам'яттю на основі табличних, однотактових, багататактових та конвеєрних операційних пристроїв. Тип виконуваної операції вказується кодом на вході керування АЛП. Типово в АЛП виконуються такі операції: зсув - зміщення кодів, які зберігаються в регістрах, вліво або вправо на задане число розрядів; додавання до слова 1 або -1 - операція рахунку; дешифрування - перетворення двійкового коду в однорядний код; шифрування

- перетворення однорядного коду в двійковий; порівняння - визначення відношення старшинства двох слів або їх рівності; порозрядне доповнення - формування оберненого коду; порозрядні логічні множення і додавання двох слів; порозрядне додавання двох слів за модулем; сума двох чисел. В багатьох комп'ютерах цей перелік розширений більш складними операціями, наприклад арифметичними, відношення, обробки рядків символів, обчислення елементарних функцій і т. д.

Залежно від способу обробки операндів АЛП діляться на послідовні, послідовно-паралельні та паралельні. В першому випадку обробка операндів в АЛП здійснюється послідовно в часі над кожним розрядом, тоді як в останньому операції здійснюються паралельно в часі над всіма розрядами операндів.

За способом представлення чисел розділяють АЛП з фіксованою та з рухомою комою, причому перші можуть бути орієнтовані на обробку цілих або дробових чисел.

Залежно від способу виконання операцій АЛП діляться на однотактові, коли задана операція виконується за один такт, та багатотактові, коли для виконання операції потрібно виконати деяку кількість тактів.

АЛП можуть бути конвеєрними або скалярними. Використання конвеєрного принципу обробки даних дозволяє суттєво підвищити продуктивність АЛП та комп'ютера в цілому.

За характером використання елементів АЛП діляться на одно- та багатоблокові. В одноблокових (багатофункціональних) АЛП всі операції над всіма типами операндів виконуються тими ж вузлами, які комутуються відповідним чином залежно від потрібного режиму роботи. В багатоблокових АЛП окремі групи операцій над кожним типом операндів виконуються окремими блоками. Це дозволяє підвищити продуктивність АЛП за рахунок паралельного виконання операцій.

7.2. Способи обробки даних в арифметико-логічному пристрої

Залежно від способу обробки операндів АЛП діляться на послідовні, послідовно-паралельні та паралельні.

В послідовних АЛП обробка операндів здійснюється послідовно в часі над кожним розрядом, як це показано на рис. 7.2.

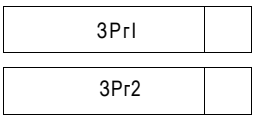


Рис. 7.2. Послідовний спосіб обробки даних в АЛП

Тут на вході АЛП є зсувні регістри ЗРгІ та ЗРг2, з яких дані порозрядно поступають на обробку. Результат з АЛП також порозрядно поступає в вихідний зсувний регістр ЗРгЗ. В кожному такті операнди в зсувних регістрах зміщуються на один розряд вправо. Крім того, можливий зворотний зв'язок з вихідного регістра до входу АЛП. Оскільки обробка здійснюється порозрядно, то для отримання результату потрібно як мінімум п тактів, де п - розрядність операндів. Для складних операцій кількість тактів може становити

п²і більше. Тобто, при використанні цього способу АЛП характеризується малою швидкодією. Разом з тим, він знаходить досить широке застосування при проектуванні малогабаритних комп'ютерів завдяки малим витратам обладнання на побудову таких АЛП.

В паралельних АЛП операції виконуються одночасно над всіма розрядами операндів, як це показано на рис. 7.3.

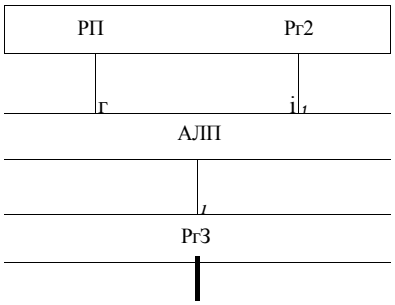


Рис. 7.3. Паралельний спосіб обробки даних в АЛП

Тут на вході АЛП є регістри Ргі та Рг2, з яких дані паралельно поступають на обробку. Результат також паралельно поступає в вихідний регістр РгЗ. Оскільки обробка здійснюється паралельно, вона виконується протягом лише одного такту незалежно від розрядності операндів. Тобто АЛП з паралельним способом обробки даних характеризується високою швидкодією, що і є причиною його широкого використання. Разом з тим, такий АЛП характеризується великими витратами обладнання на його побудову.

Послідовно-паралельний спосіб обробки даних є проміжним стосовно швидкодії та затрат обладнання в порівнянні з вище розглянутими послідовним та паралельним способами. Тут одне з вхідних даних може поступати на обробку в АЛП паралельно, а інше послідовно з видачею проміжного результату в паралельному коді, як це показано на рис. 7.4 а, або вхідні дані можуть поступати в АЛП групами по к і т розрядів, як це показано на рис. 7.4 Ь, та подаватись в вихідний регістр паралельно, або також групами.

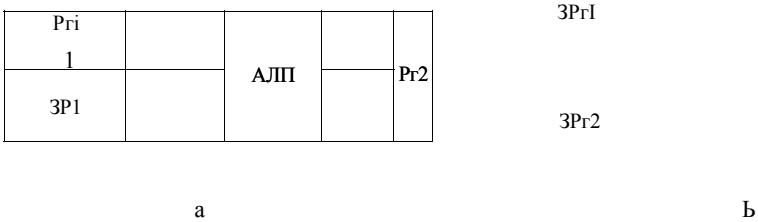


Рис. 7.4. Послідовно-паралельний спосіб обробки даних в АЛП

7.3. Елементарні операції арифметико-логічного пристрою

Складні операції в АЛП реалізуються як послідовність елементарних, тому АЛП будується на основі комбінаційних схем КС, які виконують елементарні операції. До типових елементарних операцій належать:

- зсув - зміщення кодів, які зберігаються в регістрі, вліво або вправо на задане число розрядів;

- додавання до слова 1 або -1 - операція рахунку;
- дешифрування перетворення слів в сигнали (однорядний код);
- шифрування перетворення однорядного коду в двійковий;
- порівняння визначення відношення старшинства двох слів або їх рівності
- порозрядне доповнення формування оберненого коду;
- порозрядне логічне множення і додавання двох слів
- порозрядне додавання двох слів по модулю;
- сума двох чисел

Елементарні операції є основою для виконання більш складних операцій процесора. Алгоритми виконання цих операцій представляються як послідовність елементарних, які називаються мікрокомандами, а набір мікрокоманд мікропрограмами Більше того, в більшості сучасних комп'ютерів елементарні операції входять до складу їх системи команд, не дивлячись на наявність в складі системи команд складних операцій, які вимагають виконання великої кількості елементарних, наприклад операцій компресії даних, шифрування даних і т. д. Це пояснюється двома причинами: наявність в складі системи команд комп'ютера команд виконання елементарних операцій забезпечує його універсальність, і, крім того, ці операції виконуються гранично швидко, що дозволяє досягти високих тактових частот роботи процесора

На основі комбінаційних схем для виконання вищеназваних елементарних операцій синтезуються вузли АЛП для виконання складних операцій, що буде показано далі

Арифметико логічний пристрій для виконання елементарних операцій наявний в кожному універсальному комп'ютері Розглянемо побудову стандартного 4-розрядного АЛП, функціональне позначення та входи виходи якого показано на рис. 7.5. Інтерфейс АЛП включає дві вхідних (А і В) та одну вихідну 4-розрядні шини даних. Дані з вхідних шин обробляються в АЛП відповідно до значення двійкового коду на входах керування М та БО-БЗ. Результат обробки поступає на вихідну шину Е

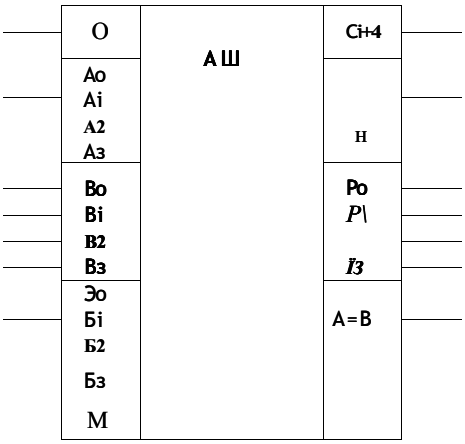


Рис. 7.5. Входи-виходи стандартного 4-розрядного АЛП

Для нарощування розрядності такі АЛП об'єднуються шляхом відповідного з'єднання входів переносу С. та С . . Основою АЛП служить суматор, що виконує операцію додавання двох чисел, схема якого доповнюється відповідними логічними елементами для розширення функцій та забезпечення переключення з однієї операції на іншу. Вхід

M (mode) задає тип виконуваної операції: логічна (M=1) чи арифметико-логічна (M=0). Виходи G і H задають функції генерації і прозорості, які використовуються для організації паралельних переносів при нарощуванні розрядності АЛП з використанням схем прискореного переносу. Вихід A=B є виходом порівняння кодів A та B на збіжність.

Перелік операцій, виконуваних описаним АЛП, приведено в табл. 7.1.

Таблиця 7.1

S	Логічна операція (M=1)	Арифметико логічна операція (M=0)
0000	NOT(A)	A+ C
0001	NOT (A OR B)	A OR B + C
0010	NOT (A) AND B	A OR NOT (B) + C.
0011	0	1 +C.
0100	NOT (A AND B)	A + A AND NOT (B) + C,
0101	NOT (B)	A OR B + A AND NOT (B) + C.
0110	A XOR B	A + NOT (B) + C,
0111	A AND NOT (B)	A AND NOT (B) +1 +C;
1000	NOT (A) OR B	A + A AND B + C
1001	NOT (A XOR B)	A + B + C.
1010	B	A OR NOT (B)+ A AND B + C,
1011	A AND B	A AND B +1 + C.
1100	1	A + A + C.
1101	A OR NOT (B)	A OR B + A + C
1110	A OR B	A OR NOT (B)+A + C
1111	A	A + 1 + C.

В таблиці прийняті наступні позначення: OR - операція диз'юнкції, AND - операція кон'юнкції, XOR - операція нерівнозначності,"+" - операція додавання,"-" - операція віднімання. Позначенням 1 та 0 в таблиці відповідають двійкові коди відповідно 1111 та 0000. Вхідний перенос поступає в молодший розряд слова, тобто до слова додається код 000C

7.4. *Складні операції арифметико-логічного пристрою*

Крім вище перерахованих елементарних операцій, в АЛП виконується велика кількість складних операцій, тобто таких, які реалізуються на основі елементарних. Можна виділити наступний перелік складних операцій АЛП, сформований на основі аналізу системи команд сучасних комп'ютерів:

- логічні операції (логічне множення, логічне додавання, інверсія і т. д.) над двійковими числами;
- операції зсуву (вправо, вліво) на задану кількість розрядів, причому в одному такті зсув може бути здійснено як на один розряд, так і на декілька розрядів;
- арифметичні операції (додавання, віднімання, множення та ділення) над двійковими числами;
- операції відношення: менше, більше, рівне, менше-рівне, більше-рівне;
- операції обчислення елементарних функцій типу exp X, In X, Sin X, Cos X, Sh X, Ch X, піднесення до степеня A^m; arctg y/x;
- операції обробки символів та рядків символів.

Потрібно відзначити, що розглянуті в розділі 4 операції перетворення даних (перетворення із формату з фіксованою в формат з рухомою комою і навпаки, перетворення з двійково-десяткового коду в двійковий та навпаки і т. д.), так само як операції реорганізації масивів і визначення їх параметрів: сортування, пошук максимуму або мінімуму, вибір заданого масиву, зсув елементів масиву, стиск масиву, а також операції пошуку символу, зсув, заміна символів в рядку, пакування рядків символів, порівняння рядків символів виконуються в процесорі на основі елементарних та основних арифметичних і логічних операцій. Разом з тим, в останніх комп'ютерах з метою підвищення продуктивності та в зв'язку з широким використанням засобів телекомунікацій та мультимедіа до складу АЛП вводяться окремі блоки для виконання вищеназваних складних операцій, а також операцій типу кодування, компресії, шифрування даних і т. д.

Розглянемо питання реалізації в АЛП складних операцій більш детально.

7.5. *Використання графа алгоритму при побудові арифметико-логічного пристрою*

Щоб виявити ефективні підходи до реалізації алгоритму виконання складної операції в АЛП, необхідно представити його в формі, яка розкриває його базові обчислювальні та структурні характеристики. Обчислювальні характеристики описуються повним набором функціональних операторів алгоритму і відповідними їм обчислювальними затримками. Структурні характеристики описуються зв'язками між функціональними операторами алгоритму і їх взаємозалежністю.

Однією із можливих форм представлення алгоритму є графічна. На рис. 7.6а представлений граф деякого алгоритму.

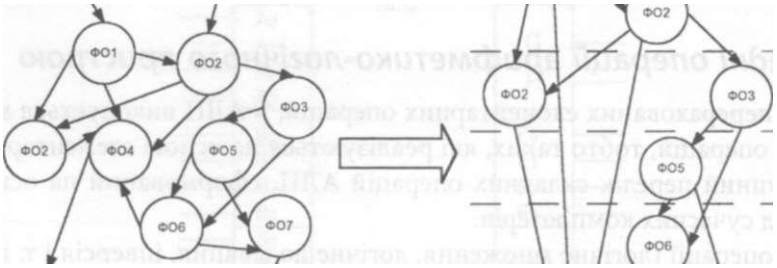


Рис. 7.6. Граф алгоритму

Кожна вершина графа представляє функціональний оператор алгоритму, кожна дуга - зв'язок між функціональними операторами. Вершина, яка відповідає функціональному оператору Фі, з'єднується з вершиною, яка відповідає функціональному оператору

Ф), тільки в тому випадку, коли результат, одержаний після виконання оператора Фі, є одним із аргументів для оператора Ф^'. Описана графічна форма представлення алгоритму дозволяє оцінити його обчислювальні характеристики, тому широко використовується при синтезі АЛП.

Представлення графа алгоритму в такому вигляді не дозволяє в повній мірі оцінити його структурні характеристики, а значить і можливості його розпаралелення та способи апаратурної реалізації.

Виявити в алгоритмі паралелізм і навіть керувати ним, забезпечуючи тим самим можливість знаходження компромісних просторово-часових співвідношень, що є головним при виборі структури обчислювального пристрою, дозволяє представлення графа алгоритму в потоковій (ярусно-паралельній) формі. Таке представлення графа означає розділення всіх його вершин по ярусах таким чином, що в і-му ярусі розміщені тільки функціональні оператори, які залежать хоча б від одного функціонального оператора (і-І)-го яруса і не залежать від функціональних операторів, які не входять в яруси з меншими ніж і номерами. Всередині яруса функціональні оператори між собою не мають з'єднань. Описаний граф називають потоковим графом (ПГ) алгоритму. На рис. 7.6Ь представлено ПГ раніше розглянутого алгоритму, де штриховими лініями розділені яруси графа.

ПГ алгоритму характеризується набором функціональних операторів Фі) (і=1, 2,...,Р;)=1,2,...,Іл), де і - номер яруса, Р - кількість ярусів,) - номер функціонального оператора в ярусі, Іл - їх кількість в ярусі, а також набором каналів Кп зв'язку між функціональними операторами.

При розгляді питань проектування пристрою для виконання деякого алгоритму у вигляді надвеликої інтегральної схеми (НВІС), використовують його конкретизований потоковий граф. В конкретизованому потоковому графі алгоритму всі функціональні оператори входять до бібліотеки функціональних елементів НВІС, на базі якої реалізується алгоритм. Тобто кожному функціональному оператору алгоритму може бути поставлений у відповідність як мінімум один функціональний елемент із бібліотеки функціональних елементів НВІС. Таке представлення алгоритму забезпечує можливість пошуку найефективнішого варіанту його апаратурної реалізації.

7.6. *Виконання складних операцій в арифметико-логічному пристрої*

Виконання деякої операції можна здійснити цілим рядом способів, для яких характерна єдина семантична основа, що визначається графом алгоритму цієї операції, тобто одні і ті ж операції можуть виконуватися по різному та з різним ступенем розпаралелювання.

Якщо використовувати для виконання графа алгоритму багатофункціональний АЛП, який виконує елементарні операції, наприклад, відповідно до табл. 7.1, то це потребує перетворення всіх процесів, які підлягають виконанню за даним алгоритмом, в послідовну процедуру переробки і передачі інформації. Більш того, значна частина функціональних операторів алгоритму реалізується в такому АЛП з використанням достатньо великого числа елементарних операцій. Разом з цим, просторова передача інформації між вершинами графа повинна бути перетворена в послідовну в часі передачу інформації між АЛП і основною пам'яттю або регістрами процесора. Таким чином, послідовно-паралельний процес виконання алгоритму згідно з його графом, що має просторову структуру, перетвориться в

послідовну процедуру переробки і передачі інформації в багатофункціональному АЛП. Це призводить до того, що швидкість обчислення складних операцій в АЛП є низькою.

Висока продуктивність обробки даних в сучасних компютерах досягається завдяки використанню просторового і часового паралелізму з урахуванням досягнень інтегральної технології. Серед основних ідей слід виділити конвеєрну організацію обчислень, поєднання обробки на різних рівнях, використання декількох паралельно працюючих АЛП з конвеєрним принципом обробки даних, застосування секцій векторних регістрів в регістровій пам'яті, розшарування пам'яті, застосування найбільш швидкої елементної бази з тією метою, щоб в максимальній мірі забезпечити розпаралелювання виконання операцій. Особливо цікавою в цьому плані є ідея підвищення продуктивності та ефективності за рахунок статичної і динамічної програмної зміни структури системи в цілях її наближення до структури графа алгоритму, тобто її реконфігурація. В основі концепції АЛП з реконфігурованою структурою лежить принцип, згідно з яким структура АЛП, функції його операційних пристроїв, а також способи організації обчислювального процесу повинні відповідати структурі графа виконуваного алгоритму, а не навпаки, як це має місце у традиційних АЛП, коли в цілях реалізації алгоритму в рамках жорсткої структури АЛП алгоритм модифікується так, що йому надається форма, відповідна структурі АЛП. Чим більше адаптована структура АЛП до специфіки вирішуваних задач, тим суттєвіше покращуються його характеристики. Така адаптація досягається шляхом перекладення обчислювальних процедур з програмного забезпечення на апаратну частину, чому сприяють вражаючі успіхи в області інтегральної технології. Це, зокрема, поява програмованих логічних інтегральних схем (ПЛІС), що дозволяє в найкоротші терміни реалізувати на їх базі крупні функціональні вузли. Це і можливість створення спеціалізованих НВІС місткістю в десятки і сотні мільйонів транзисторів, що привело до радикальних змін як в області технологічних та інструментальних засобів, так і в принципах проектування.

Таким чином, розглядаючи принципи побудови АЛП, доцільно це робити в тісному зв'язку з графом виконуваного алгоритму, що дозволяє виявити всі форми паралелізму, наявні в ньому. Граф алгоритму, а особливо його потокова форма, в наочній формі відображає фундаментальні обмеження, що визначають внутрішню семантику ряду різних алгоритмів для отримання одних і тих же результатів. Найбільш високі параметри досягаються в АЛП, структура яких більше наближена до структури графа виконуваного алгоритму.

7.7. *Структура арифметико-логічного пристрою*

В більшості комп'ютерів АЛП виконує операції над двома вхідними даними, тобто є двомісним, та видає один вихідний результат, як це показано на рис. 7.7. При цьому спочатку операнди А та В записуються у вхідні регістри Ргі і Рг2, та поступають на входи АЛП через мультиплексори МП1 і МП2, які керуються сигналами У1 та У2. Після цього в АЛП виконується задана операція, тип якої задається кодом операції. Результат операції поступає на вихід АЛП та записується у вихідний регістр РгЗ. З виходу вказаного регістра результат поступає в регістровий файл процесора, а крім того, якщо він потрібний для виконання наступної операції, він поступає через мультиплексори МП1 або МП2 на один з входів АЛП, що здійснюється шляхом подання відповідних значень керуючих сигналів на ходи мультиплексорів.

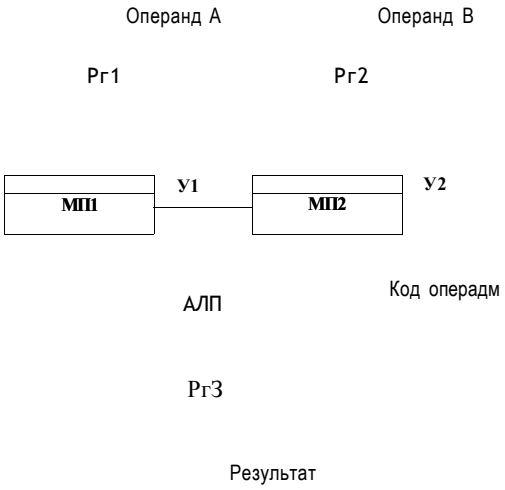


Рис. 7.7. Типова структура АЛП

В сучасних комп'ютерах АЛП є багатоблоковими. В них окремі групи операцій над кожним типом операндів виконуються окремими блоками, які називаються операційними пристроями. Це дозволяє підвищити продуктивність АЛП за рахунок паралельного виконання операцій. Узагальнена структура АЛП сучасного комп'ютера представлена на рис. 7.8.

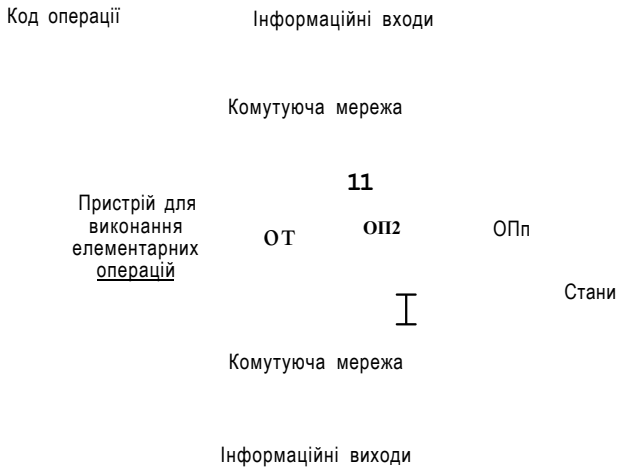


Рис. 7.8. Структура АЛП сучасного комп 'ютера

До складу АЛП, крім вищеописаного пристрою для виконання елементарних операцій, входять п операційних пристроїв ОП1, О П 2 , О П п , які виконують складні операції. Кількість п цих пристроїв та їх функції визначаються конструкторами комп'ютера залежно від сфери його використання. Входи та виходи операційних пристроїв АЛП підключаються до його інформаційних входів та виходів за допомогою комутуючих мереж, якими керує код виконуваної операції. Цим же кодом вибирається тип виконуваної операції в пристрої для виконання елементарних операцій та в операційному пристрої, якщо він може виконувати декілька операцій.

Як приклад АЛП реального комп'ютера, на рис. 7.9 представлено АЛП програмованого процесора NIOS 2.0 фірми Altera. Як бачимо, цей АЛП має наступні блоки: два вхідних регістри RA та RB, два двовходових мультиплексори для подачі даних на обробку або з вхідних регістрів, або з вихідного регістра, операційні пристрої - арифметичний, логічний, зсуву та виділення байтів і слів, а також вихідний мультиплексор, необхідний для підключення до входу вихідного регістра виходу відповідного операційного пристрою, і сам вихідний регістр.

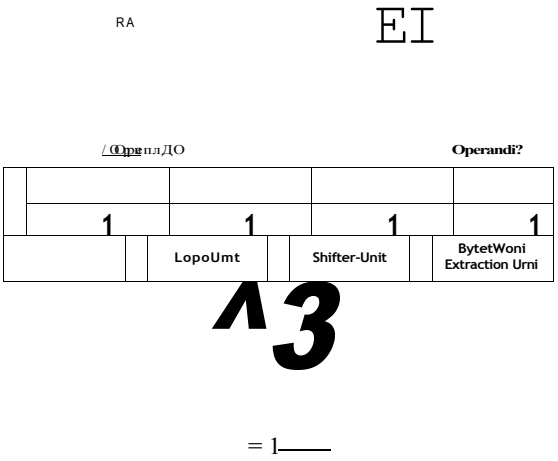
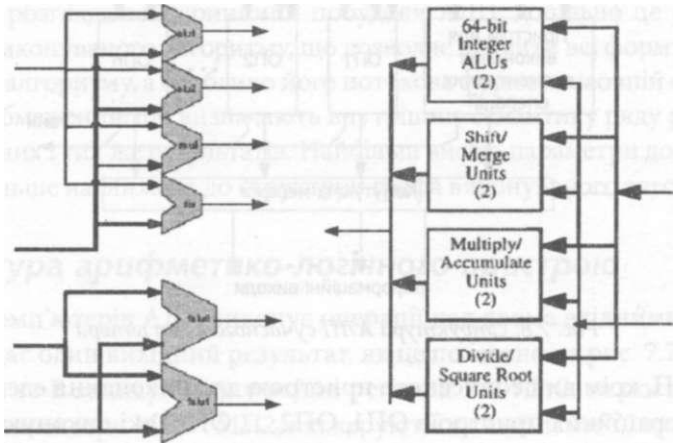


Рис. 7.9. Структура АЛП програмованого процесора NIOS 2.0 фірми Altera

Ще більшу кількість паралельних блоків мають АЛП процесорів UltraSPARC фірми Sun Microsystems та PA-8000 фірми Huwlett-Packard, структури яких наведено на рис. 7.10a та рис. 7.10b відповідно.



a b

Рис. 7.10. Структури АЛП процесорів UltraSPARC фірми Sun Microsystems (a) та PA-8000 фірми Huwlett-Packard (b)

АЛП процесора UltraSPARC фірми Sun Microsystems має два блоки для виконання елементарних операцій над цілими числами (ALU1 та ALU2), перемножувач та поділь-

ник чисел з фіксованою комою (MUL, DIV), два АЛП для обробки даних з рухомою комою (FALU0 та FALU1). АЛП процесора UltraSPARC може виконувати до чотирьох операцій за один такт.

Процесор PA-8000 фірми Huwlett-Packard є суперскалярним процесором з невпорядкованим виконанням команд, який може виконувати до чотирьох команд за один такт. Його АЛП включає по два наступних блоки (длоки, розміщені зверху донизу на рис. 7.10b): 64-розрядне АЛП для виконання елементарних операцій над цілочисловими даними, операційний пристрій зсуву та сортування, операційний пристрій множення та накопичення, операційний пристрій ділення та добування квадратного кореня.

Функції пристрою для виконання елементарних операцій раніше було розглянуто. Розглянемо далі організацію роботи операційних пристроїв.

7.8. Типи *операційних пристроїв*

Залежно від принципів побудови можна провести наступну класифікацію операційних пристроїв:

- Табличні операційні пристрої - операційні пристрої, в яких значення результату шукається за адресою, рівною значенню операнда, в наперед сформованій таблиці, в ролі якої використовується пам'ять.
- Алгоритмічні операційні пристрої - операційні пристрої, в яких задана операція виконується шляхом апаратної реалізації алгоритму цієї операції.
- Таблично-алгоритмічні операційні пристрої - стиснута в таблиці (пам'яті) інформація відновлюється шляхом виконання елементарних та складних операцій.

Під апаратною реалізацією алгоритму розуміється його виконання без участі програміста (програми, яка зчитується з основної пам'яті). При цьому алгоритмічні та таблично-алгоритмічні операційні пристрої в свою чергу діляться на:

- Багатотактові операційні пристрої - операційні пристрої, в яких задана операція виконується шляхом послідовного потактового виконання функціональних операторів алгоритму цієї операції. Потактове виконання операцій в таких пристроях задається мікропрограмою, яка складається з мікрокоманд, що виконують елементарні операції.
- Однотактові операційні пристрої - операційні пристрої, в яких апаратно відображається граф виконуваного алгоритму і задана операція виконується шляхом одноразового проходження даних через операційні вузли, які виконують функціональні оператори алгоритму цієї операції. Виходячи з принципів побудови, такі пристрої можна назвати граф-алгоритмічними операційними пристроями.
- Конвеєрні операційні пристрої - операційні пристрої, в яких апаратно відображається потоковий граф виконуваного алгоритму з розділенням ярусів графа регістрами. Виходячи з принципів побудови, такі пристрої можна назвати конвеєрними граф-алгоритмічними операційними пристроями.

Розглянемо спочатку будову та особливості найпростіших з названих табличних операційних пристроїв, далі принципи побудови багатотактових, однотактових, та конвеєрних операційних пристроїв, основні ідеї, закладені в таблично-алгоритмічних операційних пристроях, а після цього питання організації роботи операційних пристроїв, побудованих за принципами приведеної вище класифікації, для виконання конкретних операцій.

тів. Вже навіть при невеликих розрядностях аргументів об'єм ПЗП для даного випадку є значним, тому цей метод рідко застосовується для виконання операцій над більш ніж одним аргументом.

Розглянемо приклад. Нехай розрядність вхідних та вихідних даних рівна п=т=8 бітів, а кількість виконуваних функцій К=4. Тоді об'єм ПЗП буде рівним 0 = 4*8*2[16] = 256-КВ, що цілком прийнятно для реалізації.

Розглянемо інший приклад. Нехай розрядність вхідних та вихідних даних рівна п=т=16 бітів, а кількість виконуваних функцій К=4. Тоді об'єм ПЗП буде рівним 0 = 4*16*2[32] = 32СВ, що реалізувати на даний час неможливо.

До інших недоліків табличного методу, які також обмежують його використання, належать:

- зростання часу звертання до пам'яті при збільшенні її об'єму;
- великий обсяг попередніх обчислень для розрахунку вмісту таблиць;
- великі витрати часу на запис обчислених значень у ПЗП.

Таким чином, використання табличних операційних пристроїв є доцільним при малих розрядностях аргументів.

7. ТО. *Багатотактовий операційний пристрій*

До складу багатотактового операційного пристрою, структура якого приведена на рис. 7.13, входять: багатофункціональна комбінаційна схема, вхідні і вихідні регістри, а також мультиплексори, необхідні для створення каналів передавання даних. Формування керуючих сигналів для запису даних до регістрів, задання коду операції та керування пропуском даних через мультиплексори, здійснює пристрій керування операційним пристроєм, принципи побудови якого будуть розглянуті в наступному розділі.

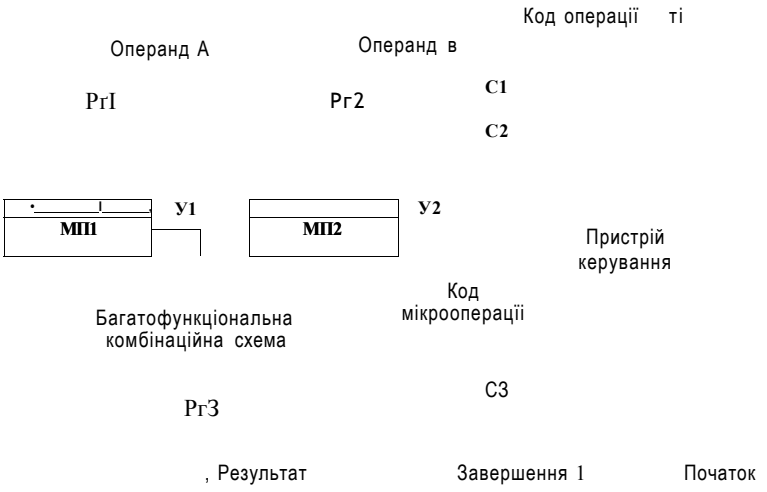


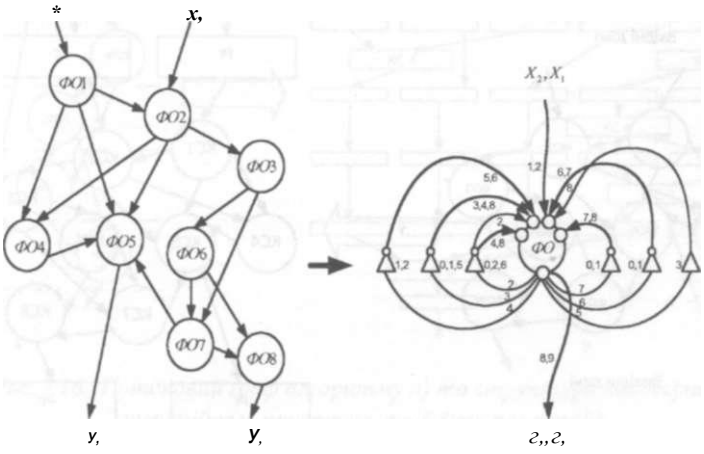
Рис. 7.13. Структура багатотактового операційного пристрою

Багатотактовий операційний пристрій виконує операції над двома вхідними даними, тобто є двомісним, та видає один вихідний результат, як це показано на рис. 7.13. При цьому спочатку операнди А та В записуються у вхідні регістри Ргі і Рг2, та по-

ступають на входи багатофункціональної комбінаційної схеми через мультиплексори МШ та МП2, які керуються сигналами У1 та *У2,* що поступають з пристрою керування. Після цього в багатофункціональній комбінаційній схемі виконується задана мікрооперація, тип якої задається кодом мікрооперації, що поступає з пристрою керування. Результат мікрооперації поступає на вихід багатофункціональної комбінаційної схеми та записується у вихідний регістр РгЗ. З виходу вказаного регістра проміжний результат поступає через мультиплексори МП1 або МП2 на один з входів багатофункціональної комбінаційної схеми для виконання наступної мікрооперації, що здійснюється шляхом подання з пристрою керування відповідних значень керуючих сигналів У1 та У2 на входи мультиплексорів. Запис даних до регістрів здійснюється сигналами С1 -СЗ з пристрою керування. Пристрій керування починає виконання операції після поступлення на його входи сигналу початку роботи, коду операції та тактових імпульсів ТІ. Після завершення виконання операції на виході регістра РгЗ буде результат операції, а пристрій керування сформує сигнал завершення роботи

При побудові багататактового операційного пристрою виникає дві задачі: синтез багатофункціональної комбінаційної схеми, яка б забезпечувала реалізацію всіх функціональних операторів алгоритму, та синтез пристрою керування, який генерує сигнали керування для забезпечення організації виконання функціональних операторів алгоритму шляхом тимчасового запам'ятовування проміжних результатів в регістрах та подання їх на входи багатофункціональної комбінаційної схеми в потрібному такті

Для синтезу багатофункціональної комбінаційної схеми та пристрою керування потрібно визначити всі мікрооперації, які необхідно виконати для реалізації заданої операції та послідовність їх виконання. Це можна зробити, зокрема, шляхом переходу від просторового графа алгоритму виконання операції до часового графа цього алгоритму, як це показано на рис. 7.14. На основі представленого на рис. 7.14а просторового графа алгоритму можна визначити кількість та типи функціональних операторів, які мають бути виконані багатофункціональною комбінаційною схемою та послідовність їх виконання шляхом зведення графа до однієї вершини ФО = {ФОі, і = 1,2,...,7} (рис. 7.14Ь). Тобто часовий граф будується шляхом об'єднання всіх функціональних операторів алгоритму та їх зв'язків.



а) *Ь)*

Рис. 7.14. Перехід від просторового графа алгоритму до часового

Просторовий граф алгоритму складається з функціональних операторів, яким відповідають обчислювальні операції, та дуг, якими здійснюється передача даних для обробки. Часовий граф алгоритму складається з однієї вершини, яка послідовно виконує операції просторового графу. Даний граф характеризується наявністю вхідних та вихідних вузлів, які є незалежними входами (виходами) вершини графу. Подавання вхідних даних та видача вихідних даних здійснюється у певні проміжки часу (такти). Вхідні дані X_1 та X_2 поступають на 1 та 2 такті, а видача вихідних результатів $У_1$ та $У_2$ відбувається на 8 та 9 тактах. На вхідних та вихідних дугах графу зображені числа, які вказують порядок спрацювання дуги в часі. Дуги, біля яких записано кілька цифр, спрацьовують більше одного разу. Трикутні вершини із вказаними значеннями затримки даних відповідають за здійснення одночасної подачі даних з вихідного вузла до вхідних вузлів вершини графу. Трикутні вершини, які мають на виході вузли, пропускають більше одного даного.

Тоді для побудови багатотактового операційного пристрою потрібно синтезувати багатофункціональну комбінаційну схему, яка б дозволяла виконати всі функціональні оператори ФО, а пристрій керування повинен задати тип виконуваного в конкретний момент часу функціонального оператора та забезпечити пересилання на входи багатофункціональної комбінаційної схеми потрібних в цей момент даних.

Багатотактові операційні пристрої характеризуються малими затратами обладнання та, відповідно, невисокою продуктивністю.

Питання побудови багатотактового операційного пристрою в цілому формалізовані і детально розглянуті в літературі.

7.11. Однотактовий операційний пристрій

Побудова однотактових операційних пристроїв передбачає повністю апаратне відображення просторового графа виконуваного алгоритму комбінаційними схемами, які виконують функціональні оператори алгоритму і з'єднані між собою відповідно до графа алгоритму, як це показано на рис. 7.15.

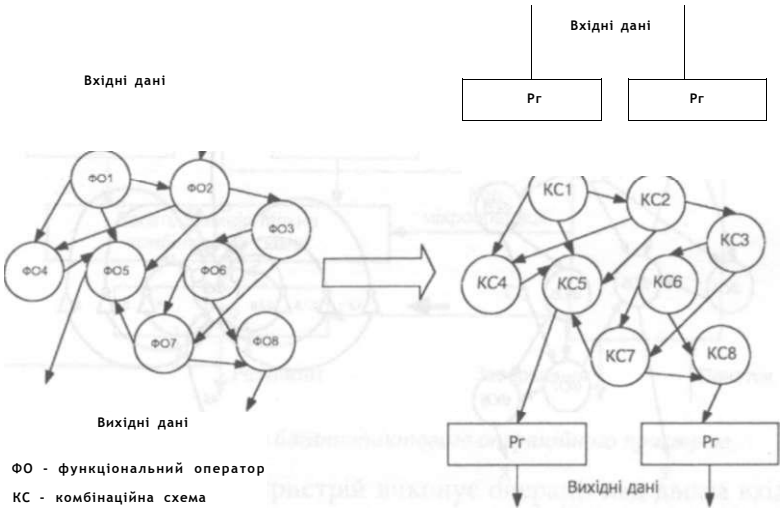


Рис. 7.15. Просторовий граф алгоритму та його апаратне відображення

Кожному функціональному оператору алгоритму ФО поставлена у відповідність комбінаційна схема КС, яка його виконує. На вході та виході операційного пристрою включені регістри.

Часова затримка в однотактовому операційному пристрої $T_{оп}$ визначається сумою затримок і. комбінаційних елементів, що лежать на найдовшому шляху проходження сигналу

$$i \qquad I$$

Цей шлях (їх може бути декілька) називається критичним шляхом.Обчислення часу виконання алгоритму в однотактовому операційному пристрої зводиться, таким чином, до знаходження критичного шляху.

Затрати обладнання $\backslash У_{оп}$ на однотактовий операційний пристрій визначаються сумою затрат обладнання \У". на реалізацію комбінаційних схем КС, тобто

$$i$$

а також на вхідні та вихідні регістри.

7.12. Конвеєрний операційний пристрій

Конвеєрний принцип обробки передбачає суміщення в часі виконання операторів алгоритму над різними даними. Одним із можливих підходів тут є конвеєризація однотактових ОП, структура яких базується на апаратному відображенні потокового графа виконуваного алгоритму. На рис. 7.16 показано структуру конвеєрного операційного пристрою, який реалізує потоковий граф алгоритму, наведений на рис. 7.6.

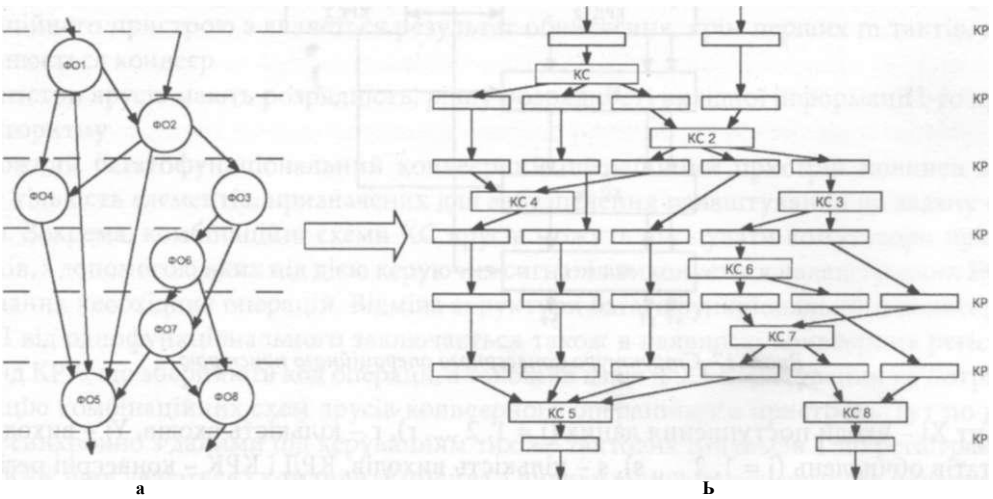


Рис. 7.16. Потоковий граф алгоритму а) та структура конвеєрного операційного пристрою, який його реалізує Ь)

Як ми вже вияснили, потоковим графом алгоритму називають таке представлення графа, яке передбачає розділення всіх його вершин по ярусах таким чином, що в і-му

ярусі розміщені тільки функціональні оператори, які залежать хоча б від одного функціонального оператора (і-І)-го яруса і не залежать від функціональних операторів, які не входять в яруси з меншими ніж і номерами. Всередині яруса функціональні оператори між собою не мають з'єднань. На рис. 7.16а штриховими лініями розділені яруси графа. При побудові конвеєрного операційного пристрою кожному функціональному оператору алгоритму поставлена у відповідність комбінаційна схема КС, яка його виконує, і, крім того, комбінаційні схеми, які виконують функціональні оператори ярусів потокового графа алгоритму, розділяються конвеєрними регістрами КР, як це показано на рис. 7.16Ь.

Таким чином, апаратне відображення алгоритму означає послідовне з'єднання комбінаційних схем, кожна з яких виконує операції одного яруса алгоритму. В конвеєрному операційному пристрої комбінаційні схеми з'єднані між собою через конвеєрні регістри.

Заданий алгоритм виконується над вхідними даними при їх однократному проходженні через конвеєрний операційний пристрій.

Найбільша довжина конвеєра такого конвеєрного операційного пристрою дорівнює кількості Р ярусів алгоритму. Такий конвеєрний операційний пристрій має найвищу частоту, яка може бути досягнута при апаратній реалізації заданого конкретизованого ПГ алгоритму.

Узагальнена структура конвеєрного операційного пристрою наведена на рис. 7.17.

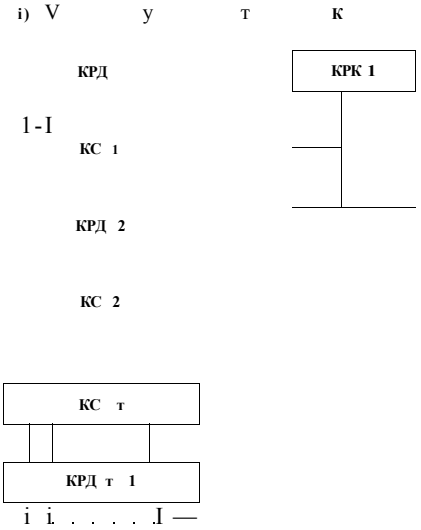


Рис. 7.17. Структура конвеєрного операційного пристрою

Тут Хі - входи поступлення даних (і = 1, 2 г), г - кількість входів, *Yj* - виходи результатів обчислень ()' = 1, 2 в), в - кількість виходів, КРД і КРК - конвеєрні регістри відповідно до даних Д і команд К, т - кількість ярусів ПГ алгоритму, п - кількість ярусів конвеєра конвеєрного операційного пристрою. Якщо структура конвеєрного операційного пристрою орієнтована на реалізацію алгоритму, який обчислює одну функцію, він називається однофункціональним, групу функцій - багатофункціональним. В багатофункціональному конвеєрному операційному пристрої програмуються канали передачі

інформації та функції, які виконуються комбінаційними схемами відповідно до графа багатофункціонального алгоритму

В однофункціональному конвеєрному операційному пристрої вихідні дані У =-.у, $у_2$, ,$у_5$ однозначно визначаються вхідними даними Х = х ^ , ,$х_г$ і функцією Ф, зашитою в його структурі: У = Ф(Х). В загальному випадку У = {$У_{кг}$}, Х - {$Х_г$}, тобто обробці підлягають матриці даних. При цьому к = 1, 2,..., в; *i* = 1, 2,..., г; *i* = 1, 2,..., N. де г, в - **кількість** входів і виходів конвеєрного операційного пристрою, N - кількість груп чисел, які поступають в конвеєрний операційний пристрій. Для кожного яруса однофункціонального конвеєрного операційного пристрою можна записати А = Ф (Аі), де Аі - вмістиме регістрів і-го яруса, Ф - операція, що визначається функціональними операторами (і+1)-го яруса ПГ алгоритму і виконується в (і+1)-му ярусі конвеєрного операційного пристрою. Так як $А_0$ = Х; А,- ФДА,,); $А_2$ = Ф^А^ і т. д., то У = $Ф_т$(Ф(...($Ф_1$(Х))...)), тобто функція Ф виконується над даними, які надходять в конвеєрний операційний пристрій, при їх проходженні через всі яруси конвеєра

Нехай в конвеєрному операційному пристрої обробляється масив чисел Х = {Х^ $Х_2$,..., Хг}, г = 1, 2,..., N. В першому такті тактовим імпульсом Т в регістр $КРД_1$ записується значення $Х_г$ Над ним в комбінаційній схемі $КС_1$ першого яруса виконується операція $Ф_г$ Другим тактовим імпульсом результати цієї операції перепишуться в регістр $КРД_2$, а в регістр $КРД_1$ запишеться значення $Х_2$. В комбінаційних схемах КС^ $КС_2$ над значеннями, що зберігаються відповідно в регістрах $КРД_1$ і $КРД_2$ виконуються операції $Ф_1$ і $Ф_2$. В третьому такті результати із $КС_2$ запишуться в $КРД_3$ і т. д. На регістри ярусів конвеєра, котрі звільняються, в кожному такті засилаються нові дані оброблюваного масиву, над якими виконуються ті ж операції. При повній загрузці конвеєра одночасно виконуються оператори всіх т ярусів алгоритму. Стан конвеєрного операційного пристрою в 1-му такті його роботи при обробці N чисел визначається вектором А(г) = |А (ОА (г)...А де I = 1, 2,..., (т + N0. При цьому $А_г$(і) = $У_і$, тобто в кожному такті на виході конвеєрного операційного пристрою з являється результат обчислення, крім перших т тактів, коли заповнюється конвеєр

Регістри ярусів мають розрядність, рівну розрядності вихідної інформації і-го яруса ПГ алгоритму

Кожний багатофункціональний конвеєрний операційний пристрій повинен мати деяку кількість елементів, призначених для забезпечення налаштування на задану операцію. Зокрема, комбінаційні схеми КС ярусів можуть вміщувати комутатори прямих зв язків, з допомогою яких під дією керуючих сигналів виконується налаштування КС на виконання необхідних операцій. Відміна структури багатофункціонального конвеєрного ОП від однофункціонального заключається також в наявності конвеєрних регістрів команд КРК, що зберігають код операції, а також зв язків для налаштування на потрібну операцію комбінаційних схем ярусів конвеєрного операційного пристрою. Тут по конвеєру синхронно з даними під керуванням тих же тактових імпульсів Т по регістрах команд КРК просуваються і команди. Команда з виходу відповідного регістра налаштовує комбінаційні схеми цього яруса на виконання необхідної операції. Для ярусів конвеєра в цьому випадку можна записати: $А_{.+1}$ = Ф^А., К.), де К. - вмістиме регістра КРК і-го яруса конвеєра. Виконувані в такому конвеєрному операційному пристрої перетворення над вхідними даними Х можна записати виразом

$$У = {}^{\Phi}{}_{т}({}^{к}{}_{т} - \Phi_{т}(к_{т}, \Phi_{т}(\ldots, \Phi(к, х)\ldots))).$$

7.13. Алгоритмічні операційні пристрої

7.13.1. Пристрої додавання і віднімання двійкових чисел з фіксованою комою

Пристрій для додавання двійкових чисел називається суматором. Суматор використовується в комп'ютері як складова частина АЛП, а також як елемент інших вузлів комп'ютера. Коротко розглянемо основні питання побудови суматора. На рис. 7.18 представлена схема однотактового алгоритмічного операційного пристрою (АОП) додавання і віднімання чисел з фіксованою комою з послідовною обробкою двійкових чисел.

Рис. 7.18. Послідовний АОП додавання і віднімання чисел з фіксованою комою

Тут вхідні операнди 1 та 2 записуються сигналами У1 та *У2* з пристрою керування до зсувних регістрів ЗРгІ та ЗРг2, виходи молодших розрядів яких з'єднані з входами однорозрядного суматора СМ, вихід суми 8. якого з'єднаний з входом старшого розряду зсувного регістра ЗРгЗ. Вихід переносу суматора з'єднаний з входом тригера Тг, який зберігає значення переносу С відповідного розряду протягом одного такту для подачі його на вхід однорозрядного суматора. Перед початком роботи вміст тригера Тг встановлюється в нуль. В кожному такті сигналом зсуву з пристрою керування вміст зсувних регістрів зсувається на один розряд вправо, а в тригер записується значення переносу з чергового розряду суми. Однорозрядний суматор працює відповідно до таблиці істинності з розділу 4 (табл. 4.5). Розряд суми з виходу однорозрядного суматора записується до зсувного регістра ЗРгЗ. Для додавання двох п-розрядних чисел в приведеному пристрої необхідно п тактів. Після цього на виході пристрою керування з'явиться сигнал завершення роботи, а в зсувному регістрі ЗРгЗ буде знаходитись результат операції.

Перевагою такого пристрою є простота та малі затрати обладнання на реалізацію суматора.

Значно частіше в комп'ютерах використовуються однотактові алгоритмічні ОП для додавання та віднімання двійкових чисел (рис. 7.19), в якому всі розряди операндів поступають на паралельний суматор з вхідних регістрів РгІ та Рг2 одночасно. Тим самим, за рахунок паралельної обробки досягається значно вища швидкодія порівняно з порозрядним додаванням операндів.

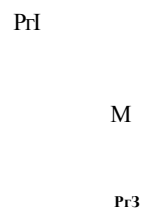


Рис. 7.19. Паралельний АОП для додавання та віднімання двійкових чисел

Однотактовий суматор будується шляхом апаратного відображення графа алгоритму додавання двійкових чисел, тобто шляхом заміни функціональних операторів алгоритму відповідними комбінаційними схемами.

Найпростішим з однотактових суматорів є однотактовий суматор, який апаратно відображає граф алгоритму додавання двійкових чисел з послідовним переносом і відповідно називається суматором з послідовним переносом. Схема цього суматора повторює граф відповідного алгоритму п-розрядного додавання двійкових чисел з послідовним переносом (рис. 4.5). Недоліком суматора з послідовним переносом є значна затримка при формуванні переносу, оскільки для отримання останнього розряду суми перенос повинен бути сформованим всіма попередніми однорозрядними суматорами.

Для зменшення кількості операцій, які знаходяться на критичному шляху, створено ряд алгоритмів додавання, в яких скорочено кількість послідовних операцій при формуванні переносу та відповідних структур суматорів. Це, зокрема, суматори з наскрізним, частково груповим і груповим переносами, в яких здійснюється паралельне формування переносів для всіх розрядів, або для груп розрядів. Досить простою та ефективною є реалізація в суматорі алгоритму за методом вибору переносу.

Оскільки питанню побудови ефективних алгоритмів додавання двійкових чисел приділено достатньо уваги в літературі, коротко розглянемо останній з названих вище підходів, тобто реалізацію алгоритму за методом вибору переносу. Розірвавши в довільному місці тракт проходження переносу та сформувавши два тракти його подальшого проходження із значенням вхідного переносу 0 та 1 відповідно, з послідуючим вибором результату за допомогою мультиплексора за значенням реального переносу в місці розриву, отримаємо показану на рис. 7.20 схему суматора.

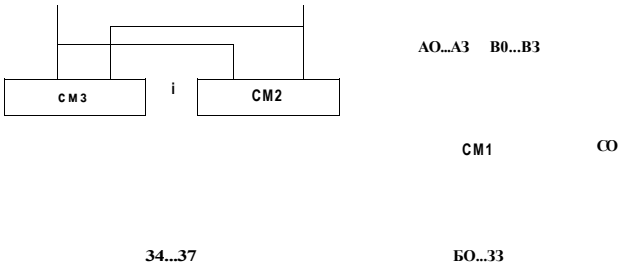


Рис. 7.20. Однотактовий суматор двійкових чисел за методом вибору переносу

Тим самим доволі просто вдалося в два рази скоротити критичний шлях формування переносу. Цей же підхід можна використати повторно і для прискорення роботи п/2-розрядних суматорів, як це показано на рис. 7.21 і т. д.

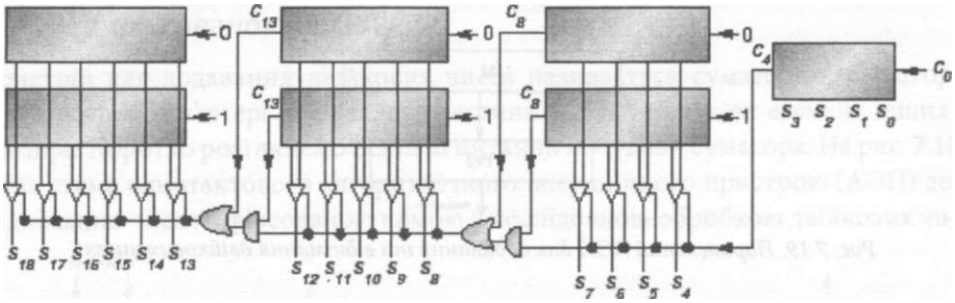


Рис. 7.21. Однотактовий суматор двійкових чисел за методом вибору переносу з чотирикратним прискоренням

Часто в комп'ютерах використовується пристрій для накопичення двійкових чисел, тобто для послідовного багатомісного додавання N чисел (рис. 7.22). В такому операційному пристрої є вхідний РгI та вихідний Рг2 регістри, а також суматор, причому один з входів суматора з'єднаний з виходом вихідного Рг2 регістра. Для забезпечення коректної роботи пристрою розрядність суматора повинна бути розширеною в сторону старших розрядів на]1о§.1М[бітів.

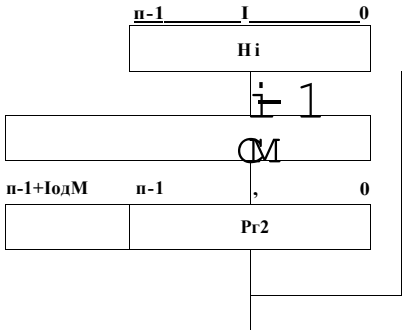


Рис. 7.22. АОП накопичення двійкових чисел

Для прискорення накопичення чисел використовують запам'ятовування не тільки сум з виходу суматора, а й переносів (так звані суматори з запам'ятовуванням переносів). В цьому випадку такт роботи пристрою в режимі накопичення чисел буде визначатися затримкою лише однорозрядного суматора.

Важливим питанням при побудові пристроїв додавання є фіксація переповнення. Для цього аналізується перенос із останнього розряду та знаки операндів. Для спрощення аналізу наявності переповнення використовуються так звані модифіковані коди з двома знаковими розрядами (0-00, 1-11). Незбіжність цих розрядів говорить про наявність переповнення.

Потрібно зауважити, що зазвичай для виконання віднімання використовується додавання, перед яким від'ємник перетворюється в обернений або доповняльний код. Пряме віднімання використовується рідко. Якщо ж така потреба є, то взамін суматора

використовується віднімач, правила побудови якого не відрізняються від правил побудови суматора, а для синтезу однорозрядного віднімана використовується залежність між входами та виходами відповідно до табл. 4.6. Якщо виконуються обидві операції - розробляється суматор-віднімач.

7.13.2. Пристрої множення двійкових чисел з фіксованою комою

Послідовне (багатотактове) множення базується на послідовному утворенні суми часткових добутків, використовуючи один або декілька суматорів, і реалізується шляхом послідовного виконання операцій додавання.

Як ми вже бачили в розділі 4, процес множення може починатися з молодших і старших розрядів множника. При цьому повну суму часткових добутків (тобто добуток) можна отримати двома шляхами:

- зсувом множеного на потрібну кількість розрядів і додаванням отриманого чергового часткового добутку до раніше накопиченої суми;
- зсувом суми раніше отриманих часткових добутків на кожному кроці на один розряд і наступним додаванням нерухомого множеного або 0 до зсунутої суми.

Таким чином, існує 4 методи множення двійкових чисел, на основі яких можна побудувати 4 алгоритми ітераційного виконання цієї операції та 4 базових структури багатотактових АОП множення двійкових чисел:

- множення починаючи з молодших розрядів множника зі зсувом суми часткових добутків вправо;
- множення починаючи з молодших розрядів множника зі зсувом множеного вліво;
- множення починаючи з старших розрядів множника зі зсувом суми часткових добутків вліво;
- множення починаючи з старших розрядів множника зі зсувом множеного вправо.

Розглянемо їх детальніше.

7.13.2.1. Багатотактовий пристрій множення двійкових чисел з молодших розрядів множника при нерухомому множеному з зсувом суми часткових добутків

Алгоритм множення двійкових чисел, який реалізує метод множення починаючи з молодших розрядів множника з зсувом суми часткових добутків вправо, описується наступним ітераційним виразом:

$$\_(г_i + x \cdot г_0)$$

$$дег_a = 0; \qquad \mathbf{i = 0, и - 1;} \qquad 2_n = 2 = XX.$$

Тут вжито наступні позначення: Х, У, *7.* - множене, множник і добуток відповідно, 2. - сума часткових добутків на і-му етапі, У(і) - і-й розряд множника, п - кількість розрядів операндів без врахування знакового розряду.

В кожному циклі множене додається до суми часткових добутків, якщо У. = 0, і не додається, якщо У. = 0, після чого сума часткових добутків множиться на 2 ^{i}, тобто зсувається на один розряд вправо. Після закінчення п-го циклу утворюється шуканий добуток, тобто $T_n = 2 =$ ХУ

Алгоритм можна представити блок-схемою, показаною на рис. 7.23.

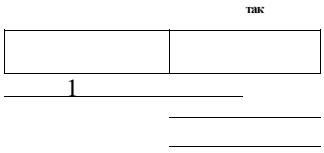


Рис. 7.23. Блок-схема алгоритму множення першим методом

Приклад:

Необхідно помножити два числа (без знакового розряду):

X = 0101 0101; У = 01101011;

Хід операцій проілюстровано в табл. 7.3:

Таблиця 7.3

i	*г*	У(У(0)	Х.У(I)	*2i+* Х.У(i)	2-,
0	0000 0000 0000 0000	ОНО 1011	0101 0101	0101 0101 0000 0000	0010 1010 1000 0000
1	0010 1010 1000 0000	ОНО 1011	0101 0101	0111 1111 1000 0000	ООП 1111 1100 0000
2	ООП 1111 1100 0000	ОНО 1011	0000 0000	ООП 1111 1100 0000	0001 1111 1110 0000
3	0001 1111 1110 0000	ОНО 1011	0101 0101	0111 0100 11100000	ООП 1010 0111 0000
4	ООП 1010 0111 0000	ОНО 1011	0000 0000	ООП 10100111 0000	0001 1101 ООП 1000
5	0001 1101 ООП 1000	ОНО 1011	0101 0101	0111 0010 0011 1000	ООП 1001 0001 1100
6	ООП 1001 0001 1100	ОНО 1011	0101 0101	1000 1110 0001 1100	0100 0111 0000 1110
7	0100 0111 0000 1110	ОНО 1011	0000 0000	0100 0111 0000 1110	0010 0011 1000 0111

Таким чином 0101 0101 . 01101011 = 0010 ООП 1000 0111.

Базова структура багатотактового АОП множення двійкових чисел за описаним методом наведена на рис. 7.24.

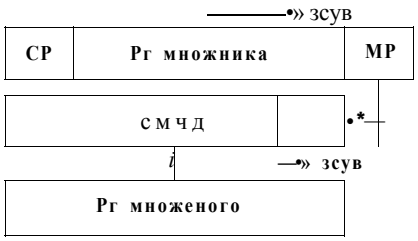


Рис. 7.24. Базова структура багатотактового АОП множення першим методом

Тут СМЧД - суматор часткових добутків. Множник зберігається в регістрі множника, а множене - в регістрі множеного. Обидва ці регістри є п-розрядними. Суматор част-

кових добутків є накопичувальним суматором, тобто на його виході є регістр з оберненим зв'язком, як це показано на рис. 7.22. В кожному такті вміст цього регістра та регістра множника зсуваються на один розряд вправо в сторону молодших розрядів. Розряд в крайньому правому тригері регістра множника випадає, а на його місце поміщається наступний розряд множника, який керує операцією СМЧД, тобто вказує, чи є в даному такті додавання, чи його немає. При зсуві молодший розряд СМЧД може випадати, а може і зберігатися, залежно від того, якої розрядності потрібен результат - п-розрядний, чи 2п-розрядний. Завдяки тому, що в даній структурі регістри множеного і множника є п-розрядними, а не 2п-розрядними, як це є в базових структурах АОП множення іншими методами, перший метод множення використовується найчастіше.

7.13.2.2. Багатотактовий пристрій множення двійкових чисел з молодших розрядів при нерухомій сумі часткових добутків з зсувом множеного вліво

Алгоритм множення двійкових чисел, який реалізує цей метод, описується наступними ітераційними виразами:

Ь с , = 2 1 ,

деX_0=X; 2_0=0; / = 0,я-1; *$2_п$=2 = XУ.*

Тут вжито наступні позначення: Х, У, Ъ - мцожене, множник і добуток відповідно, Ъ - сума часткових добутків на і-му етапі, У(і) - і-й розряд множника, п - кількість розрядів операндів без врахування знакового розряду.

Алгоритм можна представити блок-схемою, показаною на рис. 7.25.

С

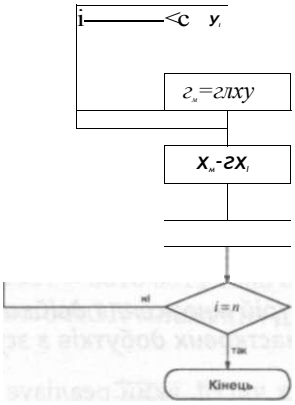


Рис. 7.25. Блок-схема алгоритму множення другим методом

Приклад:

Необхідно помножити два числа (без знакового розряду):

Х=0101 0101; ¥=01101011.

Хід операцій проілюстровано в табл. 7.4.

Таблиця 7.4

і	*7Л*	У (У(і))	Х	гі+і= гі+х.у(і)
0	0000 0000 0000 0000	0110 1011	0000 0000 0101 0101	0000 0000 0101 0101
1	0000 0000 0101 0101	0110 1011	0000 0000 1010 1010	0000 0000 1111 1111
2	0000 0000 1111 1111	01101011	0000 0001 0101 0100	0000 0000 1111 1111
3	0000 0000 1111 1111	0110 1011	0000 0010 1010 1000	0000 0011 1010 0111
4	0000 0011 1010 0111	оно 1011	0000 0101 0101 0000	0000 0011 10100111
5	0000 0011 1010 0111	0110 1011	0000 1010 1010 0000	0000 1110 0100 0111
6	0000 1110 0100 0111	0110 1011	0001 0101 0100 0000	00100011 10000111
7	0010 0011 1000 0111	0110 1011	0010 1010 1000 0000	0010 0011 1000 0111

Таким чином 0101 0101 • 01101011 = 0010 ООП 1000 0111.

Базова структура багатотактового АОП множення двійкових чисел за описаним методом наведена на рис. 7.26.

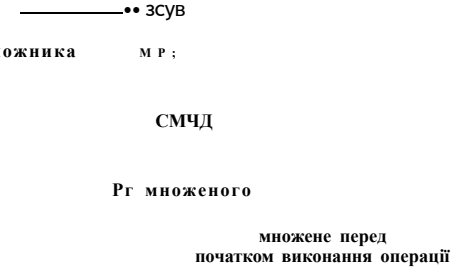


Рис. 7.26. Базова структура багатотактового АОП множення другим методом

Тут СМЧД - суматор часткових добутків. Множник зберігається в регістрі множника, а множене - в регістрі множеного. Перший з цих регістрів є п-розрядним, а другий . - 2п-розрядним. Суматор часткових добутків є накопичувальним суматором, тобто на його виході є регістр з оберненим зв'язком як це показано на рис. 7.22, який також є 2п-розрядним. Перед початком виконання операції множене знаходиться в правій частині регістра множеного. В кожному такті вміст регістра множеного зсувається на один розряд вліво в сторону старших розрядів, а вміст регістра множника в кожному такті зсувається на один розряд вправо в сторону молодших розрядів. Розряд в крайньому правому тригері регістра множника випадає, а на його місце поміщається наступний розряд множника, який керує операцією СМЧД, тобто вказує чи є в даному такті додавання, чи його немає. В порівнянні з попередньою структурою тут регістр множеного та СМЧД обов'язково мають бути 2п-розрядними.

7.73.2.3. *Багатотактовий пристрій множення двійкових чисел з старших розрядів при нерухомій сумі часткових добутків з зсувом множеного вправо*

Алгоритм множення двійкових чисел, який реалізує цей метод, описується наступними ітераційними виразами:

$$де X_i = X; \quad г_0 = 0; \quad i = 0, и-1; \quad 2=2 = XY.$$

Тут вжито наступні позначення: X, У, Ъ - множене, множник і добуток відповідно, *7л* - сума часткових добутків на і-му етапі, У(п-і-І) - (п-і-І)-й розряд множника, п - кількість розрядів операндів без врахування знакового розряду.

Алгоритм можна представити блок-схемою, показаною на рис. 7.27.

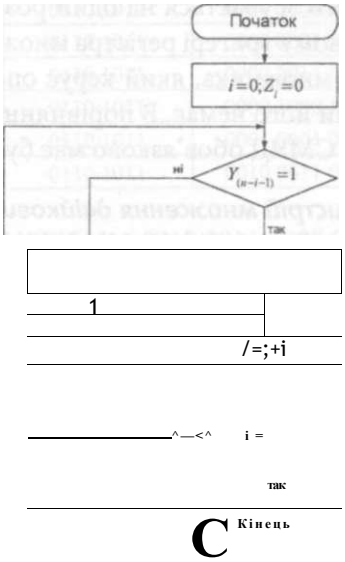


Рис. 7.27. Блок-схема алгоритму множення третім методом

Приклад:

Необхідно помножити два числа (без знакового розряду):

X=01010101; ¥=01101011.

Хід операцій проілюстровано в табл. 7.5.

Таблиця 7.5

і	*г*	У (У(п-Ї-І))	Хі+1	***Ti*+|=*Ti*+** Хі+1.У(п-і-1)
0	0000 0000 0000 0000	0110 1011	0010 1010 1000 0000	0000 0000 0000 0000
1	0000 0000 0000 0000	0110 1011	0001 0101 0100 0000	0001 0101 0100 0000
2	0001 0101 0100 0000	0110 1011	0000 1010 1010 0000	0001 1111 1110 0000
3	0001 1111 1110 0000	0110 1011	0000 0101 0101 0000	0001 1111 1110 0000
4	0001 1111 11100000	0110 1011	0000 0010 1010 1000	0010 0010 1000 1000
5	0010 0010 1000 1000	0110 1011	0000 0001 0101 0100	0010 0010 1000 1000
6	0010 0010 1000 1000	01101011	0000 0000 1010 1010	0010 0011 ООП 0010
7	0010 0011 ООП 0010	01101011	0000 0000 0101 0101	0010 0011 1000 0111

Таким чином 0101 0101 .01101011 = 0010 0011 1000 0111.

Базова структура багатотактового АОП множення двійкових чисел за описаним методом наведена на рис. 7.28.

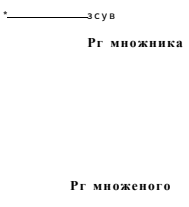


Рис. 7.28. Базова структура багатотактового АОП множення третім методом

Тут СМЧД - суматор часткових добутків. Множник зберігається в регістрі множника, а множене - в регістрі множеного. Обидва ці регістри є п-розрядними. Суматор часткових добутків є накопичувальним суматором, тобто на його виході є регістр з оберненим зв'язком як це показано на рис. 7.22, який є 2п-розрядним. В кожному такті вміст регістрів множника та множеного зсувається на один розряд вліво в сторону старших розрядів. Розряд в крайньому лівому тригері регістра множника випадає, а на його місце поміщається наступний розряд множника, який керує операцією СМЧД, тобто вказує чи є в даному такті додавання, чи його немає. В порівнянні з базовою структурою АОП множення першим методом тут СМЧД обов'язково має бути 2п-розрядним.

7.13.2.4. Багатотактовий пристрій множення двійкових чисел з старших розрядів при нерухомому множеному з зсувом суми часткових добутків вліво

Алгоритм множення двійкових чисел, який реалізує цей метод, описується наступним ітераційним виразом:

$2_м$ - 2 • 2_i + X • У(„_,_),

$деr_o=0;$ / = 0,/i-1; г . ^ ^ I T .

Тут вжито наступні позначення: Х, У, Ъ - множене, множник і добуток відповідно, *7.* - сума часткових добутків на і-му етапі, $У_{..1)}$ - (п-і-І)-й розряд множника, п - кількість розрядів операндів без врахування знакового розряду.

Алгоритм можна представити блок-схемою, показаною на рис. 7.29.

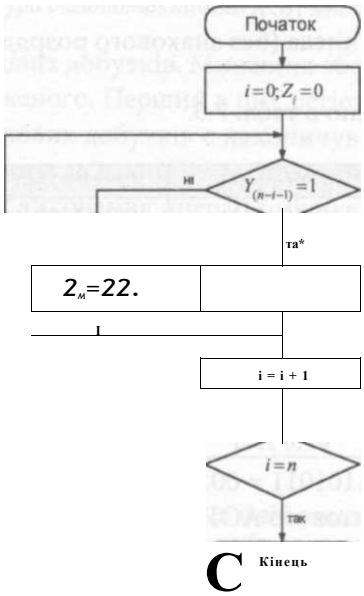


Рис. 7.29. Блок-схема алгоритму множення четвертим методом

Приклад:

Множимо два числа (без знакового розряду):

Х=01010101; У=01101011.

Хід операцій проілюстровано в табл. 7.6.

Таблиця 7.6

і	гі	У(У(п-і-1))	2.гі	гі+і=2.гі+ ХІ+І-У(**П**-І-І)
0	0000 0000 0000 0000	0110 1011	0000 0000 0000 0000	0000 0000 0000 0000
1	0000 0000 0000 0000	0110 1011	0000 0000 0000 0000	0000 0000 0101 0101
2	0000 0000 0101 0101	0110 1011	0000 0000 1010 1010	0000 0000 1111 1111
3	0000 0000 1111 1111	0110 1011	0000 0001 1111 1110	0000 0001 1111 1110
4	0000 0001 1111 1110	0110 1011	0000 0011 1111 1100	0000 0100 0101 0001
5	0000 0100 0101 0001	0110 1011	0000 1000 1010 0010	0000 1000 1010 0010
6	0000 1000 1010 0010	0110 1011	0001 0001 0100 0100	0001 0001 1001 1001
7	0001 0001 1001 1001	0110 1011	00100011 0011 0010	0010 0011 1000 0111

Таким чином 0101 0101 • 01101011 = 0010 0011 1000 0111.

Базова структура багатотактового АОП множення двійкових чисел за описаним методом наведена на рис. 7.30.

Рис. 7.30. Базова структура багатотактового АОП множення четвертим методом

Тут СМЧД - суматор часткових добутків. Множник зберігається в регістрі множника, а множене - в регістрі множеного. Перший з цих регістрів є п-розрядним, а другий - 2п-розрядним. Суматор часткових добутків є накопичувальним суматором, тобто на його виході є регістр з оберненим зв'язком як це показано на рис. 7.22, який також є 2п-розрядним. Перед початком виконання операції множене знаходиться в лівій частині регістра множеного. В кожному такті вміст регістра множеного та вміст СМЧД зсуваються на один розряд вправо в сторону молодших розрядів. Розряд в крайньому лівому тригері регістра множника випадає, а на його місце поміщається наступний розряд множника, який керує операцією СМЧД, тобто вказує чи є в даному такті додавання, чи його немає. В порівнянні з базовою структурою АОП множення першим методом тут, як в базовій структурі АОП множення другим методом, регістр множеного та СМЧД обов'язково мають бути 2п-розрядними.

В усіх чотирьох розглянутих структурах АОП множення двійкових чисел час виконання операції $I_м = пі_см$ де І - затримка СМЧД.

7.13.2.5. Багатотактовий пристрій прискореного множення

Одним із методів прискореного множення є одночасний аналіз декількох розрядів множника. Це може бути одночасний аналіз двох, трьох і більшої кількості розрядів. Для пояснення суті методу на рис. 7.31 показана схема багатотактового пристрою множення

з одночасним аналізом двох розрядів множника. Тут схема аналізу СА проводить аналіз двох розрядів множника і вказує СМЧД тип виконуваної операції: додавання відсутнє, додається значення множеного, додається подвоєне значення множеного, додається потроєне значення множеного. Зсув в регістрах відбувається одночасно на два розряди.

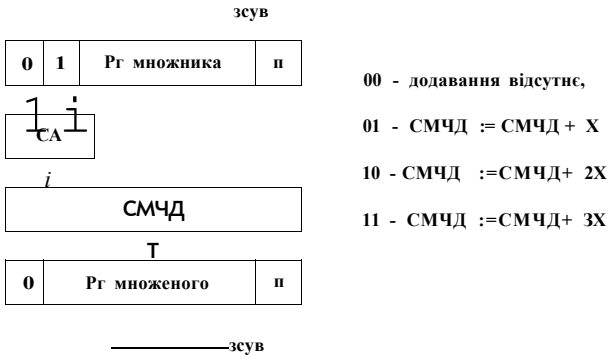


Рис. 7.31. Схема багатотактового пристрою множення з одночасним аналізом двох розрядів множника

Подібним чином може бути реалізований пристрій множення з одночасним аналізом двох розрядів множника з застосуванням всіх чотирьох вище розглянутих методів множення. При множенні чисел, представлених в доповняльному коді, доцільно використовувати пристрій, який реалізує алгоритм Бута, розглянутий в розділі 4.

Часто застосовуються асинхронні пристрої множення з одночасним аналізом всіх розрядів множника. В таких пристроях кількість тактів визначається кількістю одиниць в множнику.

Приклад:

X= 101001110001

¥ = 110000111110.

На позиціях нулів у множнику У додавання не виконується, лише зсув на відповідну кількість розрядів.

Множник У можна представити в вигляді У = 1100010000(-1)0. Тоді взамін 7 додавань необхідно виконати 3 додавання і одне віднімання. Особливо ефективний цей метод при виконанні множення на константи.

7.73.2.6. *Однотактові пристрої множення двійкових чисел з фіксованою комою*

Як вже було показано, побудова однотактових операційних пристроїв передбачає апаратне відображення просторового графа алгоритму виконання операції комбінаційними схемами, які виконують функціональні оператори алгоритму і з'єднані між собою відповідно до графа алгоритму. Тому структура однотактового пристрою множення двійкових чисел з фіксованою комою повторить відповідну структуру графа алгоритму, наведеного на рис. 4.7, як це показано на рис. 7.32.

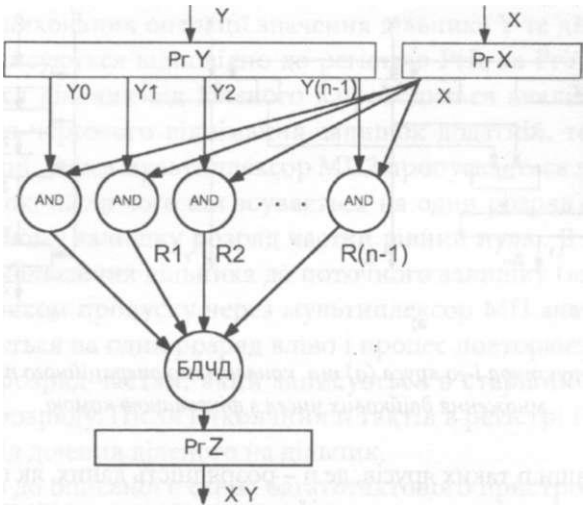


Рис. 7.32. Структура однотактового пристрою множення двійкових чисел з фіксованою комою

Тут вхідні дані X та Y поступають в регістри PrX та PrY, а з них на пристрої логічного множення AND, на яких формуються логічні добутки множеного X на розряди множника Y. Ці логічні добутки з зсувом на відповідну кількіть розрядів поступають на входи комбінаційної схеми багатомісного додавання часткових добутків БДЧД, результат множення з якої поступає в регістр PrZ, а з нього на вихід пристрою.

Комбінаційна схема багатомісного додавання часткових добутків БДЧД реалізує алгоритми, детально розглянуті в п. 4.4.4.2, де кожному оператору двомісного однорозрядного двійкового додавання має бути поставлений у відповідність однорозрядний суматор двійкових чисел, який реалізує логічні вирази відповідно до табл. 4.5.

7.13.2.7. Конвеєрні пристрої множення двійкових чисел з фіксованою комою

При побудові конвеєрного операційного пристрою множення двійкових чисел з фіксованою комою кожному функціональному оператору алгоритму ставиться у відповідність комбінаційна схема, яка його виконує, і, крім того, комбінаційні схеми, які реалізують функціональні оператори ярусів потокового графа алгоритму, розділяються конвеєрними регістрами. Алгоритм множення виконується над вхідними даними при їх однократному проходженні через конвеєрний операційний пристрій.

Якщо вибрати для реалізації граф алгоритму послідовного попарного додавання часткових добутків, отриманих починаючи з аналізу молодших розрядів множника, який представлений на рис. 4.8, то структура і-го яруса конвеєрного операційного пристрою множення двійкових чисел з фіксованою комою буде мати вигляд, показаний рис. 7.33.

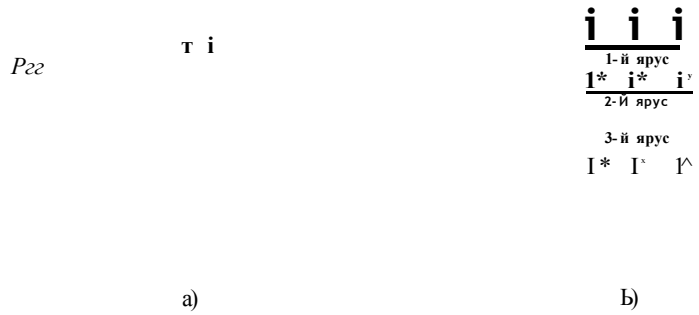


Рис. *7.33. Структура і-го яруса (а) та конвеєрного операційного пристрою (Ь) множення двійкових чисел з фіксованою комою*

Послідовно з'єднавши п таких ярусів, де п - розрядність даних, як показано на рис. 7.33Ь, отримаємо структуру конвеєрного операційного пристрою множення двійкових чисел з фіксованою комою.

Аналогічним чином можна побудувати конвеєрні операційні пристрої множення двійкових чисел з фіксованою комою на основі операторів попарного п-розрядного додавання двох чисел відповідно до інших алгоритмів множення, розглянутих в п. 4.4.4.2 розділу 4.

Не є складною і побудова потокового графа алгоритму паралельного матричного виконання багатомісної операції додавання часткових добутків, наприклад з діагональним розповсюдженням переносу відповідно до рис. 4.12, а також реалізація відповідного конвеєрного пристрою багатомісного додавання часткових добутків.

7.13.3 Пристрої ділення двійкових чисел з фіксованою комою

7.13.3.1. Багатотактові пристрої ділення двійкових чисел з фіксованою комою

Як це вже було показано в розділі 4, існує два основних варіанти виконання операції ділення: з зсувом залишків вліво та з зсувом дільника. Для реалізації АОП перший варіант вигідніший, так як вимагає використання п-розрядного віднімача, тоді як другий варіант вимагає використання 2п-розрядного віднімача. При цьому перший варіант може бути виконаний двома способами: з відновленням і без відновлення залишку. Схема багатотактового пристрою ділення за алгоритмом з відновленням залишку, який працює відповідно до блок-схеми, наведеної на рис. 4.15, показана на рис. 7.34.

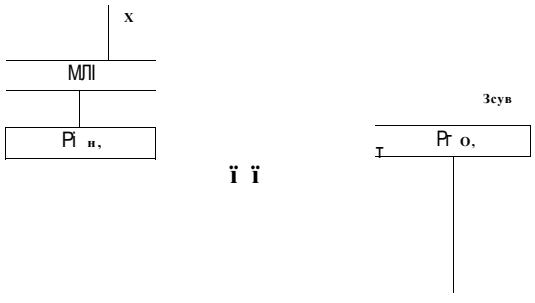


Рис. 7.34. Структура багатотактового АОП ділення двійкових чисел з відновленням залишку

Перед початком виконання операції значення дільника У та діленого Х (через мультиплексор МП1) записуються відповідно до регістрів РгД. та РгУ. В кожному такті послідовно віднімається дільник від діленого і проводиться аналіз значення поточного залишку. Якщо після чергового віднімання залишок додатній, то відповідний розряд частки рівний одиниці. Через мультиплексор МП2 пропускається значення з виходу віднімача, тобто залишок, після чого він зсувається на один розряд вліво і процес повторюється. При від'ємному залишку розряд частки рівний нулю. В цьому випадку виконується коригуюче збільшення дільника до поточного залишку (відновлення залишку), що здійснюється шляхом пропуску через мультиплексор МП значення з регістра РгК, після чого він зсувається на один розряд вліво і процес повторюється. В кожному такті визначається один розряд частки, який записується в старший розряд регістру РгС^. на місце зсунутого розряду. Після виконання п тактів в регістрі РгО.буде знаходитись п-розрядна частка від ділення діленого на дільник.

Досить подібною до описаної є схема багатотактового пристрою ділення без відновлення залишку, представлена на рис. 7.35.

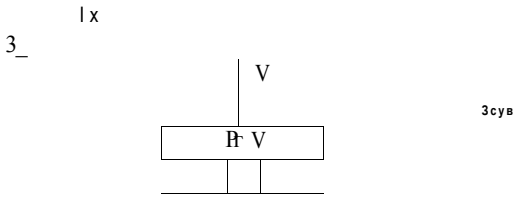


Рис. 7.35. Структура багатотактового АОП ділення двійкових чисел без відновлення залишку

Як і в попередньо розглянутому пристрої, перед початком виконання операції значення дільника У та діленого Х записуються відповідно до регістрів РгУ та РгК\. В кожному такті залежно від значення розряду частки, отриманого на попередньому такті, через мультиплексор МП на суматор СМ проходить прямий або інверсний код дільника, і тим самим дільник додається або віднімається від діленого. Якщо після чергової операції додавання або віднімання залишок додатній, то відповідний розряд частки рівний одиниці, при від'ємному залишку розряд частки рівний нулю. Після виконання операції значення з виходу суматора зсувається на один розряд вліво і процес повторюється. В кожному такті визначається один розряд частки, який записується в старший розряд регістру РгС}. на місце зсунутого розряду. Після виконання п тактів в регістра Рг(}. буде знаходитись п-розрядна частка від ділення діленого на дільник.

В обох розглянутих пристроях час виконання ділення дорівнює Тд = п ($г_{мп} + i_{см} + i_{рг}$), де складові суми є затримками в мультиплексорі, суматорі та регістрі відповідно.

Потрібно відзначити, що досить близькими до розглянутих алгоритмів і пристроїв ділення є алгоритми і пристрої добування квадратного кореня.

7.13.3.2. Однотактові та конвеєрні пристрої ділення двійкових чисел з фіксованою комою

Подібно до операції множення, побудова однотактових операційних пристроїв ділення передбачає повністю апаратне відображення просторового графа алгоритму виконання операції комбінаційними схемами, які виконують функціональні оператори алгоритму і з'єднані між собою відповідно до графа алгоритму. Тому структура однотактового пристрою множення двійкових чисел з фіксованою комою повторить відповідну структуру графа алгоритму, наведеного на рис. 4.16.

При побудові конвеєрного операційного пристрою ділення двійкових чисел з фіксованою комою кожному функціональному оператору алгоритму ставиться у відповідність комбінаційна схема, яка його виконує, і, крім того, комбінаційні схеми, які реалізують функціональні оператори ярусів потокового графа алгоритму, розділяються конвеєрними регістрами. Алгоритм ділення виконується над вхідними даними при їх однократному проходженні через конвеєрний операційний пристрій.

Якщо вибрати для реалізації граф алгоритму ділення двійкових чисел з відновленням залишку, який представлений на рис. 4.16, то і-й ярус конвеєрного операційного пристрою ділення двійкових чисел з фіксованою комою буде мати вигляд, показаний на рис. 7.36.

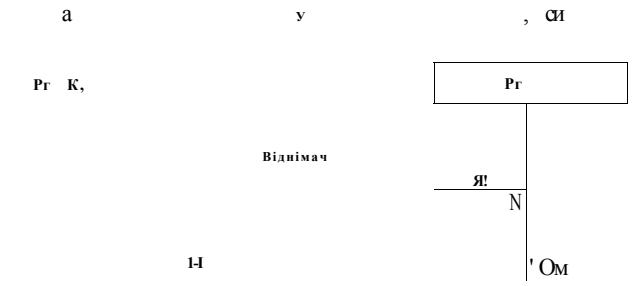


Рис. 7.36. і-й ярус конвеєрного операційного пристрою ділення двійкових чисел з фіксованою комою за алгоритмом з відновленням залишку

Якщо вибрати для реалізації граф алгоритму ділення двійкових чисел без відновлення залишку, то структура і-го яруса конвеєрного операційного пристрою ділення двійкових чисел з фіксованою комою буде мати вигляд, показаний на рис. 7.37а.

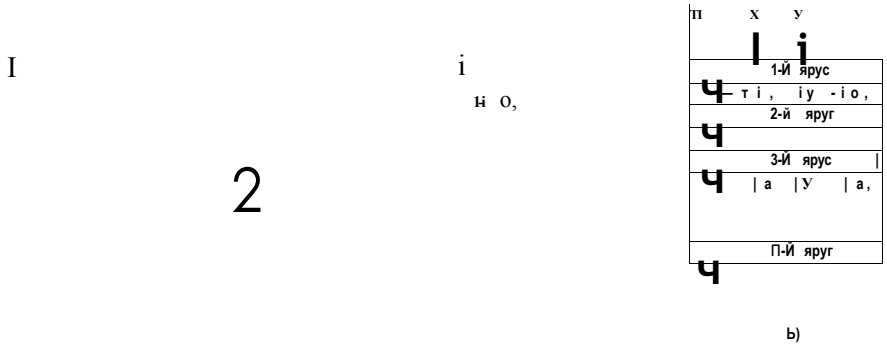


Рис. 7.37. Структура і-го яруса (а) та конвеєрного операційного пристрою (Ь) ділення двійкових чисел з фіксованою комою за алгоритмом без відновлення залишку

Послідовно з'єднавши п таких ярусів, де п - розрядність частки, як показано на рис. 7.37Ь, отримаємо структуру конвеєрного операційного пристрою ділення двійкових чисел з фіксованою комою за алгоритмом без відновлення залишку.

7.13.4. Пристрої обчислення елементарних функцій методом "цифра за цифрою"

7.13.4.1. Багатотактовий пристрій обчислення елементарних функцій методом "цифра за цифрою"

В системі команд сучасних комп'ютерів присутня велика кількість команд обчислення елементарних функцій типу exp X, In X, Sin X, Cos X, Sh X, Ch X, піднесення до степеня Am; arctg y/x тощо. Виконання цих команд на універсальному АЛП, яке виконує елементарні команди, вимагає значних витрат часу. Навіть реалізація на універсальному АЛП досить простого за складом базових операцій методу "цифра за цифрою" не дає відчутного виграшу в швидкодії внаслідок його специфіки, що знайшла віддзеркалення, наприклад, в необхідності виконання зсувів на змінне число розрядів. Тому в ряді сучасних комп'ютерів до складу АЛП вводять операційні пристрої для обчислення елементарних функцій.

Багатотактовий АОП (рис. 7.38), що реалізовує метод "цифра за цифрою" відповідно до ітераційних рівнянь, наведених в п. 4.5.2, містить:

РгХ, PrY, PrZ, РгС - регістри для зберігання початкових значень X_0, Y_0, Z_0, та констант С, а також результатів проміжних обчислень X., Y, Z..

С31, С32 - схеми зсуву на і розрядів (i=1,2,...,п);

СВ1, СВ2, СВ3 - суматори-віднімачі;

ПЗП - постійний запам'ятовуючий пристрій для зберігання констант;

МП - мультиплексор.

Зн PrY

Рис. 7.38. Структура багатотактового АОП обчислення елементарних функцій методом "цифра за цифрою"

Будь-яка з елементарних функцій може бути реалізована на даній структурі за час $T = n (i_{св} + i_{сз} + i_{рг}$, де $т_{св}$ - час затримки на суматорі-віднімачі, $i_{сз}$ - час затримки в схемі зсуву, $I_г$ - час запису даних до регістра, п - розрядність операндів. Окрім розглянутої

тут структури з паралельною обробкою даних, можуть бути реалізовані менш швидкодіючі структури пристроїв з порозрядною обробкою, а також багато варіантів проміжних структур.

До основних переваг даного методу з точки зору можливостей його реалізації в АОП належать:

- простота обчислювальних алгоритмів, що базуються лише на трьох операціях: додавання/віднімання, зсув і вибірка з ПЗП;
- однотипність обчислювальних алгоритмів для обчислення майже всіх елементарних функцій;
- однотипність виконання кожної ітерації;
- можливість побудови багатофункціональних АОП, що використовують даний метод;
- похибки цього методу достатньо повно досліджені та легко компенсуються шляхом введення додаткових розрядів.

7.Т3.4.2. ОсЗнотактовіш *та конвеєрний операційні пристрої обчислення елементарних функцій методом "цифра за цифрою"*

Постійність констант і кількості розрядів зсуву на кожній ітерації дозволяє при реалізації однотактового операційного пристрою видалити ПЗП для зберігання констант і схеми зсуву, які необхідні в багатотактових операційних пристроях (рис. 7.38).

Зсуви здійснюються шляхом відповідного з'єднання розрядів операційних елементів, а константи формуються на вході операційного елементу шляхом подачі логічного нуля або одиниці у відповідний розряд. Таким чином, для реалізації однієї ітерації алгоритму "цифра за цифрою" необхідно виконати одну операцію віднімання і дві операції додавання, якщо оператор *і.* має значення -1, або одну операцію додавання і дві операції віднімання, якщо оператор (має значення +1. Операції додавання і віднімання кодів чисел можна виконувати використовуючи комбінаційну схему суматора-віднімача. При виконанні віднімання необхідно стежити, щоб від'ємник X, У або Ъ поступав на другий вхід суматора-віднімача. Як показав аналіз, при використанні суматорів-віднімачів обробку даних в пристрої найвигідніше виконувати в доповняльному коді. Режим роботи суматора-віднімача забезпечується сигналом керування 0, якщо виконується операція додавання, і 1, якщо виконується операція віднімання.

Структура і-го яруса конвеєрного операційного пристрою для виконання однієї ітерації алгоритму "цифра за цифрою" показана на рис. 7.39а.

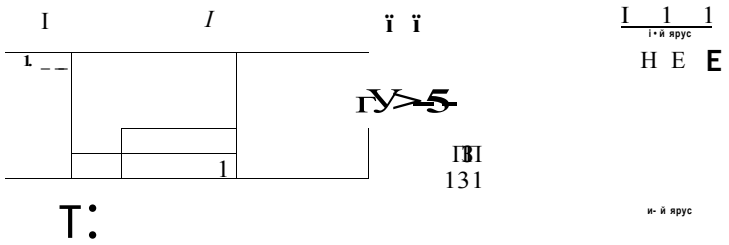


Рис. 7.39. Структура пристрою для виконання однієї ітерації алгоритму "цифра за цифрою"

Параметр р управляє подачею на суматор-віднімач СВЗ констант С. Параметр {. управляє суматорами-віднімачами, поступаючи із знакового розряду У. або *7.* через комутатор КМ 1 залежно від коду операції Ы.

Включивши послідовно п таких пристроїв, де п - кількість ітерацій, одержимо конвеєрний операційний пристрій виконання алгоритму "цифра за цифрою" (рис. 7.39Ь). Якщо з описаного пристрою видалити конвеєрні регістри, отримаємо однотактовий операційний пристрій виконання алгоритму "цифра за цифрою".

7.13.5. Пристрої для виконання арифметичних операцій над числами з рухомою комою

7.13.5.1. Пристрої додавання і віднімання чисел з рухомою комою

Як відомо, формат з рухомою комою передбачає наявність двох частин числа - порядку (Р) та мантиси (М). Числа представляються у вигляді $X = M2^{p_x}$, $У = М_у 2^{р_у}$.

При виконанні додавання та віднімання порядки операндів вирівнюються, а мантиси додаються. Порядки вирівнюються збільшенням порядку меншого операнду до значення порядку більшого операнду. Цей порядок є порядком результату. Щоб при вирівнюванні порядків величина операнда не змінювалася, його мантиса одночасно зменшується. Після виконання вирівнювання порядків виникають додаткові молодші розряди мантиси, які до, або після додавання, відкидаються або заокруглюються.

Блок-схема додавання та віднімання двійкових чисел з рухомою комою наведена на рис. 7.40.

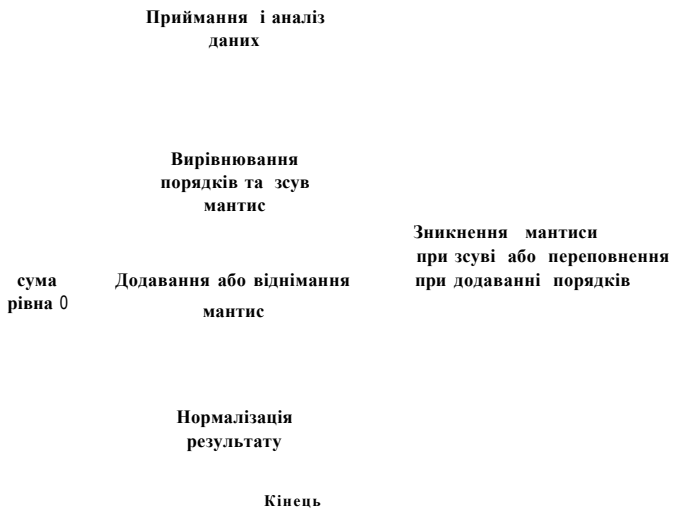


Рис. 7.40. Блок-схема додавання та віднімання двійкових чисел з рухомою комою

Детальніший розпис алгоритму додавання та віднімання двійкових чисел з рухомою комою наведено нижче, а структура однотактового операційного пристрою додавання та віднімання двійкових чисел з рухомою комою наведена на рис. 7.41. Послідовність кроків алгоритму та відповідні їм вузли пристрою є наступними:

• Віднімання порядків чисел з метою знаходження їх різниці та визначення, яке з двох чисел є більшим, а яке меншим, для чого в схемі використано віднімач.

• Зсув праворуч мантиси числа, яке має менший порядок. Для цього знак з віднімана керує мультиплексорами, які пропускають на вхід схеми зсуву відповідну мантису.

• Додавання іншої мантиси до зсунутої, яке виконується на суматорі.

• Визначення кількості нулів в старших розрядах отриманої суми з метою визначення кількості розрядів зсуву при нормалізації, для чого в пристрій введена відповідна схема.

• Зсув на схемі зсуву цієї суми вліво на кількість розрядів, рівну кількості нулів в її старших розрядах.

• Віднімання від більшого порядку числа, рівного кількості нулів в старших розрядах отриманої суми, тобто коригування порядку.

Рис. 7.41. Структура однотактового операційного пристрою додавання та віднімання двійкових чисел з рухомою комою

При необхідності представлений на рис. 7.41 операційний пристрій додавання та віднімання двійкових чисел з рухомою комою може бути конвеєризований шляхом введення конвеєрних регістрів між його ярусами.

7.73.5.2. *Пристрої множення та ділення чисел з рухомою комою*

Пристрої множення та ділення чисел з рухомою комою будуються на основі суматорів, піднімачів, перемножувачів та подільників з фіксованою комою, які вже були розглянуті, а також пристрою нормалізації чисел. Побудова пристроїв множення та ділення чисел з рухомою комою на основі названих пристроїв не є складною, оскільки алгоритми є досить простими. Тому розгУїянемо далі лише питання забезпечення коректного виконання цих операцій.

Операції множення та ділення чисел з рухомою комою описуються відповідно виразами:

$$Ъ = M_x 2^{P_x} \cdot M_y 2^{P_y} = M_x M_y 2^{(P_x + P_y)}$$

$$Ъ = Ш = M_x 2^{P_x} / M_y 2^{P_y} = M_x / M_y 2^{\wedge}$$

Блок-схеми виконання цих виразів представлено відповідно на рис. 7.42 а) та Ь).

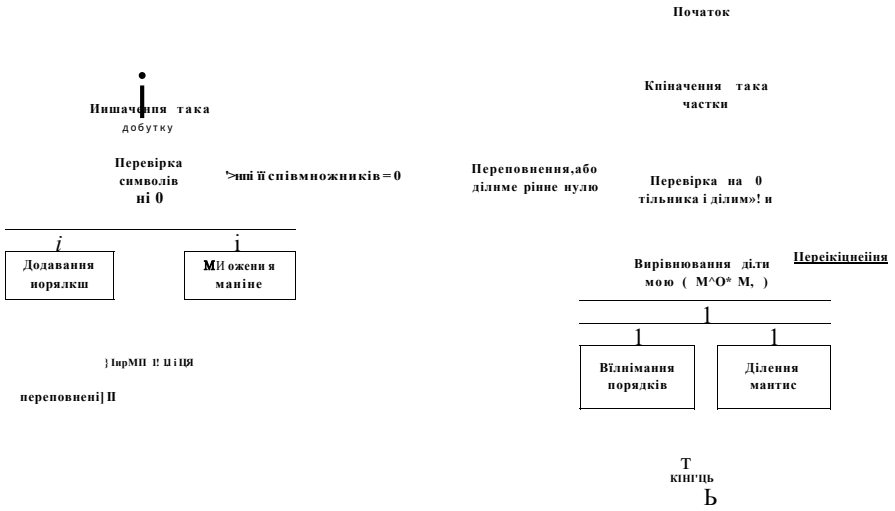


Рис. 7.42. Блок-схеми виконання операцій множення а) та ділення Ь) чисел з рухомою комою

7.14. Таблично-алгоритмічні операційні пристрої

Арифметичні операції в таблично-алгоритмічних операційних пристроях необхідні для обчислення поправки. Ці обчислення можуть базуватися на використанні різних методів, таких як: дробово-раціональні наближення, ітераційні процеси, розкладання в ряди, наближення поліномами, ланцюговими дробами і т. д. При обчисленні таблично-алгоритмічним методом аргумент звичайно розбивається на дві частини. Розбивка аргументу на більшу кількість частин при неконвеєрній реалізації нераціональна через значну кількість додаткових операцій. При конвеєрній же реалізації така розбивка в ряді випадків може виявитися вигідною.

Застосування конвеєрного принципу обробки для виконання арифметичних операцій у таблично-алгоритмічних операційних пристроях дозволяє значно скоротити об'єми пам'яті в порівнянні з прямим табличним методом без зменшення, або і з підвищенням їх продуктивності. Продуктивність операційного пристрою, в якому застосований таблично-алгоритмічний метод, може бути вищою, ніж при використанні табличного методу, тому що швидкодія ПЗП залежить від розрядності операндів п, і, при великих вимогах по точності, може перевершувати затримку в одному ярусі операційного пристрою, яку можна довести до затримки в ПЗП значно меншого об'єму.

Основною проблемою при синтезі таблично-алгоритмічних операційних пристроїв є вибір із всіх існуючих методів обчислення даної функції чи набору функцій найбільш ефективного і визначення оптимальних співвідношень між об'мом ПЗП і витратами обладнання на арифметичну частину при забезпеченні потрібної продуктивності.

В даний час існує багато таблично-алгоритмічних методів обчислення елементарних функцій, частина з яких була розглянута в розділі 4. Тут буде розглянута реалізація декількох таблично-алгоритмічних методів обчислення елементарних функцій, в яких застосований як загальний, так і частковий підходи при обчисленні поправки. Проектування таблично-алгоритмічних операційних пристроїв за іншими, не розглянутими тут, алгоритмами може бути виконане аналогічно приведеному.

Розглянемо принципи побудови таблично-алгоритмічних операційних пристроїв з використанням часткового підходу при обчисленні поправки на прикладі реалізації тригонометричних функцій.

Розбивши аргумент на дві частини X=X1+X2, утворені його старшими і молодшими розрядами, і скориставшись відомими співвідношеннями:

8іп(X1+X2)=5іпX1 СовХг+СовXI 5іпX2;

Со8(X1+X2)=Со5X1 СоБХг-БіпХШпХг,

можна значно скоротити об'єм ПЗП в порівнянні з табличним методом. Структура однотактового операційного пристрою, який реалізує наведений алгоритм, показана на рис. 7.43.

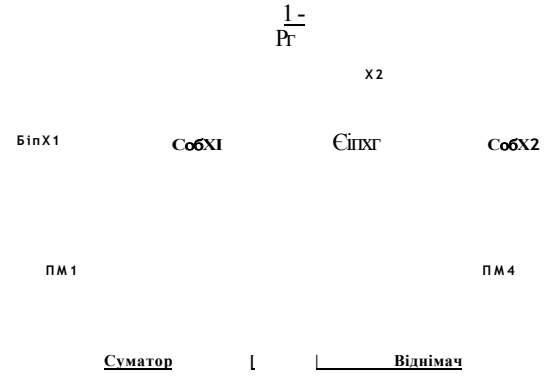


Рис. 7.43. Структура однотактового таблично-алгоритмічного операційного пристрою обчислення тригонометричних функцій

Якщо прийняти, що XI - ціле число градусів з дискретністю 1, а X2 - дробове число градусів, з дискретністю 0,01, то неважко підрахувати, що таблиці, котрі зберігають БіпXI і СовXI, будуть містити по 90 15-розрядних слів, а таблиці, що зберігають 5іпX2 і СовX2 - по 1000 15-розрядних слів. Загальний об'єм ПЗП складе 5700 бітів, що в 24 рази менше, ніж при обчисленні табличним методом. В якості ПЗП з уже прошитими значеннями тригонометричних функцій можна використати мікросхеми ДО505РЕ30068-ДО505РЕ300071 ємністю 4 Кбіти кожна. Перші дві мікросхеми містять 8 старших розрядів 512 значень функції 8іпX у діапазоні 0...П/4, а дві інші мікросхеми містять значення цієї функції в діапазоні П /4... П/2.

Ввівши конвеєрні регістри, включаючи і в перемножувачах ПМ, які також можна побудувати за конвеєрним принципом, можна підвищити продуктивність описаного пристрою. Зрозуміло, що зменшивши «об'єми таблиць, тут збільшені затрати обладнання на операційну частину, тобто введені чотири перемножувачі та суматор і віднімач.

Принципи побудови таблично-алгоритмічних операційних пристроїв з використанням загального підходу при обчисленні поправки розглянемо на прикладі обчислення елементарних функцій з використанням раціонального наближення.

Нехай обробці підлягають нормалізовані числа, представлені у форматі з фіксованою комою. Обчислення елементарних функцій будемо робити на основі методу сегментної апроксимації, відповідно до якого діапазон зміни аргументу [0,5; 1] розбивається на інтервали з наступним наближенням функції на кожнім інтервалі за допомогою виразу $У = Б(Х) = А + Ш (Х + В)^2$. Константи А та В вибираються з умови мінімізації абсолютної похибки, а константа Ш вибирається рівною ступеню числа 2 (основа системи числення), що дозволяє замінити операцію множення операцією зсуву. На різних інтервалах константи мають різні значення. Кількість інтервалів також визначається з умови мінімізації абсолютної похибки, причому границі інтервалів визначаються к старшими двійковими розрядами аргументу, що полегшує здійснення адресації пам'яті, в яку записані константи. Структура однотактового операційного пристрою, який реалізує наведений вираз, показана на рис. 7.44.

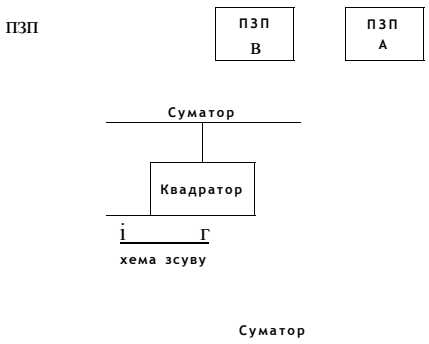


Рис. 7.44. Структура однотактового таблично-алгоритмічного операційного пристрою обчислення елементарних функцій

Для реалізації описаного необхідно три блоки постійної пам'яті для збереження коефіцієнтів, схему зсуву і два суматори. Для 12-ти розрядних операндів загальний об'єм ПЗП складає 8960 бітів. Константи А.В.Ш обчислюють з тим, щоб забезпечити задану точність обчислення. Вихідними даними для реалізації алгоритму розрахунку коефіцієнтів є: реалізована функція, величина інтервалу зміни аргументу, розрядність функції й аргументу. Після обчислення отримують наступні величини: сумарний об'єм ПЗП, число підінтервалів зміни аргументу, число розрядів, необхідних для кодування номера підінтервалу, значення констант А,В,\У У табл. 7.6 як приклад наведені значення констант, границі підінтервалів і абсолютна похибка обчислень т при обчисленні функції I/X. Пристрій, що реалізує формулу $у=А+Ш(Х+В)^2$ є багатофункціональним. Для різних функцій константи мають різні значення. Включення конвеєра команд дозволяє обчислювати на такому пристрої одночасно к функції від к даних.

Подібно до описаного, описані підходи можна застосувати до інших таблично-алгоритмічних методів обчислення елементарних функцій, наприклад, при використанні ланцюгових дробів, раціональних наближень і т. д.

7.75. *Короткий зміст розділу*

Розглянуто принципи побудови арифметико-логічного пристрою сучасних комп'ютерів, який є одним з основних вузлів процесора, призначеним для виконання арифметичних, логічних та інших операцій обробки даних. Розкрито структуру АЛП для виконання елементарних операцій та структуру багатоблокових АЛП, призначених для виконання складних операцій, які ініціюються командами обробки даних з системи команд комп'ютера. Описані багатоблокові АЛП з внутрішньою регістровою пам'яттю на основі табличних, однотактових, багатотактових та конвеєрних операційних пристроїв. Проведено класифікацію АЛП залежно від способу обробки операндів. АЛП діляться на послідовні, послідовно-паралельні та паралельні. В першому випадку обробка операндів в АЛП здійснюється послідовно в часі над кожним розрядом, тоді як в останньому операції здійснюються паралельно в часі над всіма розрядами операндів. За способом представлення чисел розділяють АЛП з фіксованою та з рухомою комою, причому перші можуть бути орієнтовані на обробку цілих або дробових чисел. Залежно від способу виконання операцій АЛП діляться на однотактові, коли задана операція виконується за один такт, та багатотактові, коли для виконання операції потрібно виконати деяку кількість тактів. АЛП можуть бути конвеєрними або скалярними. Показано, що використання конвеєрного принципу обробки даних дозволяє суттєво підвищити продуктивність АЛП та комп'ютера в цілому.

Розглянуто структури та принципи організації обчислень в табличних, алгоритмічних та таблично-алгоритмічних однотактових, багатотактових та конвеєрних операційних пристроях для виконання операцій додавання, віднімання, множення, ділення та обчислення елементарних функцій над двійковими числами в форматах з фіксованою та рухомою комою.

7.76. *Література для подальшого читання*

Питання побудови арифметико-логічного пристрою розглянуті в багатьох підручниках з організації комп'ютерів, зокрема в роботах [1-8]. Формалізації питань побудови багатотактових операційних пристроїв присвячена робота [9], в якій вони названі операційними пристроями з закріпленими мікрооперaціями. Опис стандартного 4-розрядного АЛП приведено в роботі [3]. В роботах [10-12] є опис структури арифметико-логічного пристрою процесорів Nios, UltraSPARC та PA-8000. Принципи побудови табличних та таблично-алгоритмічних операційних пристроїв розглянуті в роботах [13-15]. Принципи побудови однотактових, багатотактових та конвеєрних операційних пристроїв запропоновано в роботах [17-19].

7. 7 7. *Література до розділу 7*

1. Благовещенский Ю. В., Теслер Г. С. Вычисление элементарных функций на ЭВМ. - К., "Техника", 1977. - 208 с.

2. Байков В. Д., Смолов В. Б. Аппаратурная реализация элементарных функций в ЦВМ. - Л. ЛГУ - 96с.

3. Каган Б. М. Электронные вычислительные машины и системы. М.: Энергия, 1979. - 528 с.

4. Каган Б. М., Каневский М. М. Цифровые вычислительные машины и системы. М.: Энергия, 1974. - 680 с.

5. Майоров С. А., Новиков Г. И. Структура электронных вычислительных машин. Л. Машиностроение. 1979. - 384 с.

6. Мельник А.А. Выбор метода вычисления элементарных функций в процессорах обработки сигналов. Тезисы Всесоюзной конференции "Методы и микроэлектронные средства цифрового преобразования и обработки сигналов". - Рига, 1983.

7. Мельник А.А. О вычислении одного класса элементарных функций путем конвейерной реализации метода Волдера. -Автоматика и вычислительная техника, 1983, N 6.

8. Мельник А.А. Использование алгоритма Волдера в высокопроизводительных вычислительных БПФ. - Автометрия, 1984, N 6, с. 85-87.

9. Мельник А.А. Процессоры обработки сигналов. Препринт N 29-89, ИППММ АН УССР, 1989, 63 с.

10. Мельник А.О. Спеціалізовані комп'ютерні системи реального часу. - Львів: Державний університет "Львівська політехніка", 1996. - 54 с.

11. Коркішко Т., Мельник А., Мельник В. Алгоритми та процесори симетричного блокового шифрування. - Львів: БаК, 2003. - 168 с.

12. Оранский А.М. Аппаратные методы в цифровой вычислительной технике. -Минск, Из-во БГУ, 1977. - 208 с.

13. Справочник по цифровой вычислительной технике. Б.Н. Малиновский и др. К. Техніка, 1980. - 320 с.

14. Угрюмов Е.П. Цифровая схемотехника. - СПб.: БХВ - Санкт-Петерберг, 2000. - 528 с.

15. Altera Corporation. Nios programmer's Reference manual. March 2001.

16. Kane, Gerry. PA-RISC 2.0 Architecture, ISBN 0-13-182734-0, Prentice Hall, Englewood Cliffs, NJ, 1996.

17. Melnyk A. Synthesis of the Data Flow Graph Pipeline Operation Devices. IWK-95, Ilmenau, 1995.

18. Melnyk A.O. The Architecture of the Teal-Time Processing System Optimized to the Data Flow Intensity. International Conference, Zakopane, August, 1996.

19. D. Patterson, J. Hennessy. Computer Architecture. A Quantitative Approach. Morgan Kaufmann Publishers, Inc. 1996.

20. Patterson, D. A., 8c Hennessy, J. L. Computer Organization and Design, The Hardware/Software Interface, 2nd ed., San Mateo, CA: Morgan Kaufmann, 1997.

21. Stallings, W. Computer Organization and Architecture, 5th ed,, New York, NY: Macmillan Publishing Company, 2000.

22. Sun Microelectronics. UltraSPARC I&II. Sun Microelectronics. 1997.

23. Tanenbaum, Andrew. Structured Computer Organization, 4th ed., Upper Saddle River, NJ: Prentice Hall, 1999.

24. Voider J.E. The CORDIC trigonometric computing technique. - "IRE Trans.", 1959, 3, pp. 330-334.

7.7 *8. Питання до розділу 7*

1. Назвіть місце АЛП в комп'ютері.
2. Назвіть функції АЛП.
3. Яким чином АЛП взаємодіє з іншими вузлами процесора.
4. Наведіть класифікацію АЛП.

5. Порівняйте послідовний, паралельний та послідовно-паралельний способи обробки інформації в АЛП.

6. Поясніть роботу АЛП для виконання елементарних операцій.

7. Назвіть елементарні операції АЛП. Чому до складу системи команд сучасних комп'ютерів входять команди виконання елементарних операцій?

8. Назвіть складні арифметичні і логічні операції АЛП.

9. Поясніть, що таке граф алгоритму та як його можна використати при виборі структури операційного пристрою.

10. Приведіть класифікацію операційних пристроїв.

11. Як організована робота табличних операційних пристроїв?

12. Поясніть принципи роботи багатотактових операційних пристроїв.

13. Поясніть принципи роботи однотактових операційних пристроїв.

14. Поясніть принципи роботи конвеєрних операційних пристроїв.

15. Наведіть схему та опишіть роботу послідовного АЛП додавання та віднімання двійкових чисел з фіксованою комою.

16. Наведіть схему та опишіть роботу паралельного АЛП додавання та віднімання двійкових чисел з фіксованою комою.

17. Як побудований однотактовий суматор двійкових чисел за методом вибору переносу?

18. Які є методи прискорення роботи паралельного АЛП додавання та віднімання двійкових чисел з фіксованою комою?

19. Назвіть чотири методи та чотири базові структури множення двійкових чисел з фіксованою комою.

20. Поясніть роботу багатотактового пристрою множення двійкових чисел з молодших розрядів множника при нерухомому множеному з зсувом суми часткових добутків.

21. Поясніть роботу багатотактового пристрою множення двійкових чисел з молодших розрядів при нерухомій сумі часткових добутків з зсувом множеного вліво.

22. Поясніть роботу багатотактового пристрою множення двійкових чисел з старших розрядів при нерухомій сумі часткових добутків з зсувом множеного вправо.

23. Поясніть роботу багатотактового пристрою множення двійкових чисел з старших розрядів при нерухомому множеному з зсувом суми часткових добутків вліво.

24. Як будується однотактовий пристрій множення двійкових чисел з фіксованою комою?

25. Наведіть структуру конвеєрного операційного пристрою множення двійкових чисел з фіксованою комою.

26. Наведіть структуру багатотактового АОП ділення двійкових чисел з відновленням залишку.

27. Наведіть структуру багатотактового АОП ділення двійкових чисел без відновлення залишку.

28. Наведіть структуру конвеєрного операційного пристрою ділення двійкових чисел з фіксованою комою за алгоритмом з відновленням залишку.

29. Наведіть структуру конвеєрного операційного пристрою ділення двійкових чисел з фіксованою комою за алгоритмом без відновлення залишку.

30. Поясніть роботу багатотактового пристрою для обчислення елементарних функцій методом "цифра за цифрою".

31. Поясніть роботу конвеєрного пристрою для обчислення елементарних функцій методом "цифра за цифрою".

32. Як будуються пристрої додавання і віднімання чисел з рухомою комою?

33. Як будуються пристрої множення та ділення чисел з рухомою комою?

34. Поясніть роботу операційного пристрою для обчислення елементарних функцій табличноалгоритмічним методом.

Розділ 8

Пристрій/керу&аниш>

8.1. Функції та методи побудови пристрою керування

Пристрій керування виробляє послідовність сигналів, необхідних для виконання команди, та послідовності команд, тобто програми. Команда в комп'ютері виконується за один або за декілька тактів, в кожному із яких виконується одна або декілька мікроопе-рацій. Кожна мікрооперація представляє собою деяку елементарну дію передачі або перетворення інформації, яка ініціюється поступленням керуючого сигналу (мікронаказу) на вхід керування відповідного пристрою. Прикладом може бути керуючий сигнал, який встановлює або очищує прапорець стану, керуючий сигнал запису до регістра, керуючий код на вході мультиплексора і т. д. Для реалізації команди необхідно на відповідні керуючі входи подати розподілену в часі послідовність керуючих сигналів.

Пристрій керування є одним з вузлів процесора. Як приклад на рис. 8.1 показана взаємодія в процесорі між пристроєм керування та арифметико-логічним пристроєм і регістровою пам'яттю.

Рис. 8.1. Взаємодія пристрою керування з іншими вузлами процесора

Процес функціонування процесора в часі складається з послідовності тактових інтервалів, в яких арифметико-логічний пристрій виконує операції над операндами та видає результати обробки. Виконання даних операцій арифметико-логічний пристрій здійснює на основі відповідних сигналів керування (мікронаказів) з пристрою керування. Послідовність елементарних мікронаказів пристрій керування формує на основі коду операції та службових сигналів стану з регістрової пам'яті процесора.

Відомі два основні методи побудови логіки формування керуючих сигналів. Перший з них виражається в тому, що для кожної команди процесора існує набір логічних схем, які в потрібних тактах збуджують відповідні сигнали керування. Такий принцип керування одержав назву "жорсткої" або "запаяної" логіки.

Другий метод, який дістав назву принципу мікропрограмного керування, передбачає формування керуючих сигналів за вмістом регістра мікрокоманд, в який мікрокоманди записуються із пам'яті мікрокоманд. Шляхом послідовного зчитування мікрокоманд із пам'яті в цей регістр організується потрібна послідовність керуючих сигналів.

Крім пристрою керування процесора в комп'ютері можуть використовуватись пристрої керування вузлами комп'ютера, наприклад, пристрої керування операційними пристроями АЛП, пристрій керування процесора введення-виведення і т. д. Принципи побудови вказаних пристроїв є ідентичними.

8.2. Пристрій керування з жорсткою логікою

8.2.1. Структура пристрою керування з жорсткою логікою

Типова структурна схема пристрою керування з жорсткою логікою (в англійській термінології hardwired control) представлена на рис. 8.2.

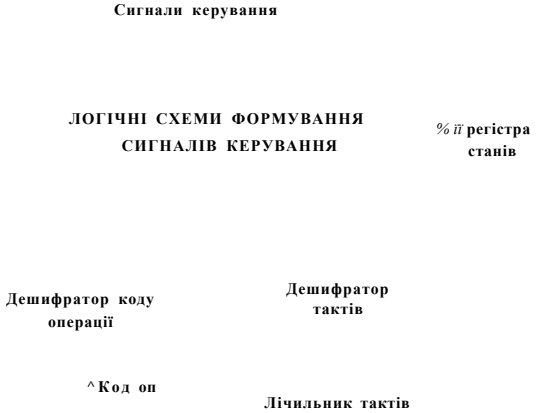


Рис. 8.2. Типова структурна схема пристрою керування з жорсткою логікою

До складу пристрою керування входить блок синхроімпульсів, який генерує тактові імпульси ТІ, потрібні для синхронізації роботи пристрою керування, лічильник тактів, в якому зберігається номер виконуваного в даний час такту, дешифратор коду операції та дешифратор тактів, які перетворюють двійковий код в однорядний, логічні схеми формування сигналів керування. Дешифратор коду операції по коду операції із регістра команд РгК формує сигнал активізації мікрооперації МО на відповідній шині. З кожним тактом до лічильника тактів додається +1 сигналом із блоку синхроімпульсів. Дешифратор тактів формує сигнали, відповідні поточному такту.

Логічні схеми формування сигналів керування відповідно до сигналів із дешифратора коду операції, дешифратора тактів та кодів умов і кодів станів із регістра станів формують сигнали керування для виконання необхідних в даному такті мікрооперацій.

Окрім наведених вище компонентів пристрою керування, до його складу входить контролер послідовності сигналів керування, який отримує тактові імпульси з блоку синхроімпульсів, а також код режиму роботи комп'ютера. Він має два окремих режими роботи: звичайний режим та режим запуску комп'ютера. Контролер послідовності сигналів керування є ядром пристрою керування. Принципи його роботи будуть наведені далі при розгляді пристрою мікропрограмного керування.

8.2.2. Методи проектування пристрою керування з жорсткою логікою

Методи проектування пристрою керування з жорсткою логікою, які застосовуються на практиці, часто є спеціально створеними для побудови конкретного пристрою і евристичними за природою, тому не можуть легко бути формалізованими. Для ілюстрації найбільш широко застосовуваних підходів, розглянемо три методи:

- Перший метод - це стандартний алгоритмічний підхід до проектування послідовніших схем. Його називають методом таблиць станів, оскільки передбачає побудову таблиць станів пристрою керування.
- Другий метод є евристичним і ґрунтується на використанні тактованих елементів часової затримки для побудови часової діаграми керуючих сигналів.
- Спорідненим з другим є третій метод, який передбачає використання лічильників для побудови часової діаграми керуючих сигналів.

Перший метод є найбільш формалізованим і дозволяє застосувати методи мінімізації кількості вентилів та елементів пам'яті. Два інші методи є менш формалізовані і передбачають синтез пристрою керування з часової діаграми сигналів керування.

8.2.3. Пристрій керування на основі таблиць станів

8.2.3.1. Абстрактні автомати

Метод таблиць станів передбачає розгляд пристрою керування як цифрового автомату, тобто логічного пристрою, який забезпечує формування сигналів керування за відповідним алгоритмом з врахуванням своїх внутрішніх станів.

Цифровий автомат можна подати у вигляді його математичної (абстрактної) і структурної моделей, які відповідно називаються абстрактним та структурним автоматами. Абстрактну модель використовують на першому етапі проектування, коли описують функціонування автомату, тобто правила переробки вхідної інформації у вихідну. На цьому етапі автомат подається у вигляді "чорної скриньки". Розгляд абстрактної моделі цифрового автомату дозволяє проводити його попередню оптимізацію ще до етапу структурного синтезу. Структурну модель застосовують для побудови схеми цифрового автомату.

Абстрактним автоматом (математичною моделлю цифрового автомату) називають сукупність з п'яти об'єктів $A = \{X, S, Z, \delta, \lambda\}$. Де:

$X = \{x_i\}$, $i \in 1,m$ - множина вхідних сигналів (вхідний алфавіт);

$S = \{s_y\}$, $y \in 1, n$ - множина станів (внутрішніх) автомату (алфавіт станів автомату);

$Y = \{y_k\}$, $k \in 1,l$ - множина вихідних сигналів (вихідний алфавіт);

8:: ***ХхБ*** —> 5"- функція переходів автомату. Функція переходів показує, що автомат, який перебуває у стані ***Sj*** при поданні вхідного стану х, переходить в деякий стан ***я„*** тобто ***я*** = 8 ***Щ Xi);***

X:: ***ХхБ -» У*** - функція виходів автомату. Функція виходів ***X*** показує, що автомат, який перебуває у стані ***8,,*** при появі вхідного сигналу ***x***$_i$ видає вихідний сигнал

Абстрактний автомат має один вхідний канал і один вихідний канал (рис. 8.3).

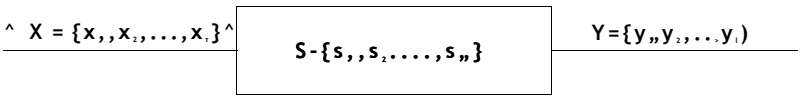


Рис. 8.3. Абстрактний автомат

В подальшому будемо використовувати так званий скінчений абстрактний автомат, в якого множина внутрішніх станів і множина вхідних сигналів (а, отже, й множина вихідних сигналів) є скінченими множинами, повністю визначений (детермінований) абстрактний автомат, в якого функція переходів 8 і функція виходів ***X*** визначені для всіх пар ***{X„Sj***), та ініціальний абстрактний автомат, в якого один із станів ***s***$_0$ ***є S*** виділено як початковий стан, з якого автомат завжди починає роботу.

Отже, на абстрактному рівні функціонування цифровий автомат розглядається як перетворювач вхідних слів у вихідні слова, які складаються з букв вхідного і вихідного алфавіту. Внутрішні стани автомату - це інформація про минуле (передісторію) розви-. тку процесу керування в часі. Вона дозволяє використати час як явну вхідну змінну. Потрібно відзначити, що абстрактний автомат функціонує в дискретному часі, а переходи з одного стану в інший проводяться миттєво.

Залежно від способу генерування значень вихідних сигналів розрізняють три типи автоматів: Мілі, Мура, С-автомат.

Автомат Мілі описується наступною системою рівнянь:

y(t) = X(x(t) s(t))

s(t+l) = b(x(t) s(t)).

Автомат Мілі можна представити у вигляді структурної схеми (рис. 8.4), вузли якої представляють відповідно функцію виходів ***X*** , функцію переходів 5 і пам'ять станів S, та з'єднані між собою відповідними зв'язками. Значення на його виході в момент часу t визначається значенням в даний момент на його вході та його станом, а також функцією виходів. Стан автомату Мілі в момент часу t+I визначається значенням в даний момент на його вході та його станом, а також функцією переходів.

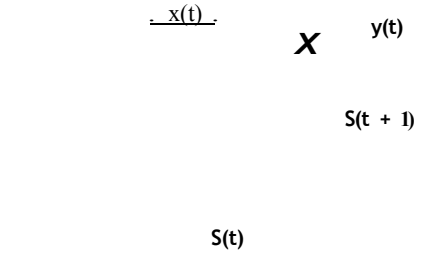


Рис. 8.4. Автомат Мілі

Автомат Мура описується наступною системою рівнянь:

y(t) = X(s(t))

s(/+1)=8(x(0 *s(t)).*

Автомат Мура також можна представити у вигляді структурної схеми (рис. 8.5), вузли якої представляють відповідно функцію виходів ***X*** , функцію переходів 5 і пам'ять станів S, та з'єднані між собою відповідними зв'язками.

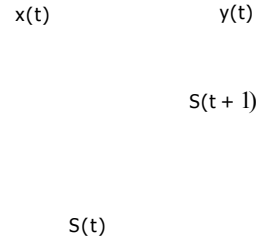


Рис. 8.5. Автомат Мура

Значення на його виході в момент часу t визначається його станом в даний момент, а також функцією виходів. Стан автомату Мура в момент часу t+1 визначається значенням в даний момент на його вході та його станом, а також функцією переходів.

С-автомат описується наступною системою рівнянь:

$_{y_1}$(1) = XXx(0 s(t))

$_{y_1}$(() = xMo)

s(t + l) = b(x(Q s(t))

С-автомат можна представити у вигляді структурної схеми (рис. 8.6), вузли якої представляють відповідно функції виходів А, 1 та А. 2, функцію переходів 5 і пам'ять станів S, та з'єднані між собою відповідними зв'язками. Цей автомат має два виходи. Значення на першому його виході в момент часу t визначається значенням в даний момент на його вході та його станом, а також першою функцією виходів. Значення на його другому виході в момент часу t визначається його станом в даний момент, а також другою функцією виходів. Стан автомату в момент часу t+1 визначається значенням в даний момент на його вході та його станом, а також функцією переходів.

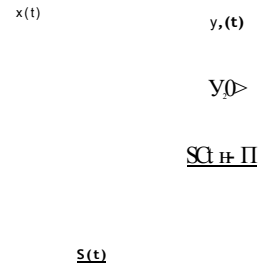


Рис. 8.6. С-автомат

Фактично С-автомат є комбінацією автоматів Мілі та Мура.

8.2.3.2. Мови опису функціонування автоматів

Для того, щоб задати абстрактний автомат, потрібно задати всі п'ять об'єктів {У,5,У,6Д}.

Множини Х, 5, У задаються як звичайні множини в математиці, наприклад, простим перелічуванням всіх її елементів, тому їх задання на практиці не викликає ніяких труднощів.

Найбільш трудомістким є задання функцій 8Д, які власне і визначають алгоритм функціонування автомату. Для опису алгоритму функціонування автомату, тобто для задання §Д, існують різні засоби, які часто називають мовами. Існують стандартні та початкові мови. Стандартні мови задають автомат одним із трьох способів: матрично (таблично), графічно, аналітично. До початкових мов відносять первісні таблиці включень, логічні схеми алгоритмів і граф-схеми алгоритмів. Стандартні мови частіше застосовуються для задання автоматів загального виду, в той же час початкові мови знайшли широке застосування для часткових автоматів.

Мова матриць (таблиць) передбачає наявність двох таблиць: таблиці переходів і таблиці виходів, або однієї таблиці з'єднань.

Таблиця переходів задає відображення $X \times 5 \to 5$, тобто задає функцію переходів 8.

Приклад. Нехай на автомат поступають вхідні сигнали, які мають три букви, та нехай він має чотири стани: Х = *{x,, x„ x^, 8 = {$,, $„ $,}.* В табл. 8.1 описано повністю визначений автомат для даного прикладу.

Таблиця 8.1

\ Х (І)	Х1	Х2	х3
51	Б1		Б1
Э2	БЗ	Б4	Б2
БЗ	ЭЗ	Й1	Б4
Б4	БЗ	Б1	Э2

З першого рядка табл. 8.1 видно, що перебуваючи в стані 81 при поступленні вхідного сигналу ХІ автомат не змінює свого стану, так само як і при поступленні вхідного сигналу ХЗ, а при поступленні вхідного сигналу Х2 він перейде в стан Б2. Подібним чином можна провести аналіз інших рядків таблиці.

Якщо автомат частковий, то для пар *(x„ $^),* для яких стан не визначений, в клітинці таблиці ставиться прочерк. Як видно, вигляд таблиці переходів не залежить від того, який тип автомату використовується: Мілі, Мура чи С-автомат.

Таблиці виходів цих автоматів відрізняються.

У клітинці таблиці виходів автомату Мілі ставиться вихідний сигнал *y ,* який формує автомат Мілі, що знаходиться в стані 5, і на вході якого діє сигнал *x..* Приклад повністю визначеного автомату Мілі з вхідним алфавітом Х = *{x,, x„ x,},* алфавітом станів; $5 = /i_1, 5_2, 5_3, \$_4\}$ та вихідним алфавітом У = *{y,, y„ y^* наведено в табл. 8.2.

Таблиця 8.2

\X(1)	Х1	Х2	хз
Э1	У1	У1	У2
Э2	УЗ	УЗ	УЗ
ЭВ	У2	У1	УЗ
Б4	У2	У1	УЗ

З першого рядка табл. 8.2 видно, що перебуваючи в стані 51 при поступленні вхідного сигналу XI на виході автомату буде сформовано сигнал У1, так само, як і при поступленні вхідного сигналу Х2, а при поступленні вхідного сигналу ХЗ на виході автомату буде сформовано сигнал У2. Подібним чином можна провести аналіз інших рядків таблиці.

В таблиці виходів повністю визначеного автомату Мура кожному стану автомату призначається відповідний вихідний сигнал *у* . Приклад повністю визначеного автомату Мура з алфавітом станів 5 = **/5,, 5,, 5**3,*в*,} та вихідним алфавітом У = *{у,, у?* ,,] наведено в табл. 8.3.

Таблиця 8.3

щ	У(г)
Э1	У1
Э2	У1
БЗ	У2
Б4	УЗ

З табл. 8.3 видно, що стану Б1 та 52 автомату відповідає вихідний сигнал У1, стану БЗ відповідає вихідний сигнал У2, а стану відцовідає вихідний сигнал УЗ.

С-автомат буде задаватися двома таблицями виходів, перша з яких відповідає таблиці виходів автомату Мілі, а друга - таблиці автомату Мура.

На практиці таблиці переходів і таблиці виходів часто суміщаються в одну суміщену таблицю. Табл. 8.4 є суміщеною таблицею автомату Мілі для вищенаведеного прикладу.

Таблиця 8.4

\ Х(г) 8 (1 Г \	Х1	Х2	ХЗ
	Э1/У1	82/У1	31/У2
Э2	БЗ/УЗ	54/УЗ	эг/уз
БЗ	БЗ/У2	БІ/УІ	34/УЗ
Э4	ЭВ/У2	Э1/У1	Э2/УЗ

З першого рядка табл. 8.4 видно, що перебуваючи в стані Б1 при поступленні вхідного сигналу XI автомат залишиться в тому ж стані, а на виході автомату буде сформовано сигнал У1, при поступленні вхідного сигналу Х2 автомат перейде в стан 82, а на виході автомату буде сформовано сигнал У1, при поступленні вхідного сигналу ХЗ автомат залишиться в тому ж стані, а на виході автомату буде сформовано сигнал У2. Подібним чином можна провести аналіз інших рядків таблиці.

Як вже зазначилось вище, можна задати керуючий автомат за допомогою єдиної таблиці з'єднань. Таблиця з'єднань абстрактного автомату є квадратною і містить стільки стовпців та рядків, скільки різних станів має даний автомат. В клітинці ставиться вхідний сигнал, під дією якого відбувається перехід автомату зі стану в стан. Якщо матрицею з'єднань задається автомат Мілі, то разом з вхідним сигналом вказується вихідний сигнал, який автомат Мілі видає, виконуючи перехід (табл. 8.5).

Таблиця 8.5

" \ S(t) S (t) \ ^	S1	S2	S3	S4
S1	X1/Y1 X3/Y2	X2/Y1		
S2		X3/Y3	X1/Y3	X2/Y3
S3	X2/Y1		X1/Y2	X3/Y3
S4	X2/Y1	X3/Y2	X1/Y2	

З першого рядка табл. 8.5 видно, що автомат залишається в тому ж стані 81 при поступленні вхідних сигналів XI та ХЗ, і при цьому на його виході будуть відповідно сигнали У1, та У2, та переходить в стан 82 при поступленні вхідного сигналу Х2, і при цьому на його виході буде сигнал У1. Подібним чином можна провести аналіз інших рядків таблиці.

Для автомату Мура в матриці з'єднань вихідні сигнали ставляться біля станів автомату, які ідентифікують рядки матриці.

Мова графіки передбачає застосування для задания абстрактного автомату орієнтованого графа. Стан автомату зображається вершинами графа, а переходи між станами - дугами між відповідними вершинами. При цьому конкретній дузі графа приписується буква x, вхідного алфавіту автомату, яка вказує на перехід при поступленні цього сигналу.

Якщо граф зображає автомат Мілі (рис. 8.7), то вихідні сигнали автомату ставляться на дугах графа (згідно з таблицею виходів) разом з буквою вхідного сигналу. Тут в якості прикладу взято автомат Мілі, описаний в табл. 8.4.

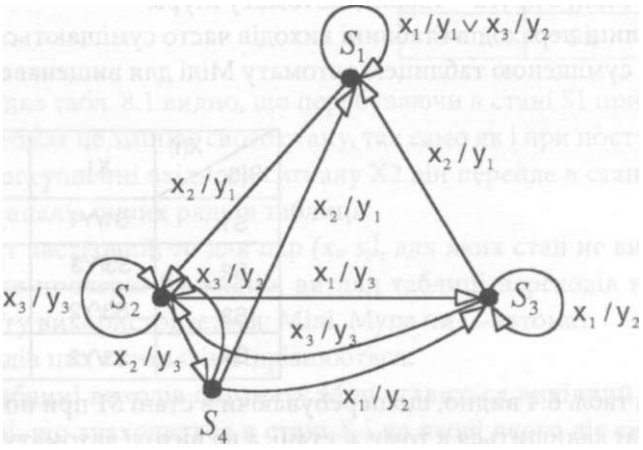


Рис .8.7. Граф автомату Мілі

Якщо графом зображається автомат Мура (рис. 8.8), то вихідні сигнали автомату ставляться біля вершини графа відповідно до таблиці виходів автомату (табл. 8.3).

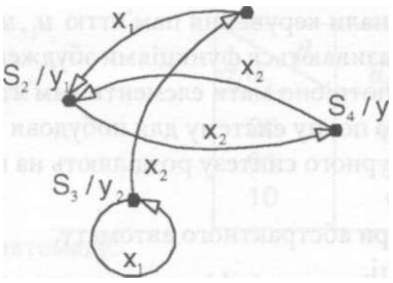


Рис. 8.8. Граф автомату Мура

Мова аналітичних виразів передбачає задання автомату шляхом запису для кожного стану автомату відображення F_{y_n}, яке містить набори з трьох об'єктів s_m, x, y_n, причому тільки таких, які вказують на наявність переходу автомату зі стану s • в стан s_m при дії вхідного сигналу x_l і видачі при цьому вихідного сигналу y_k.

Приклад:

Р,2 = ' Л) Л (* 2 ЇУ3\?2<Х31 Уз)}.

8.2.3.3. Структурний синтез цифрових автоматів

Процес одержання структурної схеми, яка відображає склад логічних елементів та їхні зв'язки, називають структурним синтезом. В загальному випадку задача структурного синтезу зводиться до композиції деяких простих автоматів, тобто до пошуку способу з'єднань цих автоматів між собою. Як правило, ефективно розв'язується задача структурного синтезу тільки для певного набору простих автоматів певного виду - елементарних автоматів, які складаються з елементів пам'яті, що мають більше одного стійкого стану (елементарних автоматів з пам'яттю) та комбінаційних схем (елементарних автоматів без пам'яті).

Метод синтезу, в основу якого покладені елементарні автомати, отримав назву канонічного методу структурного синтезу автоматів. Загальна структура елементарного автомату, що складається з пам'яті та комбінаційної схеми, представлена на рис. 8.9.

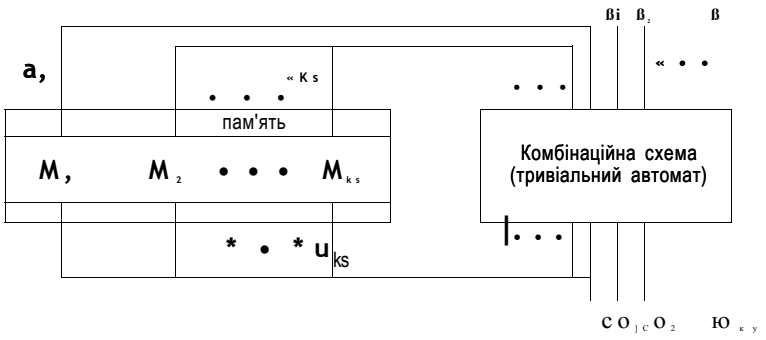


Рис. 8.9. Загальна структура елементарного автомату

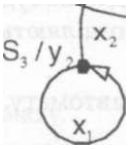


Рис. 8.8. Граф автомату Мура

Мова аналітичних виразів передбачає задання автомату шляхом запису для кожного стану автомату відображення F_y, яке містить набори з трьох об'єктів $s_n x_l y_n$, причому тільки таких, які вказують на наявність переходу автомату зі стану в стан s_n при дії вхідного сигналу x_l і видачі при цьому вихідного сигналу y_k.

Приклад:

'*& = ' у M (X , / у ,) Д (д з / д) }

^2 = & < Л fy,XS,(x, fy,\S,(x, /,,)\

8.2.3.3. Структурний синтез цифрових автоматів

Процес одержання структурної схеми, яка відображає склад логічних елементів та їхні зв'язки, називають структурним синтезом. В загальному випадку задача структурного синтезу зводиться до композиції деяких простих автоматів, тобто до пошуку способу з'єднань цих автоматів між собою. Як правило, ефективно розв'язується задача структурного синтезу тільки для певного набору простих автоматів певного виду - елементарних автоматів, які складаються з елементів пам'яті, що мають більше одного стійкого стану (елементарних автоматів з пам'яттю) та комбінаційних схем (елементарних автоматів без пам'яті).

Метод синтезу, в основу якого покладені елементарні автомати, отримав назву канонічного методу структурного синтезу автоматів. Загальна структура елементарного автомату, що складається з пам'яті та комбінаційної схеми, представлена на рис. 8.9.

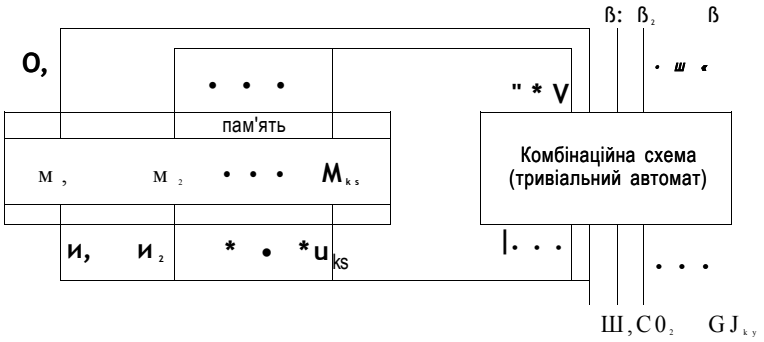


Рис. 8.9. Загальна структура елементарного автомату

Елементарний автомат має кх входів р 1 р 2... р кх, ку виходів со,со$_2$ ... та кв виходів пам'яті станів а$_1$ а$_2$. Сигнали керування пам'яттю $u_1u_2...u_k$ описуються за допомогою булевих функцій, які називаються функціями збудження. Таким чином, для побудови структурного автомату потрібно мати елементи пам'яті і набір логічних елементів, які утворюють функціонально повну систему для побудови комбінаційної схеми.

Канонічний метод структурного синтезу розділяють на наступні етапи:

- кодування,
- вибір типу та структури абстрактного автомату,
- вибір елементів пам'яті,
- побудова рівнянь булевих функцій збудження і виходів автомату,
- побудова структурної схеми автомату.

Розглянемо кожен з етапів.

Кодування. Нагадаємо, що абстрактний автомат задається в вигляді $A = \{.уЯ^бД\}$. При переході на структурний рівень множини сигналів Х та У, а також сигнали Б потрібно зобразити у вигляді двійкового вектору.

Нехай 5 = /5$_1$, 5$_2$, $5_n5^5_n5^$, тоді $к_з = \} ^2M; [=]$ $[= 3$. Тобто для нумерації кожного стану потрібно 3 розряди, тоді 5 = {000,001,010,011,100,101}. Фізично в структурному автоматі буде три стани, кожен з яких може прийняти тільки два значення 0 або 1. Сукупність значень цих трьох станів буде відповідати одному із станів абстрактного автомату.

Приклад: автомат описується суміщеною таблицею переходів та виходів (табл. 8.6).

Таблиця 8.6

	Х1	Х2
		ЄЇ/УЗ
Э2	33/У2	вілч
		Б2Г/2

Тобто він має три стани M_s = п = 3, два вхідних сигнали M_x = т = 2 та чотири вихідних сигнали M_y = 1 = 4. Вони відповідно можуть бути закодовані наступною кількістю розрядів: K_s = 2, K_x = 1 та K_y = 2.

Результати кодування вхідних сигналів наведено в табл. 8.7, станів в табл. 8.8, та вихідних сигналів - в табл. 8.9.

Таблиця 8.7

Щ	*Р*
Х1	0
Х2	1

Таблиця 8.8

	а	*а2*
Э1	0	0
Б1	0	1
Э1	1	0

Таблиця 8.9

У(і)	*со*	*и)2*
У1	0	0
У2	0	1
УЗ	1	0
У4	1	1

Тоді суміщена таблиця переходів та виходів (табл. 8.6) з закодованими входами, станами та виходами, буде мати вигляд табл. 8.10.

Таблиця 8.10

\ *в*	0 м,и$_2$ / ***щсо***$_2$	1 И|И$_2$ / ***щоз***$_2$
00	01/00	00/10
01	10/01	00/11
10	10/00	01/01

Побудова абстрактного автомату.

Структурна схема цифрового автомату Мілі для розглядуваного прикладу має вигляд, показаний на рис. 8.10.

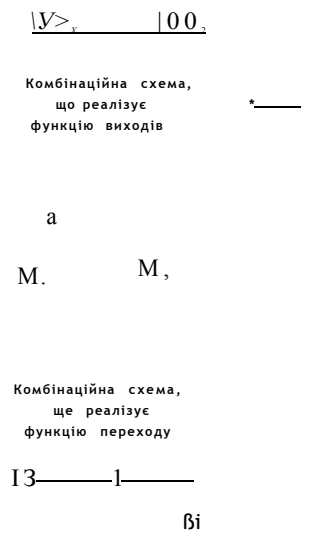


Рис. 8.10. Структурна схема цифрового автомату для розглядуваного прикладу

Вибір елементів пам'яті.

В якості елементів пам'яті структурного автомату можуть бути використані всі відомі типи тригерів, зокрема D-тригери, RS-тригери, Т-тригери, JK-тригери.

Якщо в якості елементів пам'яті вибираються тригери, які мають вхід синхронізації, то структурний автомат буде синхронним, а якщо вибираються асинхронні тригери, то автомат буде асинхронним.

Побудова рівнянь булевих функцій збудження і виходів автомату.

Провівши кодування та вибравши систему логічних елементів, можна однозначно визначити структуру комбінаційних схем автомату. Рівняння булевих функцій будуються на основі таблиці істинності функції збудження, яка в свою чергу будується на основі структурної таблиці переходів і таблиці переходів елемента пам'яті. Для наведеної вище таблиці маємо:

$u_x = \mathbf{a_1 a_2 ß}$

$\mathbf{н_2 = a_1 oc 2ß v a_1 a 2ß}$

$w_1 = ßa_1$

$w_2 = \mathbf{a_1 a_2 v a_1 a 2ß}$.

Таблиця 8.10

	0	1 и,и$_2$ / ***щт***$_2$
00 01 10	01/00 10/01 10/00	00/10 00/11 01/01

Побудова абстрактного автомату.

Структурна схема цифрового автомату Мілі для розглядуваного прикладу має вигляд, показаний на рис. 8.10.

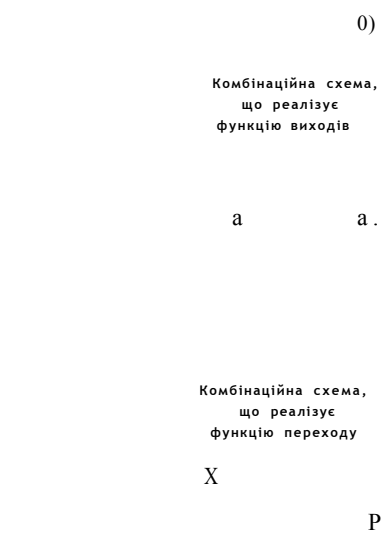


Рис. 8.10. Структурна схема цифрового автомату для розглядуваного прикладу

Вибір елементів пам'яті.

В якості елементів пам'яті структурного автомату можуть бути використані всі відомі типи тригерів, зокрема Б-тригери, КБ-тригери, Т-тригери, ІК-тригери.

Якщо в якості елементів пам'яті вибираються тригери, які мають вхід синхронізації, то структурний автомат буде синхронним, а якщо вибираються асинхронні тригери, то автомат буде асинхронним.

Побудова рівнянь булевих функцій збудження і виходів автомату.

Провівши кодування та вибравши систему логічних елементів, можна однозначно визначити структуру комбінаційних схем автомату. Рівняння булевих функцій будуються на основі таблиці істинності функції збудження, яка в свою чергу будується на основі структурної таблиці переходів і таблиці переходів елемента пам'яті. Для наведеної вище таблиці маємо:

«, =a,oc$_2$P

и$_2$ **=a,a2Pya,a2P**

>?, =pa,

н>$_2$ **=a,a$_2$ya,a2P .**

Побудова структурної схеми автомату.

Маючи наведені логічні рівняння, синтезується схема автомату, наведена на рис. 8.11.

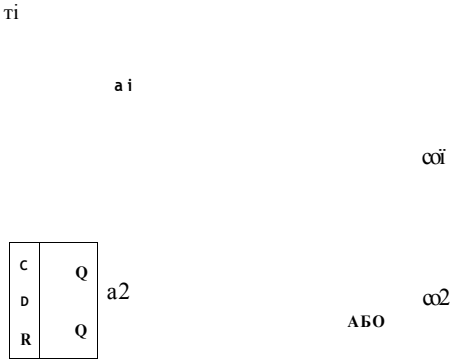


Рис. 8.11. Схема цифрового автомату для розглянутого прикладу

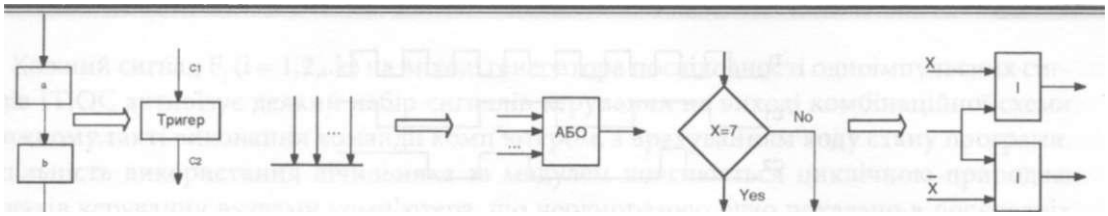
8.2.4. Пристрій керування на основі синхронних елементів часової затримки

Пристрій керування на основі синхронних елементів часової затримки будується з блок-схеми, яка вказує порядок формування послідовності сигналів керування. Структура таким чином побудованого пристрою керування по суті повторює структуру блок-схеми, тобто схема пристрою керування відображає послідовність формування сигналів керування. В основу методу побудови пристрою керування вказаного типу покладено те, що формування набору керуючих сигналів в послідовні моменти часу можна здійснити шляхом здійснення їх часової затримки. Тобто, маючи сигнал СІ в момент часу її, шляхом його затримки на один такт, можна сформувати сигнал С2 в момент часу *12* і т. д. В доволі спрощеному вигляді правила побудови пристрою керування на основі блок-схеми є наступними:

1. Кожні дві послідовно з'єднані мікрооперації замінюються одним елементом затримки, як це показано на рис. 8.12а. В якості елемента затримки може бути використаний тригер типу *V,* який керується спільним тактовим сигналом. Мікронакази подаються на вхід керування відповідного вузла комп'ютера прямо з входу та виходу елемента затримки.

2. Лінії зв'язків блок-схеми, які об'єднуються в одну лінію, перетворюються в к-входову логічну схему АБО, де к - **кількість ліній, як** це показано на рис. 8.12Ь.

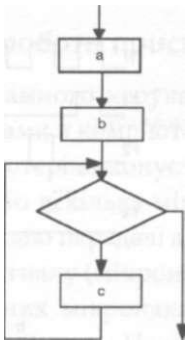
3. Умовна вершина, яка відображає умовний перехід в потоці керуючих сигналів, реалізується двома логічними елементами І, як це показано на рис. 8.12с. Фактично ця вершина є однорозрядним демультиплексором, який керується сигналом х.



а Ь с

Рис. 8.12. Заміна двох послідовно з'єднаних мікрооперацій на один елемент затримки а), к ліній на к-входовий елемент АБО Ь), та умовну вершину на два елементи 1с)

На рис. 8.13а показано фрагмент типової блок-схеми, що задає сигнали керування, які потрібно сформувати в послідовних тактах, а на рис .8.13Ь показано відповідний їй фрагмент схеми пристрою керування, отриманий за вище описаними правилами.



Тригер

Тригер

1 _ £ Г

Рис. 8.13. Фрагмент блок-схеми, яка задає сигнали керування а), та відповідний їй фрагмент схеми пристрою керування Ь)

Не дивлячись на простоту описаного методу проектування пристрою керування на основі синхронних елементів затримки, цей метод має той недолік, що число потрібних схем затримки приблизно рівне числу станів п, тоді як в раніше розглянутому методі таблиць станів кількість елементів пам'яті, які виступають в даному випадку в ролі елементів затримки, рівна $\log_2 n$. Крім того, тут існує проблема синхронізації багатьох розподілених елементів затримки.

8.2.5. Пристрій керування на основі лічильників

В основу методу побудови пристрою керування на основі лічильників покладено часову діаграму роботи комп'ютера, яка відображає зміну в часі кожного сигналу керування. В якості прикладу на рис. 8.14 наведено фрагмент часової діаграми роботи комп'ютера, де ТІ - тактові імпульси, які поступають з блоку синхроімпульсів (рис. 8.2), С1-С5 - частина сигналів керування, які мають бути вироблені пристроєм керування.

Рис. 8.14. Фрагмент часової діаграми роботи комп'ютера

Основним елементом пристрою керування на основі лічильників є лічильник за модулем к, виходи якого з'єднані з дешифратором (рис. 8.15а).

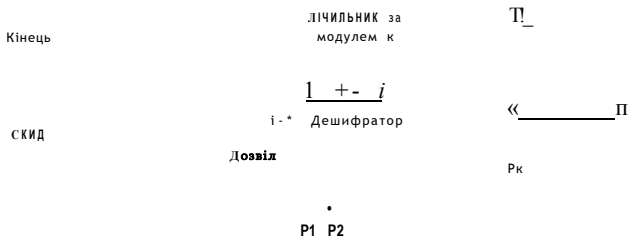


Рис. 8.15. Генератор послідовності одноімпульсних сигналів
а) та часова діаграма сигналів на його виході Ь)

Коли на вході лічильника за модулем к є тактові імпульси, він проводить їх підрахунок від нульового до к-го імпульсу, після чого цикл повторюється. В результаті на виході дешифратора буде формуватися послідовність одноімпульсних сигналів Б1, *¥2,..., ¥к,* часова діаграма яких наведена на рис. 8.15Ь. Кожний з цих сигналів має одиничне значення лише протягом одного тактового періоду. Тим самим, час одного циклу роботи лічильника поділено на к рівних частин. Два додаткових вхідних сигнали початку та кінця роботи та тригер типу забезпечують формування сигналів дозволу роботи лічильника та його скиду. Назвемо схему, представлену на рис. 8.15а, генератором послідовності одноімпульсних сигналів (ГПОС). Тоді базова частина схеми пристрою керування на основі лічильника буде мати вигляд, показаний на рис. 8.16.

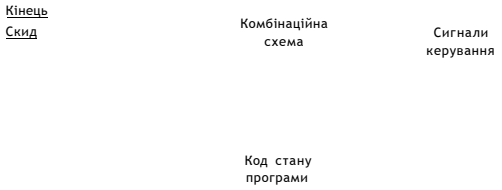


Рис. 8.16. Схема пристрою керування на основі лічильника

Кожний сигнал Б. (і = 1,2,..к) на виході генератора послідовності одноімпульсних сигналів ГПОС активізує деякий набір сигналів керування на виході комбінаційної схеми в кожному такті виконання команди комп'ютером, з врахуванням коду стану програми. Доцільність використання лічильника за модулем пояснюється циклічною природою сигналів керування вузлами комп'ютера, що неодноразово було показано в попередніх розділах.

Потрібно відзначити, що лічильник за модулем к може бути використаний і в схемі пристрою керування на основі синхронних елементів часової затримки взамін к послідовно з'єднаних тригерів, так само як к послідовно з'єднаних тригерів можуть замінити лічильник за модулем к та дешифратор у вище наведеній схемі (рис. 8.15).

8.3. Пристрій мікропрограмного керування

8.3.1. Організація роботи пристрою мікропрограмного керування

Пристрій мікропрограмного керування виробляє послідовність сигналів, необхідних для виконання програми в комп'ютері. Програма складається з деякої послідовності команд. Команда в комп'ютері виконується за один або за декілька тактів, в кожному із яких виконується одна або декілька мікроопераций. Кожна мікрооперація представляє собою деяку елементарну дію передачі або перетворення інформації, яка ініціюється поступленням керуючого сигналу (мікронаказу) на вхід керування відповідного пристрою. Послідовність елементарних мікронаказів, які пристрій керування формує в одному такті, називають мікрокомандою. Послідовність мікрокоманд, що необхідно виконати для виконання однієї команди, називаються мікропрограмою. Звичайно, мікропрограма може складатися і лише з однієї мікрокоманди.

Основними принципами, які покладені в основу побудови пристрою мікропрограмного керування, є наступні:

1. Всі мікронакази, які повинні бути виконані в одному такті роботи комп'ютера, збираються в одне керуюче слово, яке називають мікрокомандою.

2. Кожній команді з системи команд комп'ютера ставиться у відповідність послідовність мікрокоманд, необхідних для її виконання, тобто мікропрограма виконання команди в комп'ютері.

3. Всі мікрокоманди зберігаються в пам'яті. Це може бути основна пам'ять комп'ютера, але в більшості комп'ютерів для зберігання мікрокоманд використовується окрема пам'ять, яку називають пам'яттю мікрокоманд.

4. Для реалізації деякої команди необхідно зчитати з пам'яті мікрокоманд відповідну послідовність мікрокоманд (мікропрограму) та подати розподілену в часі послідовність керуючих сигналів на відповідні керуючі входи вузлів комп'ютера.

На рис. 8.17 показано основний елемент пристрою мікропрограмного керування - пам'ять мікрокоманд, та вузли на її входах і виходах, а саме мікропрограмний лічильник (МКПЛ) для зберігання адреси мікрокоманди та регістр мікрокоманди (РгМК).

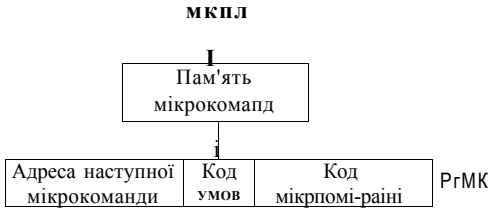


Рис. 8.17. Пам'ять макрокоманд з регістром адреси мікрокоманди на вході та регістром мікрокоманди на виході

На рис. 8.17 показано і формат самої мікрокоманди,.до складу якої входять наступні поля: код мікрооперації, за яким формуються мікронакази, що виконуються в одному такті роботи комп'ютера, код умов, котрий вказує, при яких умовах може бути змінено послідовність читання мікрокоманд з пам'яті, а також адреса наступної мікрокоманди.

Структура пристрою мікропрограмного керування представлена на рис. 8.18.

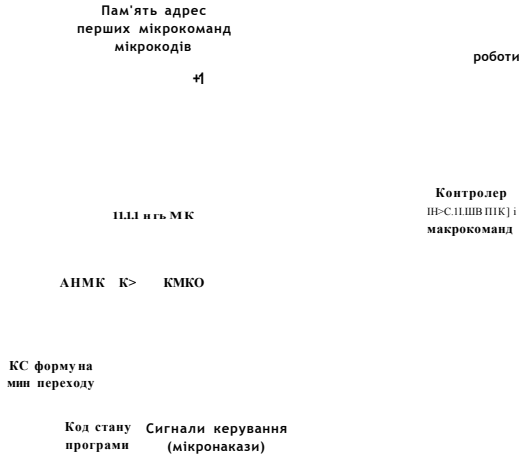


Рис. 8.18. Структура пристрою мікропрограмного керування

В регістрі команди РгК зберігається команда, яка підлягає виконанню в комп'ютері. За її кодом операції КОП з пам'яті адрес перших мікрокоманд мікрокодів зчитується адреса пам'яті мікрокоманд, в якій знаходиться перша мікрокоманда із послідовності мікрокоманд (мікропрограми, або як її іще називають, мікрокоду) її виконання. Пам'ять мікрокоманд в більшості випадків реалізується на основі постійного запам'ятовуючого пристрою ПЗП, хоча може бути реалізована і на основі оперативного запам'ятовуючого пристрою ОЗП, особливо на етапах відлагодження комп'ютера. В цій пам'яті зберігаються мікрокоманди для всіх команд комп'ютера, а також для початку роботи комп'ютера

та для обробки переривань. В схемі формування адреси СФА, до складу якої входять пам'ять адрес перших мікрокоманд мікрокодів, суматор-мультиплексор СМ-МП, мікропрограмний лічильник МКПЛ та комбінаційна схема КС формування переходу, визначається адреса комірки пам'яті мікрокоманд, в якій знаходиться наступна мікрокоманда. Ця адреса формується з врахуванням полів адреси наступної мікрокоманди АНМК та коду умови з регістра мікрокоманди, а також сигналів стану, які поступають з регістра слова стану програми. Мікрокоманда зчитується із пам'яті в регістр мікрокоманди РгМК, в якому вона зберігається протягом одного такту роботи комп'ютера. На основі коду мікрооперації КМКО з регістра мікрооперації на виході дешифратора ДШМКО формуються сигнали керування.

Робота компонента пристрою керування синхронізується з контролера послідовності мікрокоманд, який отримує ззовні тактові імпульси з генератора тактових імпульсів, а також код режиму роботи комп'ютера. Контролер послідовності мікрокоманд є ядром пристрою керування. Він має два окремих режими роботи: звичайний режим та режим запуску комп'ютера.

В звичайному режимі контролер послідовності мікрокоманд генерує сигнали керування роботою пристрою керування. Тактові імпульси на його вході забезпечують його часову синхронізацію, що дозволяє генерувати наступні послідовності сигналів:

1. Керування процесом формування адреси мікрокоманди. Ця адреса одержується шляхом:

- запису в мікропрограмний лічильник МКПЛ адреси першої мікрокоманди мікропрограми виконання відповідної команди з пам'яті адрес перших мікрокоманд мікрокодів;
- запису в мікропрограмний лічильник МКПЛ адреси наступної мікрокоманди мікропрограми виконання відповідної команди після виконання в суматорі-мультиплексорі СМ-МП операції приросту вмісту МКПЛ на 1;
- запису в мікропрограмний лічильник МКПЛ адреси наступної мікрокоманди АНМК з адресного поля РгМК, з врахуванням коду умови КУ з поля умовного переходу РгМК, та коду стану програми з регістра слова стану програми, що здійснюється в комбінаційній схемі КС формування переходу.

Керування зчитуванням мікрокоманд з пам'яті мікрокоманд за адресами з мікропрограмного лічильника МКПЛ та їх записом до регістра мікрокоманд РгК.

Стимулювання дешифратора мікрооперацій ДШМКО, який здійснює дешифрування коду мікрооперації КМКО з відповідного поля РгМК, та видає сигнали керування (мікронакази).

Кожна команда процесора ініціює виконання відповідної мікропрограми. Коли пристрій керування закінчує виконання мікропрограми однієї команди комп'ютера, про що його контролер послідовності мікрокоманд інформується сигналом з виходу дешифратора мікрооперацій ДШМКО, він вибирає з основної пам'яті наступну команду, та записує її в регістр команди РгК сигналом з виходу дешифратора мікрооперацій ДШМКО, після чого приступає до її виконання шляхом зчитування з пам'яті мікрокоманд наступної мікропрограми.

В ході виконання різних команд можуть використовуватися загальні ділянки мікропрограм, які називаються мікропідпрограмами.

В режимі запуску комп'ютера пристрій керування встановлює вміст різних регістрів комп'ютера в початковий стан шляхом їх скиду, або запису до них деяких конкретних значень. Після цього він записує апаратно генеровану адресу до програмного лічильника ПЛ (не мікропрограмного лічильника МКПЛ), та починає виконання програми. Для деяких комп'ютерів апаратно генерована адреса - це вектор скиду, який є адресою першої команди комп'ютера, яка виконується після старту. Для інших комп'ютерів апаратно генерована адреса - це адреса вектора скиду, який є адресою першої команди комп'ютера, яка виконується після старту. В цьому випадку пристрій керування спочатку повинен вибрати з основної пам'яті вектор скиду, та записати його до програмного лічильника ПЛ.

8.3.2. Організація мікропрограм в пам'яті мікрокоманд

Є багато шляхів можливої організації мікропрограм в пам'яті мікрокоманд. Один з них, ілюстрований на рис. 8.23, передбачає розміщення мікрокоманд в пам'яті послідовно. Кожна машинна команда має свою власну послідовність мікрокоманд в пам'яті мікрокоманд. Додатково пам'ять мікрокоманд містить мікрокоманди для проведення вибірки команд з основної пам'яті, запуску переривання, та деяких інших дій керування.

Розглянемо цей тип організації мікропрограм в пам'яті мікрокоманд. Після того, як пристрій керування вибирає машинну команду з основної пам'яті і розміщує її в регістрі команди РгК, він повинен генерувати так звану адресу точки входу для коду операції команди КОГІ, яка є адресою першої мікрокоманди мікропрограми. Наприклад, якщо пристрій керування вибирає мікрокоманду для коду операції КОПІ, він повинен генерувати адресу АІ, як показано на рис. 8.19. Формування адреси точки входу є задачею пам'яті адрес перших мікрокоманд мікрокодів блоку обчислення адреси (рис. 8.18). Після того, як пристрій керування формує адресу точки входу, контролер послідовності мікрокоманд збільшує вміст мікропрограмного лічильника МКПЛ, щоб одержати адресу кожної наступної мікрокоманди.

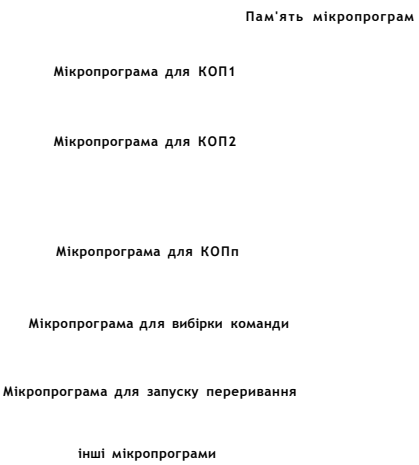


Рис. 8.19. Розміщення мікрокоманд послідовно в пам'яті

Після виконання останньої мікрокоманди в мікропрограмі (тобто, після завершення виконання однієї машинної команди), контролер послідовності мікрокоманд повинен ще раз виконувати мікропрограму вибірки наступної команди з основної пам'яті. На рис. 8.19 це є перехід до комірки $А_{вх}$ пам'яті мікрокоманд. Постає питання - як може контролер послідовності мікрокоманд керувати розгалуженнями в межах мікропрограм пам'яті мікрокоманд? На рис. 8.18 показано один з варіантів вирішення цього завдання. До цих пір ми розглядали мікропрограми без розгалужень. Однак існують мікрокоманди переходу, які містять адресу переходу та мікронакази для схеми формування адреси СФА додатково до мікронаказів для інших вузлів комп'ютера. Контролер послідовності мікрокоманд разом з схемою формування адреси СФА, використовує адресу наступної мікрокоманди (АНМК) та код умови переходу (КУ) для визначення адреси наступної мікрокоманди.

Для керування переходом в коді мікрооперації КМКО кожної мікрокоманди наявна інформація про належність цієї мікрокоманди до мікрокоманд переходу. Тому в кожному такті на виході дешифратора мікрокоманд ДШМКО генерується мікронаказ, який інформує контролер послідовності мікрокоманд про належність, або неналежність, даної мікрокоманди до мікрокоманд переходу. Якщо це мікрокоманда переходу, то контролер послідовності мікрокоманд повідомляє схему формування адрес про необхідність формування адреси переходу, а не приріст на одиницю вмісту мікропрограмного лічильника МКПЛ. При цьому, якщо це безумовний перехід, то перехід відбувається за адресою з адресного поля мікрокоманди. Якщо ж це умовний перехід, то адреса наступної мікрокоманди формується з врахуванням коду умов переходу та коду стану програми, який поступає з регістра стану програми регістрової пам'яті процесора.

8.3.3. Горизонтальне та вертикальне мікропрограмування

Завдання вибору формату мікрокоманди є досить складним. З одного боку, мікрокоманда повинна вміщувати коди керування вузлами комп'ютера, забезпечувати організацію послідовності мікрокоманд в мікропрограму та можливість зміни порядку мікрокоманд в мікропрограмі. З іншого боку, ці функції повиьні виконуватись з мінімальною кількістю бітів в мікрокоманді, з мінімальною кількістю слів в пам'яті мікрокоманд, та з мінімальним часом на виконання мікропрограми. Розглянемо деякі питання вибору ефективного формату мікрокоманди.

За способом формування керуючих сигналів розрізняють горизонтальне і вертикальне мікропрограмування.

При використанні горизонтального мікропрограмування кожний розряд поля коду мікрооперації мікрокоманди формує один керуючий сигнал (мікронаказ) для відповідного входу керування функціонального вузла комп'ютера (рис. 8.20). Кількість розрядів мікрокоманди визначається з виразу М = п+к+пт, де п - розрядність адреси мікрокоманди, яка рівна п = к^$_2$М, де N - **кількість** мікрокоманд в пам'яті, к - **кількість** мікронаказів, необхідних для керування вузлами пристрою керування, т - **кількість** мікронаказів, необхідних для керування вузлами комп'ютера. В цьому випадку відпадає потреба в дешифраторі мікрокоманд.

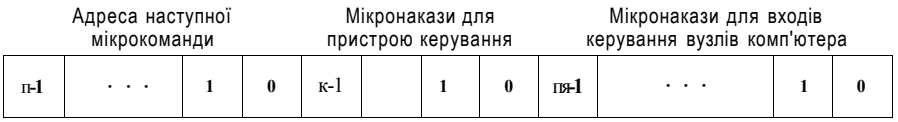


Рис. 8.20. Формат мікрокоманди при використанні горизонтального мікропрограмування

оскільки у комп'ютерах кількість мікронаказів може досягати декількох сотень, мікрокоманда в цьому випадку стає дуже широкою.

Кількість керуючих бітів мікрокоманди зменшують використовуючи наступні способи:

- Групуванням бітів. В групи об'єднуються такі мікронакази, які завжди виконуються одночасно. При цьому для групи виділяється лише один біт.
- Групування форматів. В групи об'єднуються такі мікронакази, з яких в даному такті виконується тільки один. Ці мікронакази кодуються в полі, де кількість бітів к = $\log_2 L$, де Ь - кількість мікронаказів.
- Групування мікронаказів. Групі мікронаказів виділяється один біт мікрокоманди та використовується багатотактова синхронізація.

При використанні вертикального мікропрограмування мікрокоманда складається з полів коду мікрооперації, коду умов переходу і адреси наступної мікрокоманди, як це було розглянуто раніше (рис. 8.17). Тобто формат мікрокоманди подібний до формату команди комп'ютера. Цей метод дозволяє більш ефективно використовувати поля мікрокоманди, тобто команда є коротшою, а об'єм пам'яті меншим, порівняно з горизонтальним мікропрограмуванням. Разом з тим, горизонтальне мікропрограмування є швидшим, оскільки не вимагає використання дешифраторів.

8.4. *Порівняння пристроїв керування з жорсткою логікою та пристроїв мікропрограмного керування*

Вище були розглянуті два основних методи побудови логіки формування сигналів керування. Перший з них, який одержав назву "жорсткої" або "запаяної" логіки, виражається в тому, що для кожної команди процесора існує набір логічних схем, які в потрібних тактах збуджують відповідні сигнали керування. Другий метод, який називають принципом мікропрограмного керування, передбачає формування сигналів керування за вмістом регістра мікрокоманд, в який мікрокоманди записуються із пам'яті мікрокоманд. Шляхом послідовного зчитування мікрокоманд із пам'яті в цей регістр організується потрібна послідовність сигналів керування. Завдяки тому, що мікрокоманди записуються до пам'яті, вміст якої при потребі можна частково, або повністю замінити, пристрої мікропрограмного керування мають наступні основні переваги в порівнянні з пристроями керування з жорсткою логікою:

- В них можна використовувати мікропрограми, які вже були відлагоджені та апробовані на інших комп'ютерах.
- Шляхом заміни мікропрограми в пам'яті мікрокоманд комп'ютер можна модифікувати з метою покращання технічних характеристик чи розширення функцій, і, тим самим, продовжити термін його використання.

• Можуть бути використані наробки мікропрограм в наступних поколіннях комп'ютерів однієї сім'ї.

• Мікропрограмування є простішим, ніж керування з жорсткою логікою, що спрощує розробку пристрою керування.

• Простішим є обслуговування мікропрограмованих комп'ютерів та їх відлагодження завдяки простішій заміні мікрокоманд та мікропрограм.

В швидкодії мікропрограмне керування програє керуванню з "жорсткою" логікою. Тому, завдяки створенню мов опису апаратних засобів комп'ютера та потужних програмних засобів високорівневого проектування, пристрої керування з жорсткою логікою знайшли ширше застосування в сучасних комп'ютерах.

8.5. *Короткий зміст розділу*

Пристрій керування є одним з вузлів процесора. Відомі два основних методи побудови пристроїв керування: пристрої керування з жорсткою логікою та пристрої мікропрограмного керування. В комп'ютері, крім пристрою керування центрального процесора, можуть використовуватись пристрої керування вузлами комп'ютера, наприклад, пристрої керування операційними пристроями АЛП, пристрій керування процесора введення-виведення і т. д. Принципи побудови вказаних пристроїв є ідентичними.

Розглянуто структуру та організацію роботи пристрою керування з жорсткою логікою, а також методи проектування пристроїв керування з жорсткою логікою: на основі таблиць станів, на основі тактованих елементів часової затримки, та на основі лічильників.

Метод таблиць станів передбачає розгляд пристрою керування як цифрового автомату. Цифровий автомат подано у вигляді його математичної (абстрактної) і структурної моделей, які відповідно називаються абстрактним та структурним автоматами. Абстрактну модель використано на першому етапі проектування, коли описується функціонування автомату, тобто правила переробки вхідної інформації у вихідну. На цьому етапі автомат подано у вигляді автоматів Мілі, Мура та С-автомату. Слід зауважити, що розгляд абстрактної моделі цифрового автомату дозволяє проводити його попередню оптимізацію ще до етапу структурного синтезу. Показано приклад застосування структурної моделі для побудови схеми цифрового автомату.

Описано основні засади проектування пристроїв керування з жорсткою логікою на основі синхронних елементів часової затримки та на основі лічильників. Відзначено, що вихідними даними для такого проектування є часові зміни сигналів керування, отримані з часової діаграми роботи комп'ютера.

Описана робота пристрою мікропрограмного керування. Наведені основні поняття мікропрограмування: мікронаказ, мікрокоманда, мікропрограма, горизонтальне та вертикальне мікропрограмування. Сформовані принципи, покладені в основу побудови пристрою мікропрограмного керування, серед яких в першу чергу необхідно відзначити те, що кожній команді з системи команд комп'ютера ставиться у відповідність мікропрограма її виконання в комп'ютері, всі мікрокоманди зберігаються в пам'яті мікрокоманд,

а для реалізації деякої команди необхідно зчитати з пам'яті мікрокоманд відповідну мікропрограму та подати розподілену в часі послідовність керуючих сигналів на відповідні керуючі входи вузлів комп'ютера. Розглянуті питання розміщення мікрокоманд в пам'яті, формат мікрокоманди та способи його оптимізації. Порівняння пристроїв керування з жорсткою логікою та пристроїв мікропрограмного керування показало, що перші є швидшими, а другі, завдяки тому, що мікрокоманди записуються до пам'яті, вміст якої при потребі можна частково, або повністю, замінити, є простішими при проектуванні та обслуговуванні.

8.6. Література для подальшого читання

До перших публікацій з питань проектування пристрою керування комп'ютера на основі цифрових автоматів Мілі, Мура, С-автомату, а також принципи побудови пристроїв мікропрограмного керування належать праці [5-9]. Серед перших книг з питань синтезу цифрових автоматів необхідно виділити книгу [3]. Цьому ж питанню присвячена і книга [17]. Побудова пристроїв керування на основі синхронних елементів часової затримки та лічильників описана в роботі [10]. З питань оптимізації мікропрограм доцільно почитати роботи [11-13], а в роботах [14-16] описані основні принципи мікропрограмування.

8.7. Література до розділу 8

1. Лазарев В. Г., Пийль Е. И. Синтез управляющих автоматов. - М.: Энергия, 1978. - 408 с.

2. Баранов С. И. Синтез микропрограммных автоматов. - Л.: Энергия, 1974. - 215 с.

3. Глушков В. М. Синтез цифровых автоматов. - М.: Физматгиз, 1962. - 476 с.

4. Huffman D. A. The synthesis of sequential switching circuits. 1954, vol. 257, № 3, p. 161-190; № 4. p. 275-303.

5. Kochen M. Extension of moore-shannon model for relay circuits. - "IBM Journ. Res. and Devel.", 1959, vol. 3, № 2, p. 169-186.

6. Mealy G. H. A method for synthesizing sequential circuits. - "BSTJ", 1955, vol. 34, № 5, p. 1045-1079.

7. Wilkes M. V., Stringer J. B. Microprogramming and the design of the control circuits in an electronic digital computer. - "Proc. Cambridge Philos. Soc". 1953, vol. 49, № 4, p. 230.

8. Wilkes M. V. Microprogramming. - "Proc. East. Joint Corn-put. Conf.", 1959, vol. NT-114, № 7. p. 18-20.

9. Wilkes M. V., Renwick W, Wheeler D. J. The design of the control unit of an electronic digital computer. - "Proc. of the Inst of El. Eng.", pt. B, 1958, vol. 105, № 20, 121 p.

10. Hayes J.P. Computer Architecture and Organization. McGRAW-Hill, 1988.

11. Agerwala, T.: "Microprogram Optimization: A Survey", IEEE Trans. Comput., vol. C-25, pp. 962-973, October 1976.

12. Das, S. R., D. K. Banerji, and A. Chattopadhyay: "On Control Memory Minimization in Microprogrammed Computers", IEEE Trans. Comput., vol. C-23, pp. 845-848, September 1973.

13. Davidson, S., et al.: "Some Experiments in Local Microcode Compaction for Horizontal Machines", IEEE Trans. Comput., vol. C-30, pp. 4«M77, July 1981.

14. Agrawala, A. K., and T. G. Rauscher: Foundations of Microprogramming: Architecture, Software, and Applications, Academic, New York, 1976.

15. Andrews, M.: Principles of Firmware Engineering in Microprogram Control, Computer Science Press, Potomac, Md., 1980.

16. Husson, S. S.: Microprogramming: Principles and Practices, Prentice-Hall, Englewood Cliffs, N.J., 1970.

17. Kohavi, Z., Switching and Finite Automata Theory, 2d ed., McGraw-Hill, New York, 1978.

8.8. Питання до розділу 8

1. Призначення пристрою керування
2. Що таке мікрооперація?
3. Що таке мікронаказ?
4. Що таке макрокоманда?
5. Що таке мікропрограма?
6. Назвіть місце поступлення керуючих сигналів
7. Назвіть два основних методи побудови логіки формування керуючих сигналів
8. В чому заключається принцип керування "жорсткої" або "запаяної" логіки?
9. В чому заключається принцип мікропрограмного керування?
10. Наведіть типову структурну схему пристрою керування з жорсткою логікою та поясніть її роботу
11. Для чого призначений блок синхроімпульсів?
12. Для чого призначений лічильник тактів?
13. Для чого призначені дешифратор коду операції та дешифратор тактів?
14. Назвіть методи проектування пристрою керування з жорсткою логікою
15. В чому заключається суть методу методом таблиць станів?
16. Що таке абстрактна та структурна моделі цифрового автомату?
17. Наведіть формальний опис абстрактного автомату
18. Опишіть автомат Мілі
19. Опишіть автомат Мура
20. Опишіть С-автомат
21. Опишіть таблицю переходів, таблицю виходів та таблицю з'єднань автомату Мілі
22. Опишіть таблицю переходів, таблицю виходів та таблицю з'єднань автомату Мура
23. Опишіть таблицю переходів, таблицю виходів та таблицю з'єднань С-автомату.
24. Як будується граф автомату Мілі?
25. Як будується граф автомату Мура?
26. Як будується граф С-автомату?
27. Поясніть етапи канонічного методу структурного синтезу цифрового автомату
28. Як використовують тактовані елементи часової затримки при побудові пристрою керування?
29. Як використовують лічильники при побудові пристрою керування?
30. Як будується часова діаграма роботи комп'ютера?
31. Які основні принципи покладені в основу побудови пристрою мікропрограмного керування?
32. Приведіть формат мікрокоманди

33. Приведіть структуру пристрою мікропрограмного керування та поясніть організацію його роботи.

34. Які функції контролера послідовності мікрокоманд?

35. Як формується адреса мікрокоманди?

36. Як організовані мікропрограми в пам'яті мікрокоманд?

37. Що таке горизонтальне мікропрограмування?

38. Що таке вертикальне мікропрограмування?

39. Які є способи зменшення кількості керуючих бітів мікрокоманди при використанні горизонтального мікропрограмування?

40. Порівняйте пристрої керування з жорсткою логікою та пристрої мікропрограмного керування.

Розділ 9

%(шітарШне&сі пам'шшу кжт'штера/

Пам'ять є одним із основних вузлів комп'ютера, що призначений для зберігання інформації, тобто програм і даних. Функції пам'яті забезпечуються запам'ятовуючими пристроями, які здійснюють приймання, зберігання і видачу інформації в процесі роботи комп'ютера. Процес приймання інформації в запам'ятовуючий пристрій називають записом, процес видачі інформації - зчитуванням, а спільно їх визначають як процеси звернення до пам'яті.

В даному розділі будуть розглянуті питання побудови пам'яті з довільним, впорядкованим та асоціативним доступом. Кожний тип пам'яті має свої переваги та недоліки, які визначають місце його використання в комп'ютері. Буде розглянута структура багаторівневої пам'яті та її типи, які входять до складу внутрішньої та зовнішньої пам'яті комп'ютера а також основні характеристики пам'яті: ємність, організація, швидкодія, час доступу, період звернення, вартість. Зокрема буде проведено аналіз можливих варіантів організації регістрових файлів процесорів та обмежень, які спричиняє використання багатопортового регістрового файла, а також розглянуто ряд нових структур регістрового файла: інтегрованого, розподіленого кластерного, з керованою комутацією, з віконною організацією, ієрархічного. Буде розглянута динамічна та статична організація даних в регістрових файлах та описана робота регістрового файла на базі черги з програмованою затримкою.

Будуть також розглянуті принципи роботи пам'яті з асоціативним доступом та описано чотири основні елементи її організації: з повним паралельним асоціативним доступом, з неповним паралельним асоціативним доступом, з послідовним асоціативним доступом, з частково асоціативним доступом.

Буде наведено опис варіантів побудови основної пам'яті та можливості по скороченню часу доступу до інформації. Зокрема буде розглянута блокова структура пам'яті, принципи розшарування пам'яті та нарощування її розрядності. Буде описано роботу різних типів постійного запам'ятовуючого пристрою.

При розгляді зовнішньої пам'яті буде наведений порядок розміщення інформації на магнітному диску, типи сучасних дискових систем, в тому числі масиви магнітних дисків з надлишковістю, оптична пам'ять та пам'ять на магнітних стрічках.

9.1. Типи та характеристики пам'яті комп'ютера

9.1.1. Багаторівнева структура пам'яті комп'ютера

Пам'ять комп'ютера є багаторівневою. На різних рівнях в комп'ютері використовуються різні типи пам'яті (рис. 9.1). Практично кожен функціональний вузол комп'ютера має у своєму складі пам'ять. Так, в процесорі знаходиться регістрова надоперативна пам'ять, а також постійна пам'ять для зберігання мікропрограм, яка є складовою частиною пристрою керування, та постійна пам'ять констант, яка використовується в АЛП, наприклад, у складі табличних або таблично-алгоритмічних операційних пристроїв. Між процесором і основною пам'яттю може бути включено кілька рівнів кеш пам'яті. Крім основної пам'яті, в комп'ютері є ще пам'ять процесорів введення-виведення та зовнішня пам'ять великого об'єму, реалізована на магнітних дисках, стрічках та барабанах, а також оптична та флеш. Як видно з рисунку, за місцем розташування пам'ять поділяють на внутрішню і зовнішню. Найбільш швидкісну пам'ять, тобто кеш-пам'ять першого рівня, зазвичай розміщують на одному кристалі з центральним процесором, в якому є своя регістрова пам'ять. До внутрішньої пам'яті належать також основна пам'ять, пам'ять процесора введення-виведення та кеш пам'ять другого і подальших рівнів (кеш пам'ять другого рівня може також розміщуватися на кристалі процесора). Повільну пам'ять великої ємності (магнітні й оптичні диски, магнітні стрічки та барабани) називають зовнішньою пам'яттю, оскільки до ядра комп'ютера ці пристрої підключаються аналогічно до пристроїв введення-виведення.

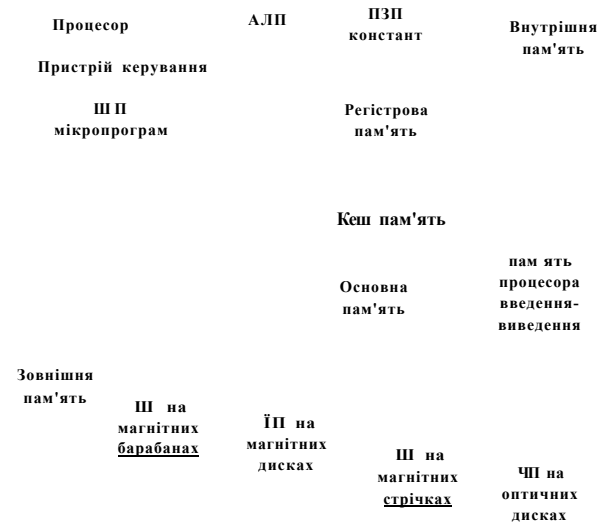


Рис. 9.1. Багаторівнева структура пам'яті комп'ютера

9.1.2. Типи пам'яті

В комп'ютері використовуються різні типи пам'яті, які, залежно від способу доступу до інформації, можуть бути класифіковані наступним чином:

• Пам'ять з довільним доступом. До цієї пам'яті в кожному такті може бути записане число або зчитане з неї число за довільною адресою. За принципом пам'яті з довільним доступом побудовано регістровий файл регістрової пам'яті процесора та основна пам'ять.

• Пам'ять із впорядкованим доступом. Із такої пам'яті дані вибираються в порядку, який визначається внутрішньою структурою пам'яті. До такої пам'яті належить зокрема пам'ять з послідовним доступом, з якої дані зчитуються послідовно одне за одним. За принципом пам'яті з послідовним доступом побудовано, зокрема, буферну пам'ять, яка використовується у пристроях введення-виведення, зовнішню пам'ять на магнітних стрічках.

• Пам'ять з асоціативним доступом. В такій пам'яті дані шукаються за їх змістом або за деякою їх ознакою. За принципом пам'яті з асоціативним доступом побудовано, зокрема, кеш пам'ять.

Існує також пам'ять з прямим доступом. Кожен запис має унікальну адресу, що відображає її фізичне розміщення на носії інформації. Звернення здійснюється як адресний доступ до початку запису з подальшим послідовним доступом до певної одиниці інформації усередині запису. В результаті час доступу до певної позиції є величиною змінною. Такий режим характерний для магнітних дисків.

Цей список може бути розширений іншими типами пам'яті, які ще не знайшли широкого застосування в комп'ютері, наприклад, пам'яттю з програмованим доступом, коли записані до пам'яті дані зчитуються в наперед заданому порядку. Кожний тип пам'яті має свої переваги та недоліки, які визначають місце використання відповідної пам'яті в комп'ютері. Розглянемо організацію роботи та проведемо аналіз названих типів пам'яті детальніше.

Найчастіше в комп'ютері використовується пам'ять з довільним доступом або адресна пам'ять. Ця пам'ять ділиться на два типи: оперативний запам'ятовуючий пристрій (ОЗП), англійський термін Random Access Memory (RAM), та постійний запам'ятовуючий пристрій (ПЗП), англійський термін Read Only Memory (ROM).

Пам'ять із довільним доступом складається з комірок, кожна з яких зберігає одиницю інформації, яка називається словом. Слова складаються із бітів із значеннями 0 або 1. В слові є п бітів, де п - довжина слова. Кожен біт має свій номер. Нумерація бітів в слові здійснюється справа-наліво, або зліва-направо. Зазвичай слово має довжину п = 2 к, де к = 0,1,2,... біт, наприклад 8 (к = 3), 32 (к = 5) і т. д.

Комірки пам'яті нумеруються, тобто кожна з них має свій номер, або адресу. Ту ж саму адресу має і слово, яке зберігається в даній комірці. Місце розміщення слова в пам'яті називається адресою слова. Якщо пам'ять може зберігати М слів розрядністю п кожне, то в якості адреси використовуються числа від 0 до М-1 (рис. 9.2). М адрес є адресним полем (простором) даного комп'ютера. Використовуючи двійкове кодування, необхідно m бітів для представлення всіх адрес, де m =[log,M]. Значення в дужках означає більше ціле. Зазвичай пам'ять комп'ютера будується так, щоб М було кратним ступеню двійки, що дозволяє ефективніше використовувати розрядну сітку адреси, а також спрощує обробку адрес. Існує поняття "організація пам'яті", яке вказує на розрядність комірок і їх кількість в пам'яті, тобто $n*2^m$. Це значення вказує також ємність пам'яті в бітах.

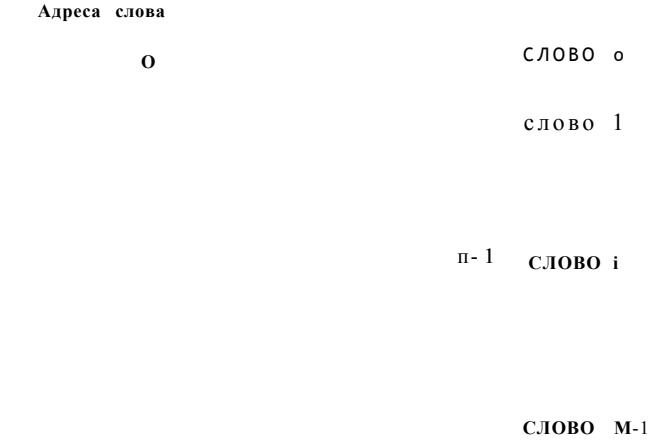


Рис. 9.2. Адресна пам'ять організацією п.2ⁿ

Пам'ять з довільним доступом виконує дві основні операції: вибірку Fetch (або зчитування Read) і запам'ятовування Store (або запис Write). Робота цієї пам'яті організована наступним чином. В режимі запису на адресний вхід пам'яті подається адреса комірки, в яку потрібно записати дане і сигналом запису це дане записується у вказану адресою комірку пам'яті. В режимі зчитування на адресний вхід пам'яті подається адреса комірки, з якої потрібно зчитати дане, і сигналом зчитування це дане зчитується з вказаної адресою комірки пам'яті (рис. 9.3).

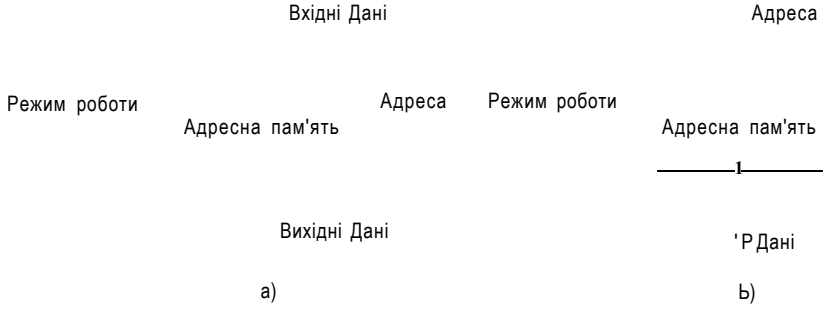


Рис. 9.3. Інтерфейс адресної пам'яті

Інтерфейс адресної пам'яті (рис. 9.3 а) включає т-розрядну шину адреси, п-розрядну шину вхідних даних, п-розрядну шину вихідних даних та однорозрядний вхід задання режиму роботи: запису або зчитування. Зазвичай використовуються також вхід дозволу використання пам'яті та вхід для подачі тактових імпульсів, необхідних для синхронізації роботи пам'яті. Слід зауважити, що у варіанті інтерфейсу адресної пам'яті, представленому на рис. 9.3 а, шини вводу та виводу даних розділені. Часто пам'ять будується з об'єднаними шинами вводу та виводу даних (тобто шина даних тут є двонаправленою) (рис. 9.3 Ь), що спрощує використання такої пам'яті в комп'ютерах із одношинною та багатошинною структурами.

Пам'ять з довільним доступом може бути реалізована у вигляді оперативної або постійної пам'яті, та у вигляді стека програмно-доступних регістрів (регістрового файла). В останньому випадку (рис. 9.4) адреса вказує номер регістра. Тут ДшА - дешифратор

адреси. Розрядність адреси дорівнює т = *[log*, к], де к - **кількість** регістрів. Як відомо, регістр - це найшвидший запам'ятовуючий елемент комп'ютера. Він використовується для короткотермінового зберігання одного слова інформації. Регістр будується на основі тригерів, кожен з яких зберігає найменшу одиницю інформації біт, тобто до нього може бути записано 0 або 1. Для зберігання п-розрядного слова регістр повинен мати п тригерів.

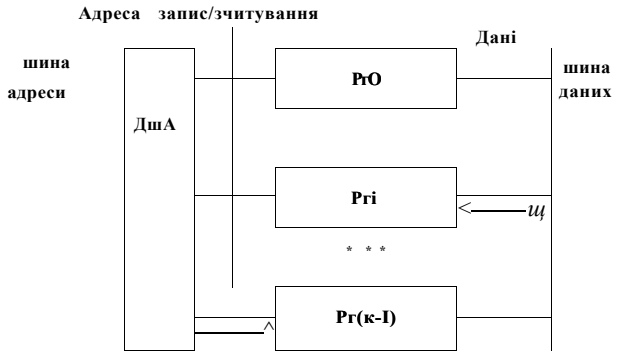


Рис. 9.4. Стек програмно-доступних регістрів

В режимі запису інформація записується лише в вибраний дешифратором адреси ДшА регістр. А в режимі зчитування на шину даних поступає лише інформація з вибраного дешифратором адреси ДшА регістра. Виходи інших регістрів в цей час знаходяться в стані високого імпедансу.

З пам'яті з впорядкованим доступом дані вибираються в порядку, який визначається її внутрішньою структурою. Пам'ять з впорядкованим доступом будується таким чином, що дані вибираються, або записуються, через вхідний регістр, а правило їх переміщення між комірками пам'яті "зашите" в зв'язках між ними.

До такої пам'яті належить зокрема пам'ять з послідовним доступом, широко використовувана в комп'ютерах, з якої дані зчитуються послідовно одне за одним в порядку їх запису або в порядку зчитування.

Пам'ять з послідовним доступом будується таким чином, що дані зчитуються або записуються до цієї пам'яті в послідовному порядку одне за одним, утворюючи деяку чергу. Зчитування здійснюється з черги слово за словом в порядку запису або в зворотному порядку. Прямий порядок зчитування забезпечується пам'яттю типу FIFO з дисципліною обслуговування "перший прийшов - перший вийшов" (First In, First Out). Обернений порядок зчитування забезпечується пам'яттю типу LIFO з дисципліною обслуговування "останній прийшов - перший вийшов" (Last In, First Out), або, що є тим же самим, пам'яттю типу FILO з дисципліною обслуговування "перший прийшов - останній вийшов" (First In, Last Out). Пам'ять з послідовним доступом, яка побудована на регістрах, часто називається стеком. Приклад такої пам'яті показано на рис. 9.5, де дані з шини через регістр даних РгД поступають до послідовно з'єднаних регістрів пам'яті РгО, Ргі,... Рг(к-І), по яких рухаються вверх або вниз залежно від значення сигналу режиму роботи.

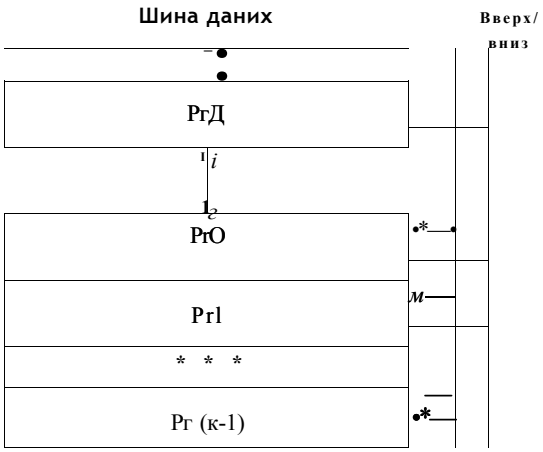


Рис. 9.5. Стекова пам'ять типу FILO

Ця пам'ять є найшвидшою та простою в реалізації. Як видно з рис. 9.6, де показано інтерфейс пам'яті з послідовним доступом з розділеними та об'єднаними входом і виходом, в цій пам'яті відсутній адресний вхід.

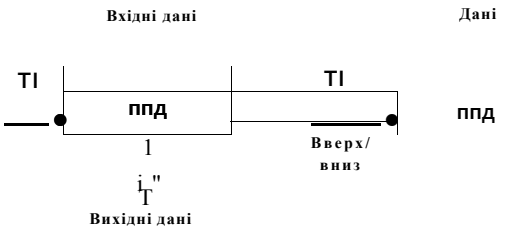


Рис. 9.6. Інтерфейс пам'яті з послідовним доступом з розділеними та об'єднаними входом і виходом

Пам'ять з розділеними входом і виходом може одночасно приймати вхідні дані та видавати попередньо прийняті дані, які рухаються з входу до виходу за допомогою тактових імпульсів ТІ. Коли ж вхід і вихід об'єднані, то дані вводяться в пам'ять тактовими імпульсами, які поступають по входу режиму роботи "вверх", а виводяться з пам'яті тактовими імпульсами, які поступають по входу режиму роботи "вниз".

Недоліком пам'яті з послідовним доступом є значний час доступу до конкретної одиниці інформації. В гіршому випадку для такого доступу може виникнути потреба в перегляді всього об'єму пам'яті.

Пам'ять з асоціативним доступом (або асоціативна пам'ять) зберігає разом з даними і їх ознаки, в ролі якої може бути і саме дане. Ядро пам'яті з асоціативним доступом показане на рис. 9.7. Роль комірок цієї пам'яті виконують регістри. Числа записуються в довільні вільні регістри пам'яті. Дані вибираються з такої пам'яті на основі збігу їх ознак з заданою. Для цього ознаки даних з усіх регістрів пам'яті поступають на схему порівняння, де порівнюються з заданою ознакою із регістра ознаки, і на вихід пам'яті поступають дані, ознаки яких збігаються з заданою.

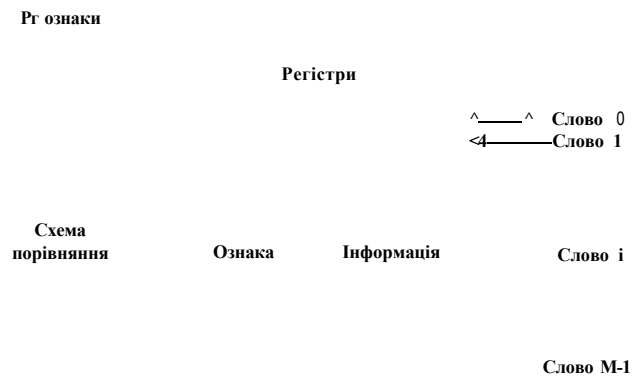


Рис. 9.7. Ядро пам'яті з асоціативним доступом

В паралельній асоціативній пам'яті одночасно порівнюються з аргументом всі розряди всіх полів ознак пам'яті (так званий одночасний пошук по слову). В послідовній асоціативній пам'яті одночасно порівнюються з бітом аргументу по одному біту кожного поля ознаки (так званий порозрядний пошук). Можливість паралельної роботи є основною перевагою асоціативної пам'яті. З неї можна одночасно зчитати всі дані з однаковими ознаками. Можливість швидкого одночасного перегляду даних в такій пам'яті забезпечує їй широке використання в комп'ютерах, зокрема за принципом пам'яті з асоціативним доступом часто будується кеш пам'ять.

Пам'ять буває енергозалежною та енергонезалежною. У енергозалежній пам'яті інформація може бути спотворена або втрачена при відключенні джерела живлення. У енергонезалежній пам'яті записана інформація зберігається і при відключенні напруги живлення. Магнітна і оптична пам'ять є енергонезалежними. Напівпровідникова пам'ять може бути як енергозалежною, так і енергонезалежною.

Залежно від технології виготовлення розрізняють наступні основні типи пам'яті: напівпровідникова пам'ять, пам'ять з магнітним носієм інформації, використовувана в магнітних дисках і стрічках, та пам'ять з оптичним носієм - оптичні диски.

9.1.3. Основні характеристики пам'яті

До основних характеристик пам'яті належать: ємність, організація, швидкодія та вартість.

Ємність виражають числом бітів або байтів, які можуть зберігатися в пам'яті. Добавлення до одиниці виміру множника К означає множення на 210 = 1024, множника М - множення на 220 = 1048576, а множника Г - множення на 230 = 1073741824.

Важливою характеристикою пам'яті є її організація, тобто кількість комірок в пам'яті М = 2ⁿ та розрядність кожної комірки п. Хоча організація пам'яті вказує на її ємність, це є самостійна характеристика, оскільки при тій же ємності організація пам'яті може бути різною.

Для кількісної оцінки швидкодії пам^яті використовують наступні характеристики:

- Час зчитування та час запису, або час зчитування/запису. Час зчитування - це інтервал часу між моментами подачі сигналу зчитування та появи слова на виході пам'я-

ті. Час запису - це інтервал часу після моменту подачі сигналу запису, достатній для запам'ятовування слова в пам'яті. Мінімально допустимий інтервал між послідовними зчитуванням та записом називається часом зчитування/запису. Цей час може перевищувати час зчитування або запису, оскільки після виконання цих операцій може бути потрібний час на відновлення початкового стану пам'яті.

- Час доступу. Для пам'яті з довільним доступом час доступу рівний інтервалу часу від моменту надходження адреси до моменту, коли дані заносяться в пам'ять або стають доступними. Для пам'яті з рухомим носієм інформації - це час, що витрачається на установку головки запису/зчитування (або носія) в потрібну позицію.
- Тривалість циклу пам'яті або період звернення. Дане поняття застосовується до пам'яті з довільним доступом, для якої воно означає мінімальний час між двома послідовними зверненнями до пам'яті. Період звернення включає час доступу плюс деякий додатковий час. Додатковий час може бути потрібен для затухання сигналів на лініях, а в деяких типах пам'яті, де зчитування інформації приводить до її руйнування, - для відновлення зчитаної інформації.

Вартість пам'яті прийнято оцінювати відношенням її загальної вартості до ємності в бітах або байтах, тобто вартістю зберігання одного біта або байта інформації. На рис. 9.8 показана зміна з роками вартості зберігання одного МВ інформації та часу доступу до одиниці інформації в не для трьох типів пам'яті: статичної напівпровідникової пам'яті з довільним доступом (SRAM), динамічної напівпровідникової пам'яті з довільним доступом (DRAM) та пам'яті на магнітних дисках (Disk).

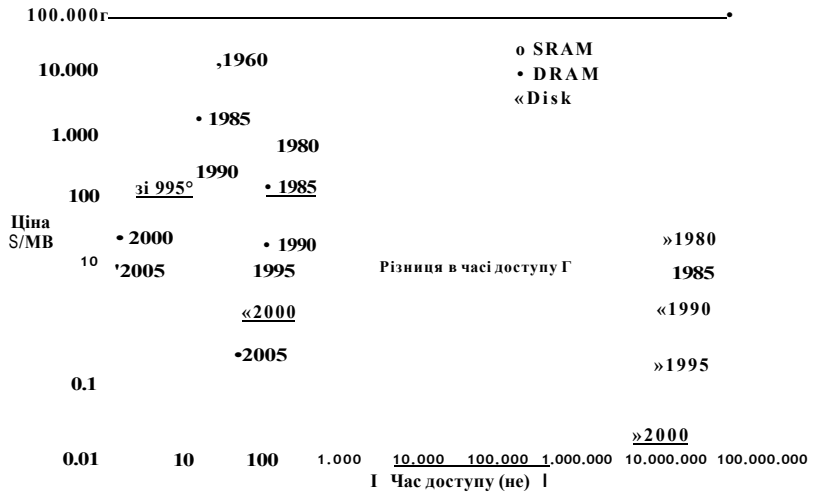


Рис. 9.8. Зміна з роками вартості зберігання одного МВ інформації та часу доступу до одиниці інформації

Як бачимо, ціна зберігання одного МВ інформації в статичній пам'яті залишається досить високою (більше, ніж в 10 разів вищою, ніж в динамічній пам'яті). Але, і час доступу до даних в цій пам'яті є значно меншим (майже в 100 разів). При цьому, як видно з рисунку, зберігається суттєвий розрив між ціною зберігання одного МВ інформації та часом доступу до даних в статичній та динамічній пам'яті, і дисковій пам'яті.

9.2 Регістровий файл процесора

9.2.1. Типи регістрових файлів

Регістровий файл (надоперативна пам'ять) - це набір програмно-доступних регістрів, які знаходяться в регістровій пам'яті процесора. В сучасних процесорах регістровий файл займає одне з центральних місць. Він використовується для локального збереження операндів, адрес команд та даних, індексів, а також дозволяє організовувати ефективний обмін даними між операційними пристроями процесора та основною пам'яттю. Вибір ефективної організації регістрового файла є одним із підходів, що дозволяє підвищити продуктивність комп'ютера. Як правило, організація регістрового файла належить до технічних характеристик комп'ютера.

Розглянемо деякі історичні аспекти розвитку організації регістрового файла процесора. Програмно-доступні регістри почали використовуватися з початку 1960-х років. В 1964 році фірма IBM розробила серію універсальних комп'ютерів IBM/360 для наукових та комерційних розрахунків, в процесорах яких був використаний регістровий файл, що включав 16 32-розрядних цілочисельних регістрів та 16 64-розрядних регістрів з рухомою комою. Також в 1964 році був виготовлений перший суперкомп'ютер для наукових розрахунків CDC 6600, процесор якого мав регістровий файл, що включав 24 регістри. В 1977 році був спроектований перший векторний суперкомп'ютер Сгау-1, регістровий файл якого мав ієрархічну структуру. Його було поділено на файл основних регістрів і файл другорядних (фонових) регістрів. Кількість основних регістрів була меншою, що дозволяло забезпечити швидкий доступ до них, тоді як кількість другорядних регістрів була більшою, проте доступ до них був повільнішим. Операнди тривалого зберігання розміщувалися в другорядних регістрах і переміщувалися в основні регістри при потребі їх використання. Сгау-1 загалом містив 656 регістрів даних і адрес (включаючи векторні регістри, але не враховуючи регістрів керування). На особливу увагу заслуговує група з восьми векторних регістрів. Кожний такий регістр міг вміщувати 64-х елементний вектор з рухомою комою. Однією 16-ти розрядною командою можна було додати, відняти чи перемножити два вектори.

З наведених прикладів видно, що для організації високопродуктивних обчислень важливо не тільки мати велику кількість регістрів, але і забезпечити швидкий доступ до даних, які в них зберігаються, за рахунок ефективної організації регістрового файлу.

На рис. 9.9 приведена класифікація типів регістрових файлів процесора. В них використовується статична організація збереження даних, тобто при записі і читанні даних вказується безпосередня адреса регістра, яка є незмінною під час виконання програми.

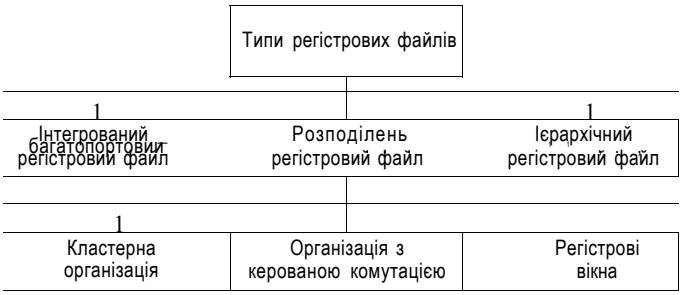


Рис. 9.9. Класифікація шипів регістрових файлів

Розглянемо наведені на рис. 9.9 типи регістрових файлів детальніше.

9.2.2. *Інтегрований багатопортовий регістровий файл*

В інтегрованому багатопортовому регістровому файлі забезпечується доступ до будь-якого регістру з кожного його порта. На рис. 9.10 показана структура інтегрованого регістрового файла з двома портами зчитування (гсіі, гсІ2), та одним портом запису (\УСІ), який містить 32 32-розрядних регістри (Рч0-Р31). В регістр РчО запис заборонений, оскільки в ньому зберігається константа нуль. Вихід кожного регістра під'єднаний до відповідних інформаційних входів мультиплексорів МП1 та МП2. Адреси (**ГБІ**, гв2), що подаються на входи керування мультиплексорів, визначають номери регістрів, з яких читаються дані. Для запису використовується демультиплексор ДМП, через який на відповідний регістр подається сигнал запису даних (**УЄ), ЩО** поступає з вхідної шини, та адреса регістра Процес запису синхронізується тактовими імпульсами, що подаються на вхід сік. Кожний з регістрів має вхід дозволу запису даних Еп, інформаційний вхід Б, та інформаційний вихід 0.

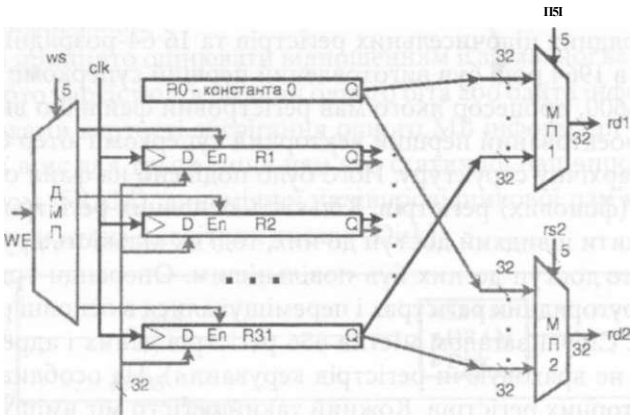


Рис. 9.10. Структура інтегрованого регістрового файла

Із збільшенням **кількості** регістрів регістрового файла зростає час доступу до даних, так як ускладнюються мультиплексори та демультиплексори, а також відповідно збільшується розмір кристалу процесора.

Збільшення кількості портів зчитування та запису призводить до збільшення кількості відповідно мультиплексорів та демультиплексорів, а також суттєво збільшує кількість шин передачі даних, що ускладнює реалізацію регістрового файла. Крім того, з'являється можливість виникнення конфліктних ситуацій при спробі запису даних до тих самих регістрів. Тому інтегрований регістровий файл доцільно використовувати в комп'ютерах, де не вимагається велика кількість портів. Наприклад, регістрові файли комп'ютерів RS/6000 фірми IBM, та SuperSPARC фірми Sun, містять 4 порти для зчитування і 2 порти для запису.

9.2.3. *Розподілений регістровий файл*

Використання розподілених регістрових файлів дозволяє зменшити площу кристала, яку займає регістрова пам'ять процесора. Розподілення регістрового файла здійснюється шляхом поділу функціональних елементів процесора на групи, кожна з яких має свій

локальний регістровий файл. Відповідно, такі регістрові файли містять меншу кількість портів та регістрів, що зменшує затрати обладнання на їх реалізацію. Як ми вже бачили, існує три типи розподілених регістрових файлів: кластерні, з керованою комутацією та з віконною організацією. Розглянемо принципи їх побудови.

9.2.3.1. Кластерний розподілений регістровий файл

Локальні регістрові файли можуть бути повністю незалежними, тобто дані для певної групи функціональних елементів процесора доступні тільки з конкретного локального регістрового файла. Такий розподілений регістровий файл, який складається з незалежних локальних регістрових файлів, відповідно до наведеної вище класифікації, називають кластерним.

Кластерні регістрові файли здебільшого використовуються у векторних процесорах. В таких процесорах одна операція над всіма п компонентами векторних операндів задається однією командою. Кожен з таких векторних операндів міститься в локальному регістровому файлі біля відповідного функціонального елемента процесора, і відпадає необхідність у повноцінних зв'язках між локальними регістровими файлами.

На рис. 9.11 показана кластерна організація регістрового файла процесора TMS320 C64x фірми Texas Instruments. В цьому процесорі функціональні елементи поділені на підмножину A (LI, SI, M l , D1) та підмножину B (L2, S2, M2, D2). Функціональні елементи процесора L (LI, L2) виконують арифметичні операції та операції порівняння. Функціональні елементи процесора S (SI, S2) виконують арифметично-логічні операції та команди керування. Функціональні елементи процесора M (M1, M2) виконують множення 16-ти розрядних операндів, а функціональні елементи процесора D (D1, D2) виконують арифметичні операції, та виконують роль генераторів адрес.

Рис. 9.11. Організація регістрового файла процесора TMS320C64x фірми Texas Instruments

Кожна підмножина має свій локальний регістровий файл. Для пересилання даних з регістрового файла однієї підмножини до функціональних елементів іншої підмножини використовуються додаткові мультиплексори МП. Фактично, можливість повноцінної пересилки обмежена, оскільки для цього в кожному з локальних регістрових файлів виділяється тільки один порт. Саме з цієї причини організацію регістрового файла TM8320C64x можна назвати кластерною. Зрозуміло, що описана організація регістрового файла для приведеного типу процесорів дозволяє зменшити затрати обладнання та прискорити час доступу до регістрів у порівнянні з інтегрованим регістровим файлом того ж об'єму.

9.2.3.2. Розподілений регістровий файл з керованою комутацією

Якщо будь-якій групі функціональних елементів процесора доступні дані з будь-якого локального регістрового файла, то це буде інший варіант організації розподіленого регістрового файла. Його називають розподіленим регістровим файлом з керованою комутацією.

Розподілений регістровий файл з керованою комутацією, структура якого приведена на рис. 9.12, складається з локальних регістрових файлів (ЛРФ), що зв'язані з функціональними елементами (ФЕ) процесора через комутуючу мережу, керування якою виконує контролер комутуючої мережі (ККМ).

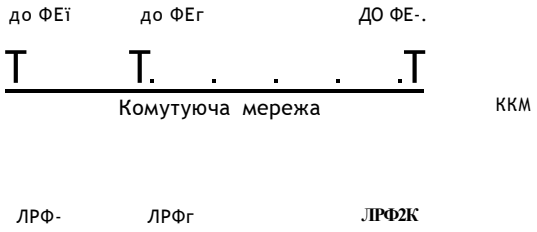


Рис. 9.12. Розподілений регістровий файл з керованою комутацією

Тут для кожного входу функціонального елемента ФЕ процесора виділяється окремий локальний регістровий файл ЛРФ з одним портом для зчитування та одним портом для запису даних. Тому кількість локальних регістрових файлів є в два рази більшою кількості функціональних елементів процесора. Об'єм (кількість регістрів) кожного локального регістрового файла є сталим, і не залежить від кількості функціональних елементів.

Порівняно з багатопортовим інтегрованим регістровим файлом тут додатково введена комутуюча мережа та її контролер, що потребує додаткових затрат обладнання.

Час доступу до регістрів розподіленого регістрового файла з керованою комутацією складається з затримки в комутуючій мережі, яка не є значною із-за невеликої кількості входів, та із часу доступу до локального регістрового файла, який не залежить від кількості функціональних елементів процесора і є сталою величиною для заданої кількості регістрів. В сумі ця величина є меншою часу доступу до регістра багатопортового інтегрованого регістрового файла з тією ж сумарною кількістю регістрів.

9.2.3.3. Розподілений регістровий файл з віконною організацією

В розглянутих структурах розподілених регістрових файлів доступ до регістрів для обчислювальних модулів розмежовувався в просторі. Існує інша організація розподілених регістрових файлів, яка передбачає розподілений доступ до регістрів в часі. Така організація отримала назву регістрових вікон. В основному вона використовується для зменшення довжини команди та споживаної потужності процесора, а також дає можливість нарощувати кількість регістрів без зміни формату команди. За затратами обладнання та часом доступу розподілений регістровий файл із віконною організацією близький до розподіленого регістрового файла з керованою комутацією. На рис. 9.13 показано структуру розподіленого регістрового файла з віконною організацією. Тут регістровий

файл поділений на N вікон, кожному з яких відповідає один з регістрових файлів РФО - РФ N-1. Для зміни активного вікна використовується сигнал вибору файла з блоку керування.

Рис. 9.13. Розподілений регістровий файл з віконною організацією

Потрібно зазначити, що організація регістрового файла у вигляді регістрових вікон є неефективною для систем із високим рівнем паралелізму, оскільки вона дозволяє збільшити кількість регістрів, але не дозволяє організувати паралельний доступ до будь-якого регістру в один момент часу.

9.2.4. *Ієрархічний регістровий файл*

В універсальних комп'ютерах важливим є той факт, що звернення процесора до пам'яті завжди локалізовано в невеликому діапазоні змін її адрес. Саме він дозволяє застосовувати ієрархічну організацію системи пам'яті, аби розв'язати невідповідність швидкодій процесора та пам'яті, яка передбачає введення кількох рівнів кеш пам'яті та ефективну організацію обміну інформацією між цими рівнями. Однак існує множина алгоритмів з великими наборами вхідних даних, що характеризуються дуже низьким рівнем повторного використання тих же комірок пам'яті. При виконанні таких алгоритмів кеш пам'ять є зайвою ланкою в системі пам'яті комп'ютера. В цьому випадку для того, щоб більш ефективно реалізувати доступ до основної пам'яті, будуються ієрархічні регістрові файли шляхом поділу інтегрованого регістрового файла на регістровий файл великого об'єму з декількома портами для виконання операцій з пам'яттю і регістровий файл меншого об'єму з великою кількістю портів для обслуговування функціональних елементів процесора. Такий поділ дозволяє зменшити затрати обладнання на реалізацію ієрархічного регістрового файла у порівнянні з інтегрованим.

Регістровий файл процесора SPARC фірми Sun можна з деякими застереженнями віднести до ієрархічних регістрових файлів, оскільки крім віконної організації він містить асоціативну регістрову кеш пам'ять.

9.2.5. *Динамічна та статична організація збереження даних в регістрових файлах*

В розглянутих структурах регістрових файлів дані зберігаються в конкретних регістрах. Доступ до цих регістрів відбувається шляхом задания їх адреси. Така статична організація збереження даних часто є неефективною для виконання деяких класів алгоритмів. Наприклад, задачі роботи із зображеннями, звуком, стиском даних в реальному масштабі часу вимагають від регістрового файлу одночасного прийому вхідних масивів даних з декількох вхідних каналів, їх затримки на потрібну кількість тактів та видачі декількома вихідними каналами раніше прийнятих масивів даних з впорядкуванням за заданим законом. Для вирішення таких задач доцільно використовувати динамічне збереження даних в регістрових файлах. Основою апаратних рішень динамічних регістрових файлів є черги. Динамічний принцип збереження даних для складних задач обробки інформації в реальному масштабі часу на основі черг має наступні переваги:

- доступ до черги є простіший, ніж доступ до статичного регістрового файлу, оскільки відсутні адреси;
- збільшення кількості регістрів у черзі не призводить до вагомого зростання складності регістрового файлу та не сповільнює його роботу;
- простота оптимізації структури регістрового файлу під заданий алгоритм;
- необхідна менша кількість адресних ліній, порівняно зі статичними регістровими файлами;
- вирішується проблема розподілу регістрів, оскільки дані записуються не в конкретний регістр, а в чергу.
- можливість реалізації регістрового файла на основі модулів напівпровідникової пам'яті з довільним доступом.

Слід зауважити наступну статистику: 40 % результатів виконання арифметичних операцій використовуються на наступному такті виконання програми, а ще 45 % - через такт. Такі дані ще раз переконують в ефективності реалізації регістрових файлів на основі черг.

Регістровий файл на основі черг може бути побудований за принципом пам'яті з послідовним доступом типу FIFO. Недоліком такої організації є те, що для алгоритмів, в яких умови сумісності черг не виконуються, різко збільшуються затрати обладнання на його реалізацію.

Кращою є реалізація регістрового файлу на основі черг, побудованого на пам'яті з програмованою затримкою. Структура такого регістрового файла, який можна віднести до розподілених регістрових файлів із керованою комутацією та динамічним збереженням даних, наведена на рис. 9.14. Такий тип структури має найбільші можливості врахування особливостей алгоритмів, що дозволяє виконувати алгоритми з максимальною продуктивністю.

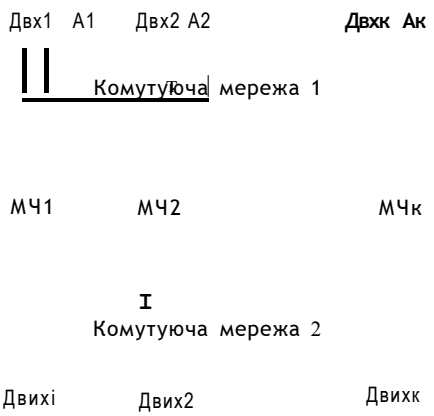


Рис. 9.14. Регістровий файл на базі черги з програмованою затримкою

Тут модуль черги (МЧ) побудовано на основі двопортової адресної пам'яті. Один порт дозволяє читати дані з пам'яті, а інший - записувати дані в пам'ять. Зчитування даних виконується аналогічно до пам'яті FIFO, тобто через кожний тактовий імпульс читається комірка за адресою, більшою на одиницю від попередньої. Збільшення адреси забезпечується лічильником. Алгоритм просування даних в модулі представляє собою генерування сигналу зчитування для потрібної комірки пам'яті даного модуля. Таким чином кожна комірка є пронумерована і зберігає свій номер. Лічильник пам'яті генерує послідовні значення, що визначають номер комірки, яка повинна видавати дані назовні. Отже, як бачимо, система зчитування дуже проста - дані зчитуються тоді, коли приходить їх черга. А впорядкування здійснюється на етапі запису: до комірки якого номера запишуться дані, на такті з тим номером вони і видадуться назовні. Адреса запису формується за допомогою лічильника та суматора шляхом додавання значення затримки (адреса А) та значення лічильника. Двхі - і-й вхідний порт даних, Двихі - і-й вихідний порт даних (і= 1,2,...,к).

9.3. Пам'ять з асоціативним доступом

9.3.1. Організація та типи пам'яті з асоціативним доступом

Пам'ять з асоціативним доступом (або асоціативна пам'ять) широко використовується в сучасних комп'ютерах. У першу чергу за принципом пам'яті з асоціативним доступом побудовано запам'ятовуючі пристрої кеш пам'яті більшості комп'ютерів. Крім того, за принципами пам'яті з асоціативним доступом побудовано буферну пам'ять в конвеєрних процесорах, зокрема це буфер попередньої вибірки команд, буфер впорядкування і т. д. Асоціативна пам'ять ефективна також для побудови пристроїв для виконання операцій числового пошуку (min, max, сортування і т. д.).

Пам'ять з асоціативним доступом передбачає зберігання разом із даними і їх ознаки, в ролі якої може бути і саме дане. Дані вибираються з такої пам'яті на основі збігу їх ознак із заданою. Тому цю пам'ять іще називають пам'яттю, що адресується за вмістом або за даними. Ядро пам'яті з асоціативним доступом приведене на рис. 9.7. На рис. 9.15 приведена детальніша структура одного з варіантів побудови пам'яті з асоціативним доступом.

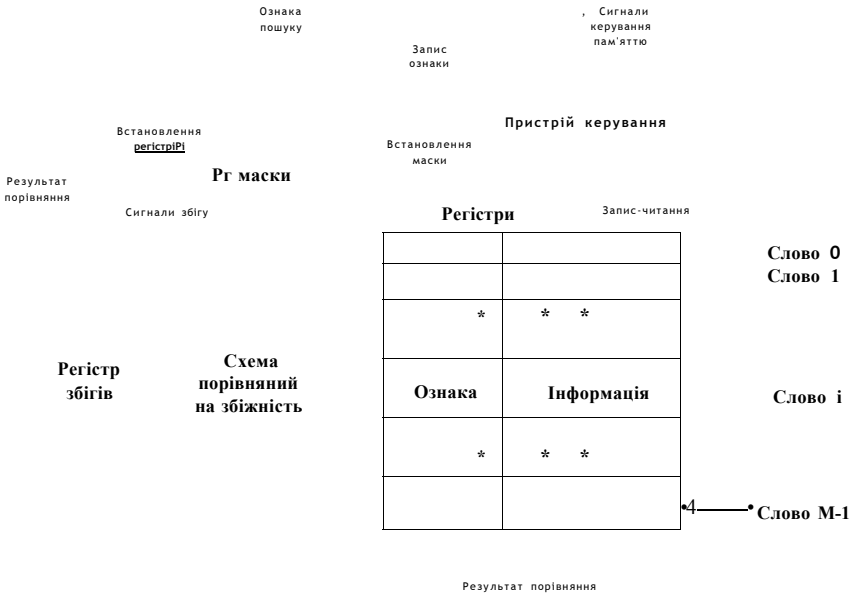


Рис. 9.15. Структура пам'яті з асоціативним доступом

Ця пам'ять включає:

- масив регістрів для зберігання N т-розрядних слів, в кожному з яких частина розрядів зайнята ознаками, а іншу частину розрядів займає службова інформація;
- регістр асоціативної ознаки, куди поміщається код шуканої інформації (ознака пошуку). Розрядність цього регістра зазвичай є меншою чи рівною довжині слова п;
- схему порівняння на збіжність, яка використовується для проведення паралельного або послідовного порівняння бітів ознак пошуку всіх слів, що зберігаються, з відповідними бітами асоціативної ознаки пошуку, і вироблення сигналів збігу;
- регістр збігів, в якому кожному регістру масиву пам'яті відповідає один розряд, в який заноситься одиниця, якщо всі розряди ознаки слова відповідного регістра співпали з однойменними розрядами асоціативної ознаки пошуку;
- регістр маски, який дозволяє заборонити порівняння певних бітів;
- пристрій керування, який на основі зовнішніх сигналів керування здійснює запис асоціативної ознаки пошуку до відповідного регістра, встановлює маску в регістрі маски, та на основі даних порівняння на збіжність з регістра збігів здійснює зчитування даних із регістрів асоціативної пам'яті, а також запис до них вхідних даних.

При зверненні до пам'яті з асоціативним доступом, спочатку сигналами з блоку керування розряди регістра маски, які не повинні враховуватися при пошуку інформації, встановлюються в нульові значення, і всі розряди регістра збігів встановлюються в одиничний стан. Після цього в регістр асоціативної ознаки заноситься код шуканої інформації (ознака пошуку) і починається її пошук, в процесі якого схема порівняння на збіжність одночасно порівнює перший біт ознак даних всіх регістрів пам'яті, з першим бітом асоціативної ознаки пошуку. Ті схеми, які зафіксували неспівпадіння, формують сигнал, що переводить відповідний біт регістра збігів у нульовий стан. Так само відбувається процес пошуку і для решти незамаскованих бітів ознаки пошуку.

У результаті, одиниці зберігаються лише в тих розрядах регістра збігів, які відповідають регістрам, де знаходиться шукана інформація. Конфігурація одиниць в регістрі збігів використовується в якості адрес, за якими проводиться зчитування даних із пам'яті.

Внаслідок того, що результати пошуку можуть мати кілька варіантів, вміст регістра збігів подається на пристрій керування, де формуються сигнали сповіщення про результати порівняння, а саме про те, що шукана інформація не знайдена, міститься в одному чи кількох регістрах. Тому, при зчитуванні спочатку проводиться аналіз результатів порівняння. Потім, при наявності інформації про те, що шукана інформація не знайдена, зчитування відміняється, при повідомленні, що шукана інформація міститься в одному регістрі, зчитується слово, на яке вказує одиниця в регістрі збігів, а при повідомленні, що шукана інформація міститься в кількох регістрах, скидається найстарша одиниця в регістрі збігів і витягується відповідне їй слово. Шляхом повторення цієї операції послідовно зчитуються всі слова.

Запис у пам'ять з асоціативним доступом проводиться без вказівки конкретної адреси, в перший вільний регістр. Для пошуку вільного регістра виконується операція зчитування, в якій не замасковані тільки службові розряди, що показують, як давно проводилося звернення до кожного регістра, і вільним вважається або порожній регістр, або той, який найдовше не використовувався.

Головна перевага пам'яті з асоціативним доступом визначається тим, що час пошуку інформації залежить тільки від числа розрядів в ознаці пошуку і швидкості опиту розрядів, і не залежить від числа регістрів пам'яті.

Разом з тим, ця пам'ять також має ряд недоліків, до основних з яких слід віднести наступні:

- Необхідність двократного звернення до однієї комірки пам'яті при записі і при зчитуванні числа в два рази сповільнює взаємодію пристроїв комп'ютера з пам'яттю з асоціативним доступом в порівнянні з варіантом пам'яті, в якій можливе лише одне звернення.
- Ця пам'ять має досить складну організацію, яка вимагає забезпечення доступу до кожної комірки з входу та виходу пам'яті або з об'єднаного входу-виходу пам'яті, причому при зчитуванні необхідно проводити порівняння заданої ознаки з ознаками даних в регістрах пам'яті, а також забезпечити пошук даних, ознаки яких співпали із заданою. Це вимагає великих затрат на елементи доступу та сповільнює роботу пам'яті.
- При створенні на основі пам'яті з асоціативним доступом багатоканальної (багатопортової) пам'яті необхідно забезпечувати одночасний доступ із кожного каналу до кожної комірки пам'яті, з одночасним порівнянням всіх ознак у комірках із шуканими ознаками, а також забезпечити пошук даних, ознаки яких співпали із заданою, що є проблематичним завданням, особливо при великих об'ємах пам'яті.

Базуючись на концепції асоціативного пошуку можна побудувати цілу низку структур пам'яті з асоціативним доступом. Конкретна структура визначається тим, як поєднані наступні чотири основні елементи її організації:

- вид пошуку інформації;
- спосіб зчитування інформації при множинних збігах;
- спосіб запису інформації;
- варіант порівняння ознак.

У кожному конкретному застосуванні пам'яті з асоціативним доступом завдання пошуку інформації може формулюватися по різному.

При простому пошуку, який був розглянутий вище, потрібен повний збіг всіх розрядів ознаки пошуку з однойменними розрядами ознак слів, що зберігаються в регістрах пам'яті.

При складному пошуку можуть ставитись наступні завдання:

- Пошуку слів із максимальним або мінімальним значенням ознаки. При цьому багатократна вибірка з пам'яті слова з максимальним або мінімальним значенням ознаки (з виключенням його з подальшого пошуку), по суті, є впорядкованою вибіркою інформації.
- Пошук слів, ознака яких порівняно з асоціативною ознакою пошуку є найближчим більшим або меншим значенням. Ця операція також забезпечує проведення впорядкованої вибірки інформації.
- Знаходження слів, чисельне значення ознаки в яких є більшим або меншим заданої величини. Подібний підхід дозволяє вести пошук слів з ознаками, що лежать всередині або поза заданими межами.

Очевидно, що реалізація складних методів пошуку вимагає внесення відповідних змін у структуру пам'яті з асоціативним доступом.

Важливим питанням є організація зчитування з пам'яті з асоціативним доступом інформації, коли з асоціативною ознакою пошуку співпадають ознаки декількох слів. У цьому випадку застосовується один із двох підходів: використовується ланцюг черговості або проводиться опитування. Використання ланцюга черговості дозволяє проводити зчитування слів у порядку зростання номера регістра пам'яті незалежно від величини асоціативних ознак. При проведенні опитувань впорядковуються ті розряди, які були замасковані, і не брали участі в пошуку. В іншому варіанті для цих цілей виділяються спеціальні розряди регістрів.

Істотні відмінності в структурі пам'яті з асоціативним доступом можуть бути пов'язані з вибраним принципом запису інформації. Раніше був описаний варіант із записом у незайнятий регістр з найменшим номером. На практиці застосовуються й інші способи, наприклад, за співпадінням ознак, за чергою та інші. Найскладнішим є запис із сортуванням інформації на вході пам'яті з асоціативним доступом за величиною ознаки слова. Тут місцеположення регістра, куди буде поміщене нове слово, залежить від співвідношення ознаки записуваного слова і ознак слів, що вже зберігаються в пам'яті з асоціативним доступом.

При побудові пам'яті з асоціативним доступом вибирають один з чотирьох варіантів порівняння ознак, тобто організації перегляду вмісту пам'яті: паралельно за всіма словами, послідовно за всіма словами, паралельно за всіма розрядами, послідовно за всіма розрядами. Ці варіанти можуть комбінуватися. Названі чотири варіанти порівняння ознак визначають принципи побудови схеми порівняння на збіжність. Відповідно до цих принципів, може бути запропоновано кілька типів організації пам'яті з асоціативним доступом. У пам'яті з повним паралельним асоціативним доступом одночасно порівнюються всі розряди ознак всіх слів пам'яті (так званий одночасний пошук по слову). Це найшвидша та найскладніша з точки зору затрат обладнання структура пам'яті з асоціативним доступом. Можливе порівняння не всіх розрядів ознак слів пам'яті, а по частинах, аж до порозрядного порівняння, коли одночасно порівнюються з бітом асоціативної ознаки пошуку по одному біту ознаки кожного слова (так званий порозрядний

пошук). Це буде структура пам'яті з неповним паралельним асоціативним доступом та з послідовною обробкою розрядів ознак слів. Можливе також проведення порівняння одночасно ознак не всіх слів, а по групах. Тоді це буде структура пам'яті з послідовним асоціативним доступом із порозрядним та паралельним порівнянням.

Розглянемо названі чотири основні типи організації пам'яті з асоціативним доступом детальніше.

9.3.2. *Пам'ять з повним паралельним асоціативним доступом*

Перший тип організації пам'яті з асоціативним доступом - пам'ять з повним паралельним асоціативним доступом. Йрикладом конкретного технічного виконання пам'яті такого типу може бути пам'ять комп'ютерної системи РЕРЕ фірми Bell Laboratories. На рис. 9.16 показана загальна структура пам'яті з повним паралельним асоціативним доступом. Пам'ять складається з деякої кількості N комірок, в яких зберігаються дані. Кожна комірка може зберігати слово з деякою кількістю п двійкових розрядів (ширина слова). На рис. 9.17 приведена будова комірки цієї пам'яті. Кожна така комірка може зберігати дані та порівнювати їх з ознакою пошуку паралельно з іншими комірками. Тут і далі розглядається випадок, коли з ознакою пошуку порівнюються всі розряди слова, хоча зрозуміло, що може бути взята і його частина. Дані та сигнали керування надходять від пристрою керування до кожної комірки пам'яті паралельно.

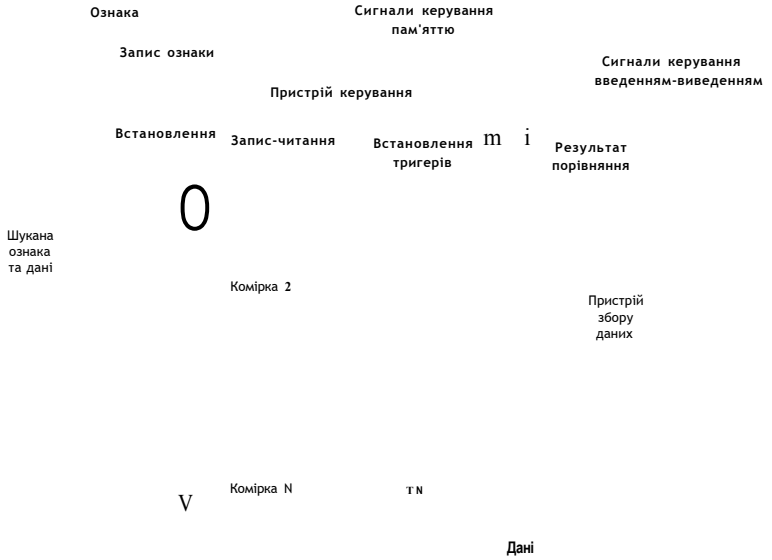


Рис. 9.16. Структура пам'яті з повним паралельним асоціативним доступом

Кожна комірка пов'язана з пристроєм керування теговим розрядом Ті. Перед початком порівняння пристрій керування надсилає команду встановлення всіх тегових розрядів в "1". Коли пристрій керування формує команду порівняння, то кожна комірка, в якій міститься слово, ознака якого не збігається з ознакою пошуку, формує сигнал, який викликає скид відповідного тегового розряду в "0". Після виконання даної команди встановленими в "1" залишаться тільки ті розряди, які відповідають коміркам, де відбувся збіг з ознакою пошуку.

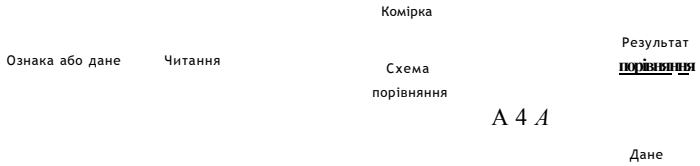


Рис. 9.17. Будова комірки пам'яті з повним паралельним асоціативним доступом

9.3.3. Пам'ять з неповним паралельним асоціативним доступом

Другий тип організації пам'яті з асоціативним доступом - пам'ять з неповним паралельним асоціативним доступом. В цій пам'яті проводиться послідовна обробка розрядів слів. Тут логічні операції порівняння визначені тільки для одного або декількох розрядів ознак даних, а не для всіх розрядів одночасно, як це є в пам'яті з повним паралельним асоціативним доступом. Наприклад, якщо порівняння визначені тільки для одного розряду ознак даних, і потрібно здійснити пошук даних, ознаки яких займають розряди 0-20, то спочатку потрібно сформувати команду для одночасного порівняння лише одного 0-го біту в кожному записі, потім 1-го, і т. д. Пам'ять даного типу має подібну загальну структуру до попередньої з тією відмінністю, що операції в пам'яті здійснюються у вигляді циклів операцій над окремими розрядами і тривалість операцій пропорційна довжині поля ознаки, яке аналізується. Комірка даної пам'яті зображена на рис. 9.18.

Рис. 9.18. Комірка пам'яті з неповним паралельним асоціативним доступом з послідовною обробкою розрядів слів

Прикладом пам'яті з неповним паралельним асоціативним доступом, в якій здійснюється послідовна обробка розрядів, може бути пам'ять системи **ЗТАПАГ^.**

9.3.4. Пам'ять з послідовним асоціативним доступом

Третій тип організації пам'яті з асоціативним доступом - пам'ять з послідовним асоціативним доступом. Тут логічні операції порівняння виконуються лише над одним словом. Доступ до даних при цьому здійснюється послідовним застосуванням логічних операцій до окремих слів, які зберігаються в пам'яті. Загальна структура такої пам'яті зображена на рис. 9.19. Оскільки при послідовній обробці слів такі властивості асоціативної пам'яті як одночасність операцій та майже не залежна від числа регістрів пам'яті тривалість пошуку відсутні, така пам'ять не в повній мірі відповідає поняттю пам'ятті з асоціативним доступом.

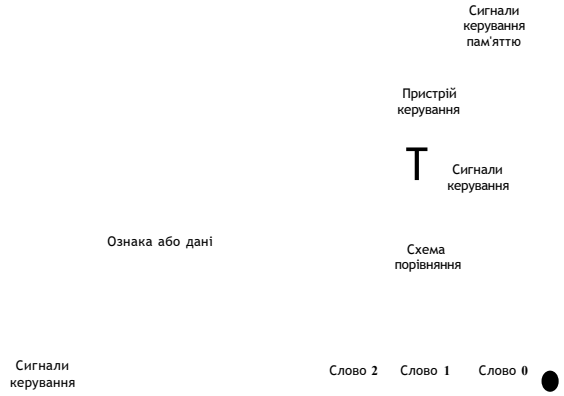


Рис. 9.19. Загальна структура пам'яті з послідовним асоціативним доступом

9.3.5. Пам'ять з частково асоціативним доступом

Четвертий тип організації пам'яті з асоціативним доступом - пам'ять із частково асоціативним доступом. Всі слова цієї пам'яті формуються в п груп. Кожна група слів обробляється паралельно з усіма іншими групами елементів шляхом послідовного переходу від одного слова групи до іншого. Отже, потрібні засоби, які дозволяють послідовно вибирати слова із груп для застосування до них заданих логічних операцій. Оскільки слова в групах вибираються послідовно, то можна обмежитись використанням пристроїв пам'яті з послідовним доступом взамін більш дорогих пристроїв пам'яті з асоціативним доступом. Загальна структура пам'яті з частково асоціативним доступом подана на рис. 9.20.

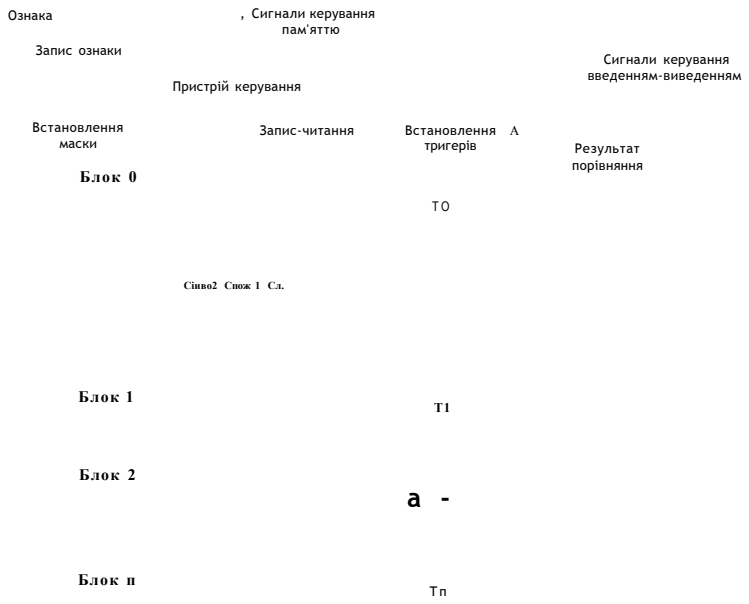


Рис. 9.20. Загальна структура пам'яті з частково асоціативним доступом

9.4. Основна пам'ять

9.4.1. Структура основної пам'яті

До комірок основної пам'яті (ОП) центральний процесор може звертатися безпосередньо, як і до регістрів регістрового файла. Інформація, що зберігається у зовнішній пам'яті, стає доступною процесору тільки після того, як буде переписана в основну пам'ять. Основну пам'ять будують як пам'ять з довільним доступом. Така пам'ять представляє собою масив комірок, а "довільний доступ" означає, що звернення до будь-якої комірки займає один і той же час і може проводитися у довільній послідовності. Кожна комірка містить фіксоване число запам'ятовуючих елементів і має унікальну адресу. Це дозволяє розрізняти комірки при зверненні до них для виконання операцій запису і зчитування (рис. 9.21).

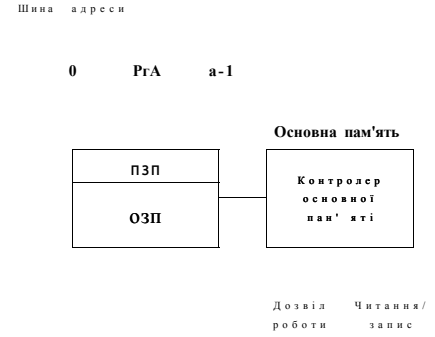


Рис. 9.21. Структура основної пам'яті

Раніше ОП комп'ютерів будувалась на феромагнітних кільцях. В сучасних комп'ютерах основна пам'ять будується на напівпровідникових мікросхемах.

У напівпровідникових мікросхемах застосовуються елементи пам'яті на основі: біполярних транзисторів; які мають структуру "метал-окисел-напівпровідник" (МОП); "метал-нітрид-окисел-напівпровідник" (МНОП); а також на основі пристроїв із зарядовим зв'язком (ПЗС); транзисторів МОП з ізольованим затвором і ін.

Основна пам'ять може включати два типи пристроїв: оперативні запам'ятовуючі пристрої (ОЗП) і постійні запам'ятовуючі пристрої (ПЗП).

Основну частину основної пам'яті утворює ОЗП, який називають оперативним, тому що він допускає як запис, так і зчитування інформації, причому обидві операції виконуються подібним чином та практично за той же час. У англомовній літературі ОЗП відповідає абревіатура R A M - Random Access Memory, тобто "пам'ять з довільним доступом", що не зовсім коректно, оскільки пам'яттю з довільним доступом є також ПЗП і регістровий файл процесора. Для більшості типів напівпровідникових ОЗП характерна енергозалежність - навіть при короткочасному перериванні живлення інформація, що зберігається в цій пам'яті, втрачається. Тому мікросхема ОЗП повинна бути постійно підключеною до джерела живлення, а ця пам'ять може використовуватися тільки як тимчасова пам'ять.

Другу групу напівпровідникових пристроїв основної пам'яті утворюють мікросхеми ПЗП (ROM — Read-Only Memory). ПЗП забезпечує зчитування інформації, але не допускає її зміни (у ряді випадків інформація в ПЗП може бути змінена, але цей процес суттєво відрізняється від зчитування і вимагає значно більшого часу).

9.4.2. *Нарощування розрядності основної пам'яті*

Розглянемо як здійснюється нарощування розрядності основної пам'яті при її побудові на основі мікросхем. Коли розрядність комірок у мікросхемі є меншою від розрядності слів комп'ютера, виникає необхідність об'єднання декількох мікросхем пам'яті.

Нарощування розрядності основної пам'яті реалізується шляхом об'єднання адресних входів окремих мікросхем пам'яті. Якщо об'єднати інформаційні входи і виходи п мікросхем ($МС_0$, МС^..., МС), кожна з яких має 2^r однорозрядних комірок, з входами і виходами пам'яті відповідно до рис. 9.22, то отримається пам'ять збільшеної розрядності. Одержану сукупність мікросхем називають модулем пам'яті. Модулем можна вважати і одну мікросхему, якщо вона вже має потрібну розрядність.

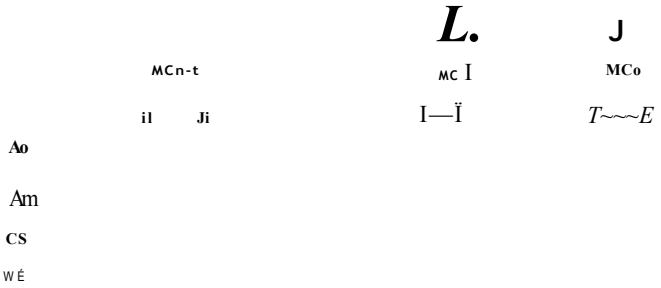


Рис. 9.22. Структура модуля пам'яті з п мікросхем

Тут позначення сигналів CS та WE означає відповідно вибірку кристала (Cristal Select) та дозвіл запису (Write Enable).

9.4.3. *Нарощування ємності основної пам'яті*

Ємність основної пам'яті сучасних комп'ютерів є настільки великою, що її не можна реалізувати на базі однієї мікросхеми. Для отримання необхідної ємності потрібно певним чином об'єднати декілька модулів пам'яті меншої ємності. У загальному випадку основна пам'ять комп'ютера практично завжди має блокову структуру, тобто містить декілька модулів.

При використанні блокової структури пам'яті, що складається з S модулів, адреса комірки перетвориться в пару (р, г), де р - номер модуля, г - адреса комірки всередині модуля. Відомі три схеми розподілу розрядів адреси А між р та г.

В блоковій схемі розподілу розрядів адреси (рис. 9.23) номер модуля р визначають старші k розрядів адреси, які поступають на входи вибору модуля CS (Chip Select).

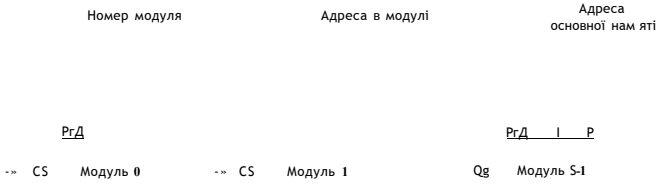


Рис. 9.23. Блокова структура пам'яті з блоковою схемою розподілу розрядів адреси

Адресний простір пам'яті розбитий на групи послідовних адрес, і кожна така група забезпечується окремим модулем пам'яті. Для звернення до основної пам'яті використовується п-розрядна адреса (п = т + к), т молодших розрядів якої ($A_{т-1}$ - A_0) поступають паралельно до регістра адреси РгА всіх модулів пам'яті (МП) і вибирають в кожному з них одну комірку. Старші к розрядів адреси ($A_{п-1}$ - $A_т$) **містять** номер модуля. У функціональному відношенні така пам'ять може розглядатися як єдина, ємність якої рівна сумарній ємності складових, а швидкодія - швидкодії окремого модуля.

В циклічній схемі розподілу розрядів адреси (рис. 9.24) номер р (р = 0,1,..., Б-І) модуля визначають к молодших розрядів адреси.

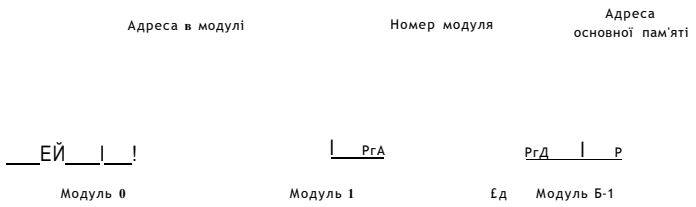


Рис. 9.24. Блокова структура пам'яті з циклічною схемою розподілу розрядів адреси

Адресний простір цієї пам'яті, як і адресний простір пам'яті з блоковою схемою розподілу розрядів адреси, розбитий на групи послідовних адрес, і кожна така група забезпечується окремим модулем пам'яті. Для звернення до основної пам'яті використовується п-розрядна адреса (п = т + к), т старших розрядів якої ($A_{п-1}$ - $A_к$) поступають паралельно до регістра адреси РгА всіх модулів пам'яті (МП) і вибирають в кожному з них одну комірку. Молодші к розрядів адреси ($A^{к-1}$ - A_0) містять номер модуля. У функціональному відношенні така пам'ять може розглядатися як єдина, ємність якої рівна сумарній ємності складових, а швидкодія - швидкодії окремого модуля.

Блоково-циклічна схема розподілу розрядів адреси є комбінацією двох попередніх схем.

9.4.4. *Розшарування пам'яті*

Крім придатності до нарощування ємності, блокова побудова пам'яті має іще одну перевагу - дозволяє скоротити час доступу до інформації. Це можливо завдяки потенційному паралелізму, властивому блоковій організації, який дозволяє досягти меншого часу доступу до інформації за рахунок паралельної роботи багатьох модулів пам'яті. Одну з використовуваних для цього методик називають розшаруванням пам'яті. У її основі лежить так зване чергування адрес, що полягає в зміні системи розподілу адрес між модулями пам'яті.

Прийом чергування адрес базується на такій властивості програм, як локальність за зверненням, згідно з якою послідовний доступ до пам'яті зазвичай проводиться до комірок, що мають суміжні адреси. Іншими словами, якщо в даний момент виконується звернення до комірки з адресою 7, то наступне звернення, найімовірніше, буде здійснюватись до комірки з адресою 8, пізніше 9 і т. д. Структура пристрою пам'яті, в якому використано чергування адрес з метою прискорення доступу до даних, наведена на рис. 9.25.

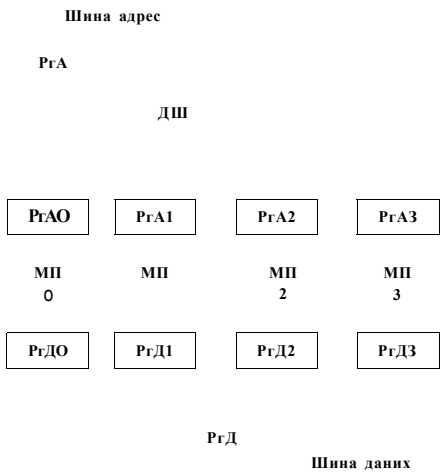


Рис. 9.25. Структура пристрою з розшаруванням пам'яті

Тут два розряди адреси з регістра адреси РгА поступають на дешифратор ДШ, виходи якого вказують, до якого з чотирьох регістрів адрес РгАО-РгАЗ модулів пам'яті МПО-МПЗ мають бути записані всі інші розряди з регістра адреси РгА. Принцип роботи пристрою з розшаруванням пам'яті пояснює рис. 9.26. Як видно з рисунку, до кожного з чотирьох модулів послідовні дані записуються по черзі.

Адреса слона	Адрес, слова	
0	2	3
4	6	7
8	10	11
12 13	14	15

Рис. 9.26. Принцип роботи пристрою з розшаруванням пам'яті

Тим самим забезпечується паралельна робота модулів пам'яті МПО-МПЗ, що дозволяє в Б разів, де Б - кількість модулів пам'яті (в чотири рази для прикладу, приведеного на рис. 9.26) прискорити роботу пам'яті.

Традиційні способи розшарування пам'яті добре працюють у рамках одного завдання, для якого характерна властивість локальності.

У багатопроцесорних системах із загальною пам'яттю, де запити на доступ до пам'яті достатньо незалежні, не виключений інший підхід, який можна розглядати як розвиток ідеї розшарування пам'яті. Для цього в систему включають декілька контролерів пам'яті, що дозволяє окремим модулям працювати автономно. Ефективність даного прийому залежить від частоти незалежних звернень до різних модулів. Кращого результату можна чекати при великому числі модулів, що зменшує вірогідність послідовних звернень до одного і того ж модуля пам'яті.

9.5. *Оперативний запам'ятовуючий пристрій*

Основна пам'ять будується на основі інтегральних мікросхем. Мікросхеми пам'яті організовані у вигляді матриці комірок, кожна з яких має п запам'ятовуючих елементів, де п - розрядність комірки, і має свою адресу. Кожен запам'ятовуючий елемент здатний

зберігати один біт інформації, оскільки він має два стабільні стани, які представляють двійкові значення О і 1. При запису інформації запам'ятовуючий елемент встановлюється в один із двох можливих станів. Для визначення поточного стану запам'ятовуючого елемента його вміст має бути зчитаний.

В мікросхемах пам'яті реалізується координатний принцип адресації комірок, згідно з яким комірка із заданим номером лежить на перетині відповідних вертикальної та горизонтальної ліній. Запам'ятовуючі елементи, об'єднані загальним горизонтальним провідником, прийнято називати рядком. Запам'ятовуючі елементи, підключені до загального вертикального провідника, називають стовпцем. Кожній горизонтальній лінії відповідає один з кодів адреси рядка, а кожній вертикальній лінії відповідає один з кодів адреси стовпця. На рис. 9.27 приведено приклад матриці, яка складається з 64-х комірок пам'яті з координатним принципом адресації. Три молодших розряди адреси (A_2, A_1, A_0) вказують адресу рядка, а три старших розряди адреси (A_5, A_4, A_3) вказують адресу стовпця. Так, комірка 27 лежить на перетині горизонтальної лінії з кодом 011 та вертикальної лінії з кодом 011.

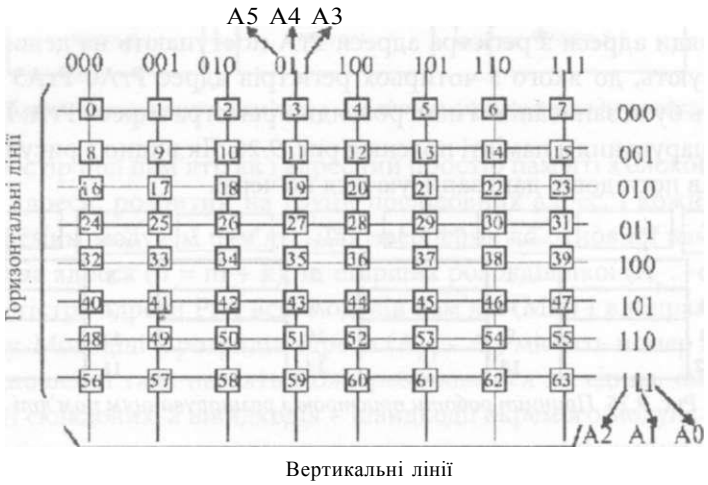


Рис. 9.27. Матриця комірок пам'яті з координатним принципом адресації

Адреса комірки, що поступає по шині адреси в мікросхему пам'яті, пропускається через логіку вибору, де вона розділяється на дві складові: адресу рядка і адресу стовпця. Адреси рядка і стовпця запам'ятовуються відповідно в регістрі адреси рядка і регістрі адреси стовпця мікросхеми (рис. 9.28). Для зменшення числа контактів мікросхеми адреси рядка і стовпця в більшості мікросхем подаються в мікросхему через одні і ті ж контакти послідовно в часі (мультиплексуються). Кожен регістр з'єднаний зі своїм дешифратором. Виходи дешифраторів утворюють систему горизонтальних і вертикальних провідників, до яких підключені матриці комірок пам'яті, при цьому кожна комірка пам'яті розташована на перетині одного горизонтального й одного вертикального провідників.

Крім адресних вертикальних провідників у мікросхемі повинна бути така ж кількість інформаційних провідників, по яких передаватиметься інформація, яка зчитується та записується до пам'яті. Сукупність запам'ятовуючих елементів і логічних схем, пов'язаних із вибором рядків і стовпців, називають ядром мікросхеми пам'яті (рис. 9.28).

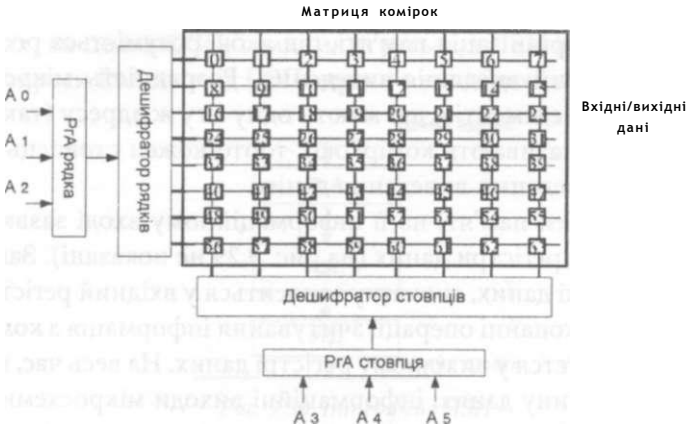


Рис. 9.28. Ядро мікросхеми пам'яті

Крім ядра, в мікросхемі є ще інтерфейсна логіка, що забезпечує взаємодію ядра із зовнішнім світом. У її завдання, зокрема, входить проведення комутації потрібного стовпця на вихід при читанні і на вхід при записі (рис. 9.29), яка здійснюється через вихідні ключі, що керуються логічними схемами запису і зчитування. При цьому логічні схеми запису і зчитування (логіка запису та логіка зчитування), а також логіка керування, яка задає режими роботи пам'яті, працюють на основі аналізу зовнішніх сигналів керування пам'яттю /RAS, /CE, /CS, /WE, /CAS.

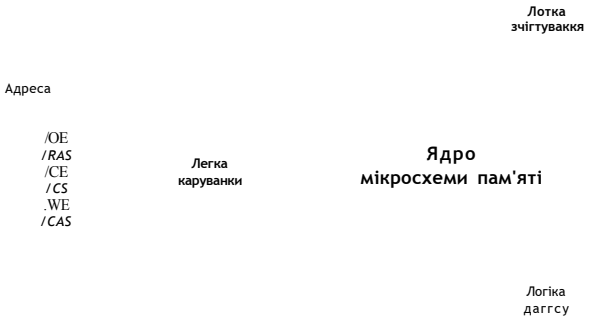


Рис. 9.29. Інтерфейсна логіка мікросхеми пам'яті

Для синхронізації процесів фіксації й обробки адресної інформації всередині мікросхеми адреса рядка (RA) супроводжується сигналом RAS (Row Address Strobe - строб рядка), а адреса стовпця (CA) - сигналом CAS (Column Address Strobe - строб стовпця). Щоб стробування було надійним, ці сигнали подаються із затримкою, достатньою для завершення перехідних процесів на шині адреси та в адресних лініях мікросхеми.

Сигнал вибору мікросхеми CS (Chip Select) дозволяє роботу мікросхеми і використовується для вибору певної мікросхеми в системах пам'яті, що складаються з декількох мікросхем.

Сигнал WE (Write Enable - дозвіл запису) визначає вид виконуваної операції (зчитування або запис).

На фізичну організацію ядра, як матрицю однорозрядних запам'ятовуючих елементів, накладається логічна організація пам'яті, під якою розуміється розрядність мікросхеми, тобто кількість ліній введення-виведення. Розрядність мікросхеми визначає кількість запам'ятовуючих елементів, що мають одну і ту ж адресу (таку сукупність запам'ятовуючих елементів називають коміркою), тобто кожен стовпець містить стільки розрядів, скільки є ліній введення-виведення даних.

Для прискорення роботи пам'яті на її інформаційному вході зазвичай встановлюються вхідний та вихідний регістри даних (на рис. 9.29 не показані). Записувана інформація, що поступає по шині даних, спочатку заноситься у вхідний регістр даних, а потім у вибрану комірку. При виконанні операції зчитування інформація з комірки до її видачі на шину даних буферизируется у вихідному регістрі даних. На весь час, поки мікросхема пам'яті не використовує шину даних, інформаційні виходи мікросхеми переводяться в третій (високоімпедансний) стан. Керування перемиканням в третій стан забезпечується сигналом OE (Output Enable - дозвіл видачі вихідних сигналів). Цей сигнал активізується при виконанні операції зчитування.

Для більшості перерахованих вище сигналів керування активним зазвичай вважається їх низький рівень, що і показано на рис. 9.29.

Керування операціями з основною пам'яттю здійснюється контролером пам'яті (рис. 9.21). Зазвичай цей контролер входить до складу центрального процесора або реалізується у вигляді зовнішнього по відношенню до пам'яті пристрою. В останніх типах мікросхем пам'яті частина функцій контролера покладається на мікросхему пам'яті. Хоча робота мікросхеми пам'яті може бути організована як по синхронній, так і по асинхронній схемі, контролер пам'яті є синхронним пристроєм, тобто він спрацьовує виключно по тактових імпульсах. З цієї причини операції з пам'яттю прийнято описувати з прив'язкою до тактів. У загальному випадку на кожну таку операцію потрібно як мінімум п'ять тактів, які використовуються у наступній послідовності:

- Вказівка типу операції (зчитування або запис) і встановлення адреси рядка.
- Формування сигналу RAS.
- Встановлення адреси стовпця.
- Формування сигналу CAS.
- Повернення сигналів RAS і CAS в неактивний стан.

Даний перелік враховує далеко не всі необхідні дії, наприклад, регенерацію вмісту пам'яті в динамічних ОЗП.

9.6. *Постійний запам'ятовуючий пристрій*

9.6.7. *Організація роботи постійного запам'ятовуючого пристрою*

Постійний запам'ятовуючий пристрій (ПЗП), як і оперативний, є складовою частиною основної пам'яті, та часто використовується при побудові інших вузлів комп'ютера, наприклад, табличних операційних пристроїв, мікропрограмних пристроїв керування і т. д. Слово "постійний" вказує на те, що в цьому пристрої один раз записана інформація не міняється взагалі, або це здійснюється при переведенні його в спеціальний режим роботи. При цьому в ньому інформація зберігається і при відсутності напруги живлення.

Як видно з рис. 9.30, де показано інтерфейс ПЗП, в ньому відсутній сигнал запису-зчитування даних, як це є в ОЗП, і дане зчитується з ПЗП в регістр даних РгД при поданні з регістра адреси РгА на вхід ПЗП адреси відповідної комірки пам'яті.

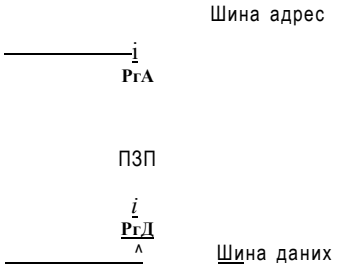


Рис. 9.30. Інтерфейс ПЗП

Для підвищення швидкодії і збільшення об'єму ПЗП використовуються ті ж підходи, що і для ОЗП. Мікросхеми ПЗП, як і мікросхеми ОЗП, побудовані за принципом матричної структури накопичувача, де у вузлах розміщені запам'ятовуючі елементи, підключені до адресних та інформаційних ліній В якості запам'ятовуючих елементів тут можуть бути використані провідники, діоди або транзистори, які можуть бути замкнуті (відповідає значенню 1), або розімкнуті (відповідає значенню 0).

Основним режимом роботи ПЗП є зчитування інформації, яке мало відрізняється від аналогічної операції в ОЗП як за організацією, так і за тривалістю. Саме ця обставина підкреслює англомовну назву ПЗП - ROM (Read-Only Memory - пам ять тільки для зчитування). В той же час запис в ПЗП у порівнянні зі зчитуванням зазвичай є складнішим і потребує великих витрат часу і енергії. Занесення інформації в ПЗП називають програмуванням або "прошивкою". Сучасні ПЗП реалізуються у вигляді напівпровідникових мікросхем, які за можливостями і способами програмування розділяють на:

- запрограмовані при виготовленні;
- одноразово програмовні після виготовлення;
- багаторазово програмовні.

9.6.2. *Запрограмований при виготовленні постійний запам'ятовуючий пристрій*

Групу ПЗП, що програмуються при виготовленні, утворюють так звані масочні пристрої і саме до них прийнято застосовувати абревіатуру ПЗП. У літературі більш поширене позначення різних варіантів ПЗП скороченнями, ніж англійські назви; тому надалі також використовуватимемо аналогічну сп-стему. Для маскованих ПЗП таким позначенням є ROM, співпадаючий із загальною назвою всіх типів ПЗП. Іноді такі мікросхеми іменують MROM (Mask Programmable ROM - ПЗП, що програмуються за допомогою маски).

Занесення інформації в масковані ПЗП складає частину виробничого процесу і полягає у підключенні або не підключенні запам'ятовуючого елемента до розрядної лінії зчитування. Залежно від цього із запам'ятовуючого елемента завжди зчитуватиметься 1 або 0. В ролі перемички виступає транзистор, розташований на перетині адресної і розрядної ліній. Які саме запам ятовуючі елементи повинні бути підключені до вихідної лінії визначає маска, що дозволяє вибрати визначені ділянки кристала. При створенні маскованих ПЗП застосовуються різні технології. У першому випадку маска просто не

допускає металізації ділянки, яка з єднує транзистор з розрядною лінією зчитування. Друга технологія пов'язана з типом транзистора у вузлі. Маска визначає, який польовий транзистор має бути імплантований в даний вузол, що працює в збагаченому режимі або в режимі збіднення. У третьому варіанті маска задає товщину оксидного шару транзистора. Залежно від цього на кристалі формується або стандартний транзистор, або транзистор з високим порогом спрацьовування.

У початковий період масковані мікросхеми були дорогі, проте зараз це один з найбільш дешевих видів ПЗП. Для ПЗП характерна висока щільність пакування запам'ятовуючих елементів на кристалі і високі швидкості зчитування інформації. Основною сферою застосування є пристрої, що вимагають зберігання фіксованої інформації. Так, подібні ПЗП часто використовують для зберігання мікропрограм у мікропроцесорах, констант у табличних операційних пристроях, шрифтів у лазерних принтерах.

9.6.3. Одноразово запрограмований після виготовлення постійний запам'ятовуючий пристрій

Створення масок для ПЗП виправдане при виробництві великого числа копій. Якщо потрібна відносно невелика кількість мікросхем із даною інформацією, розумною альтернативою є одноразово програмовні ПЗП. Структура матриці вентилів програмованого ПЗП показана на рис. 9.31.

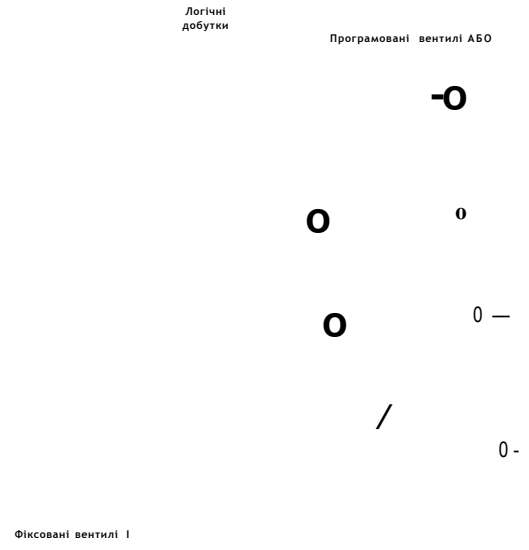


Рис. 9.31. Структура програмованого ПЗП

Одноразово програмований ПЗП складається з двох матриць І та АБО. Потрібно зауважити, що матриця І представляє собою декодер, який генерує всі логічні добутки п змінних. Зафарбовані чорним круги зображають фіксовані електричні з'єднання між входами ПЗП та комірками І. З цієї причини ця матриця називається фіксованою матрицею І.

Кожна функція формується однією коміркою АБО, яка має 2п входів. З'єднання між виходами результуючих добутків і входами комірок АБО задаються користувачем шляхом програмування комірок, які відображають з'єднання добутку та комірки АБО. В противагу фіксованого масиву вентилів І масив вентилів АБО є програмовним. На рис. 9.31 показано збільшене зображення програмовної комірки. Ця комірка включає ключ, який може бути відкритим або закритим. Закритий ключ означає включення добутку до результату, формуючи функцію 1 для вхідної комбінації. Коли ж ключ відкритий, добуток не включається і для цього входу функція рівна 0.

Оскільки ПЗП генерує всі можливі логічні добутки, будь-яка функція п змінних може бути реалізована.

Одноразово програмовних ПЗП є кілька типів

Мікросхеми типу PROM (Programmable ROM - програмовані ПЗП) інформація може бути записана тільки одноразово. Першими такими ПЗП стали мікросхеми пам яті на базі плавких запобіжників. У початковій мікросхемі у всіх вузлах адресні лінії з'єднані з розрядними. Занесення інформації в PROM проводиться електричними сигналами шляхом перепалювання окремих плавких перемичок, і може бути виконано постачальником або споживачем через деякий час після виготовлення мікросхеми. Подібні ПЗП випускалися в рамках серій К556 і К1556. Пізніше з явилися мікросхеми, де в перемичку входили два діоди, включені назустріч один одному. В процесі програмування можна було видалити перемичку за допомогою електричного пробою одного з цих діодів. У будь-якому варіанті для запису інформації потрібне спеціальне устаткування - програматори. Основними недоліками даного виду ПЗП були великий відсоток браку і необхідність спеціального термічного тренування після програмування, без якого надійність зберігання даних була невисокою

Ще один вид одноразово програмованого ПЗП це OTP EPROM (One Time Programmable EPROM EPROM з одноразовим програмуванням). У його основі лежить кристал EEPROM (див. нижче), але поміщений в дешевий непрозорий пластиковий корпус без кварцевого вікна, із за чого він може бути запрограмований лише один раз

9.6.4. *Багаторазово програмований постійний запам'ятовуючий пристрій*

В багаторазово програмованих ПЗП в програмованих комірках використовуються МОП транзистори, параметри яких змінюються під впливом високої напруги. Вміст ПЗП може бути стертий застосуванням ще одної високої напруги або застосуванням ультрафіолетового світла

Процедура програмування таких ПЗП складається з двох етапів. Спочатку проводиться стирання вмісту всіх або частини комірок, а потім проводиться запис нової інформації. У цьому класі ПЗП виділяють декілька груп

- EPROM (Erasable Programmable ROM - програмовані ПЗП, вміст яких може бути стертий);
- EEPROM (Electrically Erasable Programmable ROM програмовані ПЗП, вміст яких може бути стертий електрично);
- флеш пам'ять

У EPROM запис інформації проводиться електричними сигналами, так само як в PROM, проте перед операцією запису вміст всіх комірок повинен бути приведений до однакового стану (стерто) шляхом дії на мікросхему ультрафіолетовим опромінюванням. Кристал по

мішається в керамічний корпус, що має невелике кварцове вікно, через яке і проводиться опромінювання. Щоб запобігти випадковому стиранню інформації, після опромінювання кварцове вікно заклеюють непрозорою плівкою. Процес стирання може виконуватися багато разів. Кожне стирання займає близько 20 хвилин. Цикл програмування займає декілька сотень мілісекунд. Час зчитування є близьким до показників ROM і DRAM.

У порівнянні з PROM мікросхеми EPROM є дорожчими, але завдяки можливості багатократного перепрограмування вони часто застосовуються. Даний вид мікросхем випускався в рамках серії К573.

Привабливішим варіантом багатократно програмованої пам'яті є програмована постійна пам'ять EEPROM, вміст якої може бути стертий електрично. Стирання і запис інформації в цю пам'ять проводяться побайтово, причому стирання - не окремий процес, а лише етап, що відбувається автоматично при записі. Операція запису займає істотно більше часу ніж зчитування - декілька сотень мікросекунд на байт. У мікросхемі використовується той же принцип зберігання інформації, що і в EPROM. Програмування EPROM не вимагає спеціального програматора і реалізується засобами самої мікросхеми. Випускаються два варіанти мікросхем EEPROM: з послідовним і паралельним доступом, причому перших значно більше. У EEPROM з доступом по послідовному каналу (SEEPROM - Serial EEPROM) адреси, дані і команди передаються по одному провіднику і синхронізуються імпульсами на тактовому вході. Перевагою SEEPROM є малі габарити і мінімальне число ліній введення-виведення, а недоліком - великий час доступу. SEEPROM випускаються в рамках серій мікросхем 24Cxxx, 25Cxxx і 33Cxxx, а паралельні EEPROM - в складі серії 28Cxxx.

В цілому EEPROM є дорожчими, ніж EPROM, а мікросхеми мають менш щільне пакування комірок, тобто меншу ємність.

Відносно новий вид напівпровідникової пам'яті - це флеш-пам'ять (назву flash можна перевести як "спалах блискавки", що підкреслює відносно високу швидкість перепрограмування). Вперше анонсована у середині 80-х років, флеш-пам'ять багато в чому схожа на EEPROM, але використовує особливу технологію побудови запам'ятовуючих елементів. Аналогічно EEPROM, у флеш-пам'яті стирання інформації проводиться електричними сигналами, але не побайтово, а по блоках або повністю. Тут слід зазначити, що існують мікросхеми флеш-пам'яті з розбиттям на дуже дрібні блоки (сторінки) з автоматичним посторінковим стиранням, що зближує їх за можливостями з EEPROM. Як і у випадку з EEPROM, мікросхеми флеш-пам'яті випускаються у варіантах з послідовним і паралельним доступом.

За організацією масиву запам'ятовуючих елементів розрізняють флеш-пам'ять з тотальним очищенням, коли стирання допустимо тільки для всього масиву запам'ятовуючих елементів, та з блоковим очищенням, коли масив запам'ятовуючих елементів роздільний на декілька блоків різного або однакового розміру, вміст яких може очищатися незалежно.

Вміст флеш-пам'яті може бути очищений за декілька секунд, запис одного слова займає декілька мікросекунд, а час доступу до даних при зчитуванні - декілька десятків наносекунд.

9.7. Зовнішня пам'ять

9.7.1. Магнітні диски

Магнітні диски - це набір металевих або пластмасових пластин, покритих магнітним матеріалом, на якому зберігається інформація. Запис та зчитування інформації проводиться за допомогою головки запису-зчитування, яка є електромагнітною котушкою, причому в процесі запису-зчитування диск обертається відносно нерухомої головки.

При записі на головку подаються електричні імпульси, що намагнічують ділянку поверхні магнітного матеріалу під нею, причому характер намагніченості поверхні різний залежно від напряму струму в котушці. Зчитування базується на електричному струмі, що наводиться в котушці головки, коли під нею проходить ділянка магнітного матеріалу поверхні диска, причому в котушці наводиться струм тієї ж полярності, що використовувався для запису інформації. Не дивлячись на різноманітність типів магнітних дисків, принципи їх організації зазвичай однотипні.

Дані на диску організовані у вигляді набору концентричних кіл, названих доріжками (рис. 9.26). Кожна з них має ту ж ширину, що і головка. Сусідні доріжки розділені проміжками. Це запобігає помилкам із-за зсуву головки або із-за інтерференції магнітних полів. Як правило, для спрощення електроніки приймається, що на всіх доріжках може зберігатися однакова кількість інформації. Таким чином, щільність запису збільшується від зовнішніх доріжок до внутрішніх.

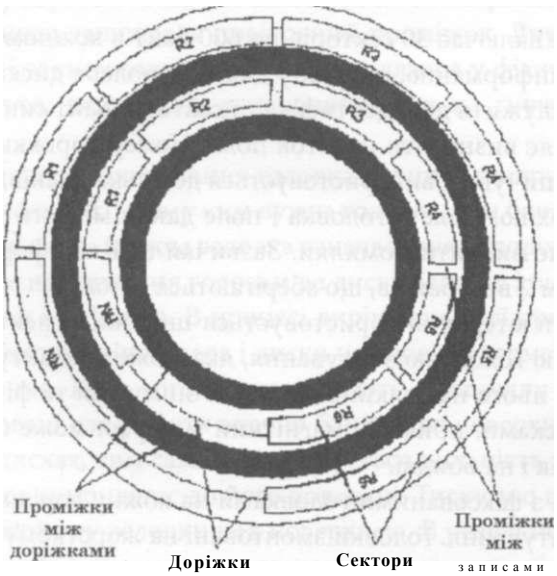


Рис. 9.32. Розміщення інформації на магнітному диску

Обмін інформацією між ОП і магнітним диском здійснюється блоками. Розмір блоку зазвичай є меншим ємності доріжки, і дані на доріжці зберігаються у вигляді послідовних областей - секторів, розділених між собою проміжками. Розмір сектора рівний мінімальному розміру блоку.

Типове число секторів на доріжці коливається від 10 до 100. При такій організації повинні бути задані точка відліку секторів і спосіб визначення початку і кінця кожного сектора. Все це забезпечується за допомогою форматування, в ході якого на диск заноситься службова інформація, недоступна користувачу і використовувана тільки апаратурою дискової пам'яті.

Приклад розмітки магнітного диску показаний на рис. 9.33.

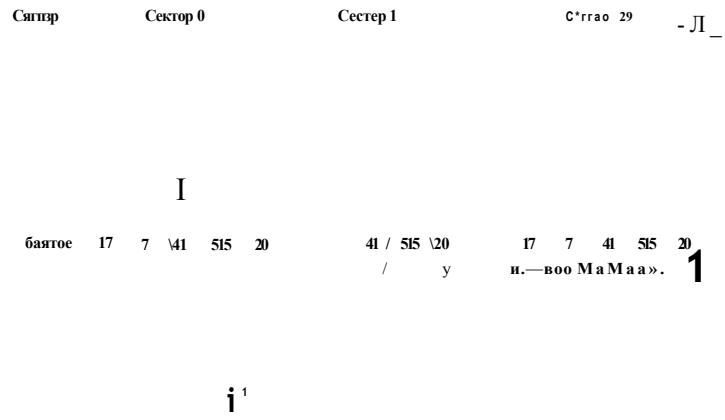


Рис. 9.33. Приклад розмітки магнітного диску типу "Вінчестер"

Тут кожна доріжка включає 30 секторів по 600 байт в кожному. Сектор зберігає 512 байт даних і керуючу інформацію, потрібну для контролера диска. Поле заголовка містить інформацію, що служить для ідентифікації сектора. Байт синхронізації є характерним кодом, що дозволяє визначити початок поля. Номер доріжки визначає доріжку на поверхні. Якщо в накопичувачі використовується декілька дисків, то номер головки визначає потрібну поверхню. Поле заголовка і поле даних містять також код циклічного контролю, що дозволяє виявити помилки. Зазвичай цей код формується послідовним додаванням за модулем 2 всіх байтів, що зберігаються в полі.

В даний час у комп'ютерах використовується широкий спектр дискових систем: з фіксованою та рухомою головкою зчитування, яка може контактувати з магнітним матеріалом, або бути від нього на деякій відстані, із знімними та фіксованими дисками, з одним та багатьма дисками, причому магнітний матеріал може бути нанесеним як на одну сторону диска, так і на обидві.

У дисковій системі з фіксованими головками на кожну доріжку приходиться по одній головці запису-зчитування. Головки змонтовані на жорсткому важелі, що перетинає всі доріжки диска. У дисковій системі з рухомими головками є тільки одна головка, також встановлена на важелі, проте важіль здатний переміщатися в радіальному напрямі над поверхнею диска, забезпечуючи можливість позиціювання головки на будь-яку доріжку.

Диски з магнітним носієм встановлюються в дискову систему, що складається з важеля, шпинделя, що обертає диск, і електронних схем, потрібних для введення і виведення двійкових даних. Незнімний диск зафіксований в дисковій системі. Знімний диск може бути вийнятий і замінений на інший аналогічний диск. Перевага системи із знімними

дисками - можливість зберігання необмеженого об'єму даних при обмеженому числі дискових пристроїв. Крім того, такий диск може бути перенесений з одного комп'ютера на інший.

Більшість дисків мають магнітне покриття з обох сторін.

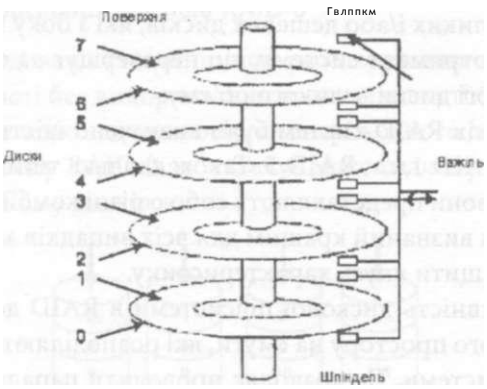


Рис. 9.34. Дискова пам'ять з дисковим пакетом

На осі може розташовуватися один, або декілька (рис. 9.34) дисків. У останньому випадку використовують термін дисковий пакет.

Залежно від вживаної головки запису-зчитування можна виділити три типи дискової пам'яті. У першому варіанті головка встановлюється на фіксованій дистанції від поверхні так, щоб між ними залишався повітряний проміжок. Другий варіант - це коли в процесі зчитування і запису головка і диск знаходяться у фізичному контакті. Такий спосіб використовується, наприклад, у накопичувачах на гнучких магнітних дисках (дискетах).

Для правильного запису і зчитування головка повинна генерувати і сприймати магнітне поле певної інтенсивності, тому чим вужча головка, тим ближче повинна вона розміщуватися до поверхні диска (вужча головка означає і вужчі доріжки, а значить, і велику місткість диска). Проте наближення головки до диска означає і більший ризик помилки за рахунок забруднення і дефектів. В процесі вирішення цієї проблеми був створений диск типу вінчестер. Головки вінчестера і диски поміщені в герметичний корпус, що захищає їх від забруднення. Крім того, головки мають дуже малу вагу й аеродинамічну форму. Вони спроектовані для роботи значно ближче до поверхні диска, ніж головки в звичайних жорстких дисках, тим самим підвищується щільність зберігання даних. При зупиненому диску головка прилягає до його поверхні. Тиску, що виникає при обертанні диска, достатньо для підйому головки над поверхнею. В результаті створюється неконтактна система з вузькими головками запису-зчитування, що забезпечує щільніше прилягання головки до поверхні диска, ніж у звичайних жорстких дисках.

9.7.2. *Масиви магнітних дисків з надлишковістю*

Оскільки жорсткі диски є основною зовнішньою пам'яттю комп'ютера, до їх характеристик, а саме вартості, продуктивності та відмовостійкості, висуваються підвищені вимоги. Причому, враховуючи те, що розрив у продуктивності між жорстким диском

і ядром комп'ютера постійно росте, а збільшення його продуктивності вдвічі було досягнуто лиш через 10 років (рис. 9.8), розробники вимушені шукати структурні методи покращення їх характеристик. Один із таких методів, а саме надлишкові масиви дешевих дисків (RAID - Redundant Arrays of Inexpensive Disks) був запропонований у 1987 році Д. Паттерсоном, R Гартом Р. Катцом. В основу RAID покладена наступна ідея: об'єднуючи в масив декілька невеликих і/або дешевих дисків, які з боку комп'ютера сприймаються як один диск, можна отримати систему, що перевершує за об'ємом, продуктивністю роботи і надійністю дорогі диски великого об'єму.

Асоціацією виробників RAID-систем було визначено шість базових типів дискових масивів RAID: RAID 0, RAID 1,..., RAID 5. Також є кілька типів RAID, які не включені в даний перелік, оскільки вони представляють собою різні комбінації базових рівнів. Хоча ні один тип не може бути визнаний кращим для всіх випадків застосування, кожен з них дозволяє відчутно покращити якусь характеристику.

Підвищена продуктивність дискової підсистеми в RAID досягається за допомогою розбиття даних і дискового простору на смуги, які розподіляються по різних дисках, відповідно до визначеної системи. Це дозволяє проводити паралельне зчитування або запис відразу кількох смуг, якщо вони розміщені на різних дисках. В ідеальному випадку, продуктивність дискової підсистеми може бути збільшена в ту кількість разів, яка рівна кількості дисків в масиві. Розмір (ширина) смуги вибирається виходячи з особливостей кожного рівня RAID, і може бути рівна біту, байту, розміру фізичного сектора диска, або ж розміру доріжки. Частіше всього послідовні смуги розділяються по послідовних дисках масиву. Так, в масиві із п дисків п перших смуг фізично розміщені як перші смуги на кожному з дисків, наступні п смуг - як другі смуги на кожному диску і т. д. Набір послідовних смуг, однаково розміщених на кожному диску масиву, називають поясом.

Інша ціль концепції RAID - це підвищення їх відмовостійкості за рахунок виявлення і корекції помилок, які виникають при відмові дисків, або в результаті збоїв. Досягається це за рахунок надлишкового дискового простору, який задіяний для зберігання додаткової інформації, що дозволяє відновити спотворені або втрачені дані. В RAID передбаченні три види додаткової інформації: дублювання, код Хемінга та використання бітів контролю парності.

Дублювання полягає в записі тих же даних на інші диски масиву, що дозволяє при відмові одного з дисків скористатися відповідною інформацією, збереженою на справних дисках. У принципі розділення інформації по дисках масиву може бути довільним, але для скорочення затрат, зв'язаних із пошуком копії, застосовується розбиття масиву на пари дисків, де в кожній парі дисків інформація є ідентичною та розміщена однаково. При такому дублюванні надмірність дискового масиву становить 100%.

Код Хемінга обчислюється для кожної групи смуг, однаково розміщених на всіх дисках масиву, тобто поясу. Коригуючі біти зберігаються на спеціально виділених для цієї цілі додаткових дисках. Так, для масиву з десяти дисків потрібно чотири додаткових диски.

Використання бітів контролю парності замість коду Хемінга передбачає обчислення контрольної смуги бітів парності для кожного набору смуг, розміщених в ідентичній позиції на всіх дисках масиву. В ній значення окремого біта формується як сума за модулем два для однойменних бітів в усіх контрольованих смугах. Для зберігання смуг бітів контролю парності потрібно тільки один додатковий диск. У випадку відмови якогось із

дисків масиву проводиться звернення до диску бітів контролю парності і дані відновлюються по бітах бітів парності та даних від інших дисків масиву.

Розглянемо шість базових типів дискових масивів RAID: RAID 0, RAID 1,..., RAID 5 та їх комбінації детальніше.

9.7.2. J. *Базовий тип дискових масивів RAID 0*

Даний тип RAID побудовано за принципом розшарування пам'яті й орієнтовано на підвищення продуктивності без використання надлишковості (рис. 9.35).

Оскільки RAID 0 не володіє надлишковістю, аварія одного диска приводить до аварії всього масиву. З іншого боку RAID 0 забезпечує максимальну продуктивність й ефективність використання об'єму дисків.

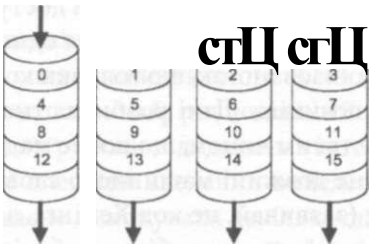


Рис. 9.35. Структура RAID 0

Область застосування: аудіо та відео додатки, що вимагають високої швидкості безперервної передачі даних, яку не може забезпечити один диск. Наприклад, дослідження, проведені фірмою Mylex, з метою визначити оптимальну конфігурацію дискової системи для станції нелінійного відео монтажу показують, що, в порівнянні з одним диском, масив RAID 0 з двох дисків дає приріст швидкості запису-зчитування на 96%, з трьох дисків - на 143% (за даними тесту Miro VIDEO EXPERT Benchmark). Мінімальна кількість дисків в масиві RAID 0 - 2шт.

9.7.2.2. Базовий тип *дискових масивів RAID 1*

В даному типі RAID надлишковість досягається дублюванням даних. Тут кожна пара дисків містить однакову інформацію і сприймається як один логічний диск (рис. 9.36).

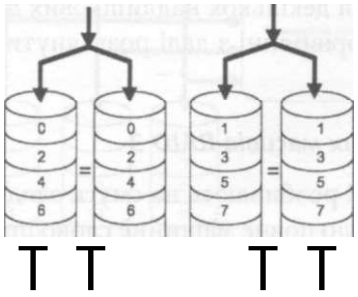


Рис. 9.36. Структура RAID 1

Запис тих же даних проводиться на обидва диски в кожній парі одночасно. Проте, диски, що входять у пару, можуть здійснювати одночасні операції зчитування різних даних. Таким чином, дублювання може подвоювати швидкість зчитування, але швидкість запису залишається незмінною в порівнянні з одним диском. RAID 1 володіє 100% надлишковістю і аварія одного диска не приводить до аварії всього масиву - контролер дисків перемикає операції зчитування/запису на працездатний диск.

RAID 1 забезпечує найвищу продуктивність серед всіх типів надлишкових дискових масивів (RAID 1 - RAID 5), але характеризується найгіршим використанням дискового простору. Мінімальна кількість дисків в масиві RAID 1 - 2шт.

9.7.2.3. Базовий тип дискових масивів RAID 2

У системах RAID 2 використовується паралельний доступ до даних на різних дисках, де у виконанні кожного запиту на введення/виведення одночасно беруть участь всі диски. Шпинделі всіх дисків синхронізовані так, що головки кожного диску в кожен момент часу знаходяться в однакових позиціях. Дані розбивають на смуги завдовжки в 1 біт і розподіляють по дисках масиву таким чином, що повне машинне слово представляється поясом, тобто число дисків рівне довжині машинного слова в бітах. Для кожного слова обчислюється коригуючий код (зазвичай, це код Хемінга, здатний коригувати одиночні 1виявляти подвійні помилки), який, також побітово, зберігається на додаткових дисках (мал. 5.41). Наприклад, для масиву, орієнтованого на зберігання 32-розрядних слів (32 основних диски) потрібно сім додаткових дисків, так як коригуючий код складається з 7 розрядів.

При записі обчислюється коригуючий код, який заноситься на відведені для нього диски. При кожному читанні проводиться доступ до всіх дисків масиву, включаючи додаткові. Зчитані дані разом із коригуючим кодом подаються на контролер дискового масиву, де відбувається повторне обчислення коригуючого коду і його порівняння з тим кодом, що зберігався на надмірних дисках. Якщо присутня одиночна помилка, контролер здатний швидко її розпізнати і виправити, так що час зчитування не збільшується.

RAID 2 дозволяє досягти високої швидкості введення-виведення при роботі з великими послідовними записами, але стає неефективним при обслуговуванні записів невеликої довжини. Основна перевага RAID 2 полягає у високому ступені захисту інформації, проте використаний в цій схемі метод корекції вже існує в диску, оскільки більшість дисків зберігають коди корекції помилок для кожного сектора.

Тому, оскільки використання декількох надлишкових дисків є неефективним, RAID 2 не дає особливих переваг у порівнянні з далі розглянутим RAID 3 і практично не застосовується.

9.7.2.4. *Базовий тип дискових масивів RAID 3*

Як і у випадку з RAID 2, дані розбивають на смуги завдовжки в 1 біт і розподіляють по дисках масиву таким чином, що повне машинне слово представляється поясом, тобто число дисків рівне довжині машинного слова в бітах, а один з дисків масиву відводиться для зберігання інформації про парність (рис. 9.37). Тобто різниця між RAID 3 і RAID 2 - потрібен лише один додатковий диск для зберігання біту парності, а не декілька дисків для зберігання коду Хемінга.

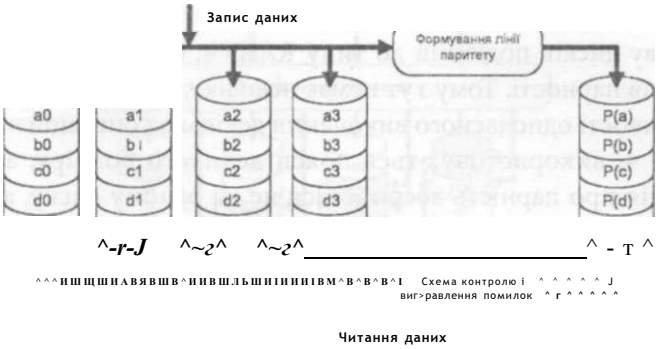


Рис.9.37. Структура RAID 3

У разі відмови одного з дисків, відновлення інформації, що зберігалася на ньому, можливе шляхом виконання операції виключного АБО (XOR) за інформацією на працездатних дисках. Кожен запис, зазвичай, розподілений по всіх дисках, і тому цей тип масиву дисків є ефективним для роботи в додатках з інтенсивним обміном із дисковою підсистемою. Оскільки кожна операція введення-виведення звертається до всіх дисків масиву, RAID 3 не може одночасно виконувати декілька операцій. Тому RAID 3 хороший для однозадачного оточення з довгими записами. Для роботи з короткими записами потрібна синхронізація обертання дисків, оскільки інакше зменшується швидкість обміну. Застосовується рідко, оскільки програє RAID 5 по використанню дискового простору. Мінімальна кількість дисків в масиві RAID 3 - 3шт.

9.7.2.5. Базовий тип дискових масивів RAID 4

RAID 4 ідентичний RAID 3 за винятком того, що тут розмір поясів є набагато більшим одного сектора. В цьому випадку зчитування можна здійснювати з одного диска (не рахуючи диска, що зберігає інформацію про парність), тому можливе одночасне виконання декількох операцій зчитування (рис. 9.38). Проте, оскільки кожна операція запису повинна відновити вміст диска парності, одночасне виконання декількох операцій запису неможливе. Цей тип масиву не має помітних переваг перед масивом типу RAID 5.

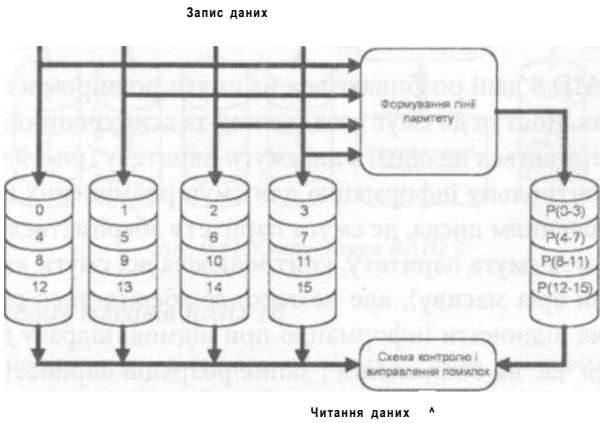


Рис. 9.38. Структура RAID 4

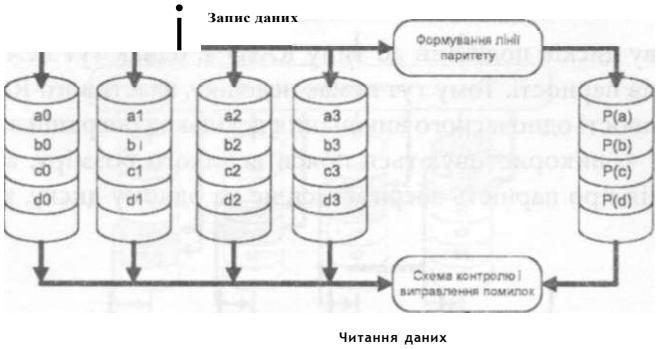


Рис.9.37. Структура RAID 3

У разі відмови одного з дисків, відновлення інформації, що зберігалася на ньому, можливе шляхом виконання операції виключного АБО (XOR) за інформацією на працездатних дисках. Кожен запис, зазвичай, розподілений по всіх дисках, і тому цей тип масиву дисків є ефективним для роботи в додатках з інтенсивним обміном із дисковою підсистемою. Оскільки кожна операція введення-виведення звертається до всіх дисків масиву, RAID 3 не може одночасно виконувати декілька операцій. Тому RAID 3 хороший для однозадачного оточення з довгими записами. Для роботи з короткими записами потрібна синхронізація обертання дисків, оскільки інакше зменшується швидкість обміну. Застосовується рідко, оскільки програє RAID 5 по використанню дискового простору. Мінімальна кількість дисків в масиві RAID 3 - 3шт.

9.7.2.5. *Базовий тип дискових масивів RAID 4*

RAID 4 ідентичний RAID 3 за винятком того, що тут розмір поясів є набагато більшим одного сектора. В цьому випадку зчитування можна здійснювати з одного диска (не рахуючи диска, що зберігає інформацію про парність), тому можливе одночасне виконання декількох операцій зчитування (рис. 9.38). Проте, оскільки кожна операція запису повинна відновити вміст диска парності, одночасне виконання декількох операцій запису неможливе. Цей тип масиву не має помітних переваг перед масивом типу RAID 5.

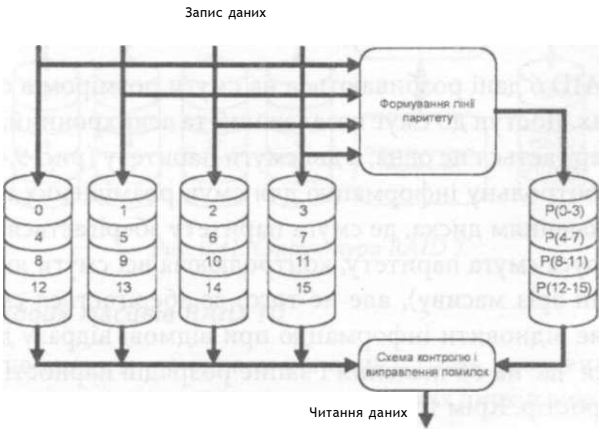


Рис. 9.38. Структура RAID 4

9.7.2.6. *Базовий тип дискових масивів RAID 5*

Цей тип масиву дисків подібний до типу RAID 4, однак тут немає окремого диску для зберігання бітів парності. Тому тут немає недоліку, властивого RAID 4, який заключається у неможливості одночасного виконання декількох операцій запису. У цьому масиві, як і в RAID 4, використовуються пояси великого розміру, але, на відміну від RAID 4, інформація про парність зберігається не на одному диску, а на всіх дисках по черзі (рис. 9.39).

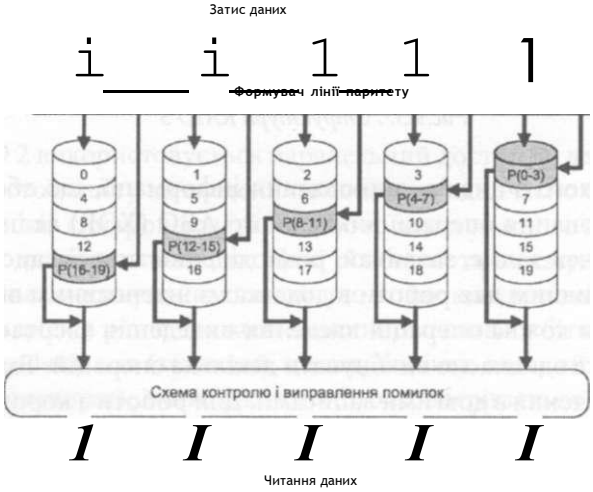


Рис. 9.39. Структура RAID 5

Операції запису звертаються до одного диска з даними і до іншого диска з інформацією про парність. Оскільки біти парності для різних поясів зберігаються на різних дисках, виконання декількох одночасних операцій запису є неможливим, тільки в тих окремих випадках, коли або пояси з даними, або пояси з бітами парності знаходяться на тому ж диску. Чим більше дисків у масиві, тим рідше співпадає місцеположення поясу і біта парності.

Область застосування цього типу дискових масивів - надійне зберігання масивів даних великого об'єму. Мінімальна кількість дисків в масиві RAID 5 - 3шт.

9.7.2.7. *Тип дискових масивів RAID 6*

Як і в RAID 5 в RAID 6 дані розбиваються на смуги розміром в один блок і розподіляються по всіх дисках. Доступ до смуг незалежний та асинхронний. Різниця в тому, що на кожному диску зберігається не одна, а дві смуги паритету (рис. 9.40). Перша з них, як і в RAID 5, містить контрольну інформацію для смуг, розміщених на горизонтальному зрізі масиву (за виключенням диска, де смуга паритету зберігається). В додаток формується і записується друга смуга паритету, контролююча всі смуги якогось одного диску масиву, (вертикальний зріз масиву), але не того, де зберігається смуга паритету. Така схема масиву дозволяє відновити інформацію при відмові відразу двох дисків. З іншої сторони, збільшується час на обчислення і запис розрядів парності та вимагається додатковий дисковий простір. Крім того, реалізація даної схеми пов'язана з ускладненням контролера дискового масиву. Тому, даний тип RAID зустрічається рідко.

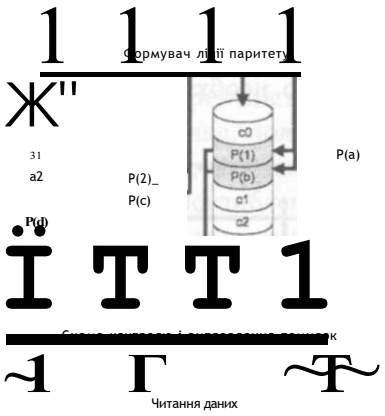


Рис. 9.40. *Структура RAID 6*

9.7.2.8. Тип дискових масивів RAID 7

RAID 7 не входить до складу основних типів дискових масивів. До складу масиву дисків RAID 7, крім самого масиву асинхронно працюючих дисків, входять кешпам'ять та контролер, які керуються вбудованою в контролер операційною системою реального часу (рис. 9.41). Дані розбиті на смуги розміром у блок і розділені по дисках масиву. Смуги бітів парності зберігаються на спеціально виділених для даної цілі одному або кількох дисках.

Даний тип RAID при роботі з великими файлами не поступається за продуктивністю RAID 3. Разом з тим, RAID 7 може так само ефективно, як і RAID 5, проводити кілька одночасних операцій зчитування і запису для невеликих об'ємів даних, що забезпечується використанням кеш пам'яті і названої операційної системи.

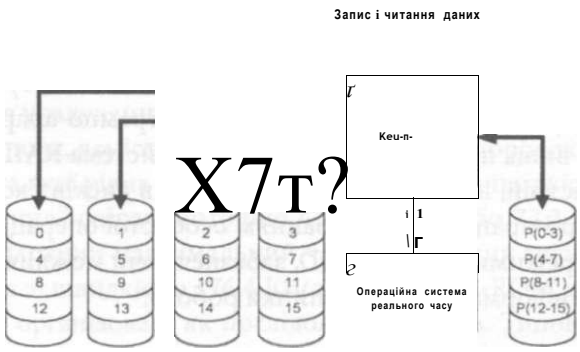


Рис. 9.41. Структура RAID 7

9.7.2.9. Тип дискових масивів RAID 10

RAID 10 також не входить до складу основних типів дискових масивів дана схема співпадає з RAID 0, але, на відміну від неї, роль окремих дисків виконують дискові масиви, побудовані за схемою RAID 1.

Таким чином, в RAID 10 поєднується розшарування пам'яті й дублювання даних (рис. 9.42). Це дозволяє досягти високої продуктивності, характерної для RAID 0, при рівні відмовостійкості, характерної для RAID 1. Основним недоліком цього типу RAID є висока вартість реалізації. Крім того, необхідність синхронізації всіх дисків приводить до ускладнення контролера.

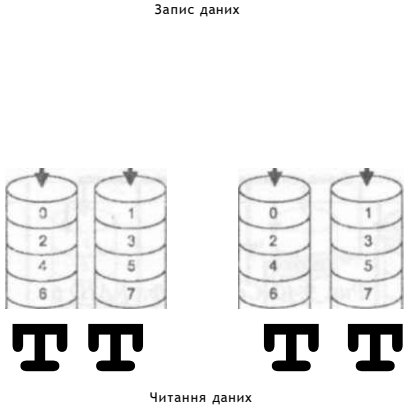


Рис. 9.42. Структура RAID 10

Таку конфігурацію дискових масивів називають дворівневим RAID або RAID 10 (RAID 0+1). Область її застосування: дешеві масиви, в яких головне - надійність зберігання даних.

Як вже було сказано вище, можливі й інші комбінації базових типів дискових масивів. Наприклад, при поєднанні технологій RAID 0 та RAID3, коли в додаток до принципів організації RAID 0 роль окремих дисків виконують дискові масиви, організовані за схемою RAID 3, отримується схема RAID 53. Часто застосовують розділення загального дискового простору на частини, причому об'єм кожної з них визначається користувачем. Одна частина функціонує як RAID одного типу, інша - як RAID іншого типу. Це дозволяє досягти вищої продуктивності та можливості резервного копіювання на випадок збою.

Потрібно відзначити, що керування масивами RAID може бути реалізоване програмно, як, наприклад, у Windows NT, апаратно або програмно-апаратно. При апаратній реалізації досягається вища продуктивність, причому система RAID може бути виконана як автономний пристрій, в якому поєднуються масив дисків і контролер. Контролер містить мікропроцесор і працює під керуванням особистої операційної системи, реалізовуючи різноманітні режими роботи RAID, забезпечуючи можливість заміни несправних дисків без втрати інформації та без зупинки роботи.

9.7.3. *Оптична пам'ять*

У 1983 році була представлена перша цифрова аудіосистема на базі компакт дисків (CD - compact disk). Компакт-диск - це односторонній диск, здатний зберігати більш ніж 60-хвилинну аудіоінформацію. Величезний комерційний успіх CD сприяв розвитку технології дешевих оптичних запам'ятовуючих пристроїв. За подальші роки були створені різні системи пам'яті на оптичних дисках, три з яких у прогресуючому ступені приживаються в комп'ютерах: CD-ROM, WARM і оптичні диски із стиранням.

9.7.3.1. *Постійна пам'ять на основі компакт дисків*

Для аудіо компакт-дисків і CD-ROM використовується ідентична технологія. Основна відмінність полягає у тому, що програвачі CD-ROM міцніші і містять пристрої для виправлення помилок, які забезпечують коректність передачі даних з диска у комп'ютер. Диск виготовляється з пластмаси, наприклад, полікарбонату, і покритий забарвленим шаром з високою віддзеркалюючою здатністю, зазвичай алюмінієм. Цифрова інформація наноситься у вигляді мікроскопічних поглиблень у відзеркалювальній поверхні. Запис інформації проводиться за допомогою сильно сфокусованого променя лазера високої інтенсивності. Так створюється так званий майстер-диск, з якого потім друкуються копії. Поглиблення на копії захищаються від пилу і пошкоджень шляхом покриття поверхні диска прозорим лаком

Інформація з диска зчитується малопотужним лазером, розташованим у програвачі. Лазер освітлює поверхню диска, що обертається, крізь прозоре покриття. Інтенсивність відбитого променя лазера міняється, коли він потрапляє в поглиблення на диску. Ці зміни фіксуються фотодетектором і перетворюються на цифровий сигнал

Поглиблення, розташовані ближче до центру диска, переміщаються щодо променя лазера повільніше, ніж більш віддалені. Через це необхідні заходи для компенсації відмінностей в швидкості так, щоб лазер міг прочитувати інформацію з постійною швидкістю

Одне з можливих вирішень аналогічне до вживаного в магнітних дисках - збільшення відстані між бітами інформації, залежно від її розташування на диску. В цьому випадку диск може обертатися з незмінною швидкістю і, відповідно, такі дискові пристрої відомі як пристрої з постійною кутовою швидкістю. Зважаючи на нераціональне використання зовнішньої частини диска метод постійної кутової швидкості в CD-ROM не підтримується. Натомість інформація по диску розміщується в секторах однакового розміру, які скануються з постійною швидкістю за рахунок того, що диск обертається із змінною швидкістю. В результаті поглиблення прочитуються лазером із постійною лінійною швидкістю. При доступі до інформації у зовнішнього краю диска швидкість обертання є меншою і зростає при наближенні до осі. Ємність доріжки і затримки обертання зростають зі зсувом від центру до зовнішнього краю диска

Випускаються CD-ROM різної ємності. У типовому варіанті відстань між доріжками складає 1,6 мкм, що, з урахуванням проміжків між доріжками, дозволяє забезпечити 20 344 доріжки. Фактично ж, замість безлічі концентричних доріжок, є одна доріжка у вигляді спіралі, довжина якої рівна 5,27 км. Постійна лінійна швидкість CD-ROM - 1,2 м/с, тобто для "проходження" спіралі потрібно 4391 секунд або 73,2 хвилини. Саме ця величина складає стандартний максимальний час програвання аудіодиска. Оскільки дані прочитуються з диска зі швидкістю 176,4 КБ/с, ємність CD-ROM рівна 774,57 МБ

Дані на CD-ROM організовані як послідовність блоків. Типовий формат блоку показаний на рис. 9.43.

Рис. 9.43. Формат блоку CD-ROM

Блок включає наступні поля:

- Синхронізацію. Це поле ідентифікує початок блоку і складається з нульового байта, десяти байтів, що містять тільки одиничні розряди, і знову байта зі всіх нулів.
- Ідентифікатор. Заголовок, що містить адресу блоку і байт режиму. Режим 0 визначає порожнє поле даних; режим 1 відповідає за використання коду, що коригує помилки, і наявність 2048 байт даних; режим 2 визначає наявність 2336 байт даних і відсутність коригуючого коду.
- Дані. Дані користувача.
- Коригуючий код (КК). Поле призначене для зберігання додаткових даних користувача в режимі 2, а в режимі 1 містить 288 байт коду з виправленням помилок.

Рис. 9.44 ілюструє організацію інформації на CD-ROM. Як вже наголошувалося, дані розташовані послідовно по спіралевидній доріжці. Для варіанту з постійною лінійною швидкістю довільний доступ до інформації стає складнішим.

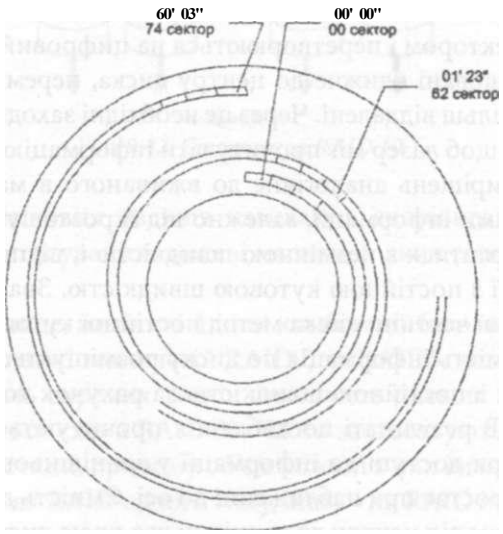


Рис. 9.44. Організація диска з постійною лінійною швидкістю

Останнім часом намітився перехід до нового типу оптичних дисків DVD (Digital Video Data). Диски DVD складаються з двох шарів завтовшки 0,6 мм, тобто мають дві робочих поверхні, і забезпечують зберігання по 4,7 Гбайт на кожній. У технології DVD використовується лазер з меншою довжиною хвилі (650 нм проти 780 нм для стандартних CD-ROM), а також витонченіша схема корекції. Все це дозволяє збільшити число доріжок і підвищити щільність запису. Крім того, при записі застосовується метод компресії інформації, відомий як MPEG2.

9.7.3.2. Оптичні диски із стиранням

Серед багатьох технологій оптичних дисків з можливістю багатократного перезапису інформації комерційно прийнятною виявилася тільки магнітооптична. У таких системах енергія лазерного променя використовується спільно з магнітним полем. Запис і стирання інформації відбуваються за рахунок реверсування магнітних полюсів маленьких областей диска, покритого магнітним матеріалом. Лазерний промінь нагріває

опромінювану пляму на поверхні і у цей момент магнітне поле може змінити орієнтацію магнітних полюсів на опромінюваній ділянці. Оскільки процес поляризації не викликає фізичних змін на диску, йому не страшні багатократні повторення. При читанні напрям магнітного поля можна визначити за поляризацією лазерного променя. Поляризоване світло, відбите від певної плями, змінює свій кут віддзеркалення залежно від характеру намагніченості.

9.7.4. *Магнітні стрічки*

Пам'ять на базі магнітних стрічок використовується в основному для архівації інформації. Носієм служить тонка стрічка полістиролу завширшки 0,38-2,54 см і завтовшки близько 0,025 мм, покрита магнітним шаром. Стрічка намотується на бобіни різного діаметру. Дані записуються послідовно, байт за байтом, від початку стрічки до її кінця. Час доступу до інформації на магнітній стрічці є значно більшим, чим у раніше розглянутих видів зовнішньої пам'яті.

Зазвичай уздовж стрічки розташовується 9 доріжок, що дозволяє записувати упоперек стрічки байт даних і біт парності. Інформація на стрічці групується в блоки - записи. Кожен запис відділяється від сусідньої міжблоковим проміжком, що дає можливість позиціювання головки зчитування/запису на початок будь-якого блоку. Ідентифікація запису проводиться по полю заголовка, що міститься в кожному записі. Для вказівки початку і кінця стрічки використовуються фізичні маркери у вигляді металізованих смужок, що наклеюються на магнітну стрічку, або прозорих ділянок на самій стрічці. Відомі також варіанти маркування початку і кінця стрічки шляхом запису на неї спеціальних кодів - індикаторів.

В комп'ютерах зазвичай застосовуються котушкові пристрої з вакуумними системами стабілізації швидкості переміщення стрічки. У них швидкість переміщення стрічки складає близько 300 см/с, щільність запису - 4 КБ/см, а швидкість передачі інформації - 320 КБ/с. Типова котушка містить 730 м магнітної стрічки.

У пам'яті на базі картриджів використовуються касети з двома котушками, аналогічні стандартним аудіокасетам. Типова ширина стрічки - 8 мм. Найбільш поширеною форхмою таких пристроїв є DAT (Digital Audio Tape). Дані на стрічку заносяться по діагоналі, як це прийнято у відеокасетах. За розміром такий картридж приблизно удвічі менший, ніж звичайна компакт-диск-касета, і має товщину 3,81 мм. Кожен картридж дозволяє зберігати декілька ГБ даних. Час доступу до даних невеликий (середнє між часами доступу до дискет і до жорстких дисків). Швидкість передачі інформації вища, ніж у дискет, але нижча, ніж у жорстких дисків.

Другим видом пам'яті на базі картриджів є пристрій стандарту DDS (Digital Data Storage). Цей стандарт був розроблений в 1989 році для задоволення вимог до резервного копіювання інформації з жорстких дисків у могутніх серверах і розрахованих на багато користувачів системах. По суті, це варіант DAT, що забезпечує зберігання 2 ГБ даних при довжині стрічки 90 м. У пізнішому варіанті стандарту DDS-DC (Digital Data Storage — Data Compression) за рахунок застосування методів стиснення інформації ємність стрічки збільшена до 8 ГБ. Нарешті, третій тип запам'ятовуючого пристрою на базі картриджів також призначений для резервного копіювання вмісту жорстких дисків, але при менших об'ємах такої інформації. Цей тип відповідає стандарту QIC (Quarter Inch

Cartridge tape) і відоміший під назвою стример. Відомі стримери, що забезпечують зберігання від 15 до 525 МБ інформації. Залежно від інформаційної місткості і фірми-виготовлювача змінюються і характеристики таких картриджів. Так, число доріжок може варіюватися в діапазоні від 4 до 28, довжина стрічки - від 36 до 300 м і т.д.

9.8. Короткий зміст розділу

У даному розділі розглянуті питання організації роботи з пам'яттю комп'ютера. В комп'ютері використовуються різні типи пам'яті, які, залежно від способу доступу до інформації, можуть бути класифіковані наступним чином: пам'ять з довільним доступом, із впорядкованим доступом та з асоціативним доступом. Кожний тип пам'яті має свої переваги та недоліки, які визначають місце його використання в комп'ютері.

Розглянута структура пам'яті комп'ютера та типи пам'яті, які входять до складу внутрішньої та зовнішньої пам'яті комп'ютера. Приведено основні характеристики пам'яті: ємність пам'яті, організація пам'яті, швидкодія пам'яті, час доступу до пам'яті, період звернення до пам'яті, вартість. Названо типи пам'яті залежно від технології виготовлення, енергозалежності, за методом доступу до даних.

Проаналізовано можливі варіанти організації регістрових файлів процесорів. Показано обмеження, які спричиняє використання багатопортового регістрового файла. Розглянуто ряд нових структур регістрового файла: інтегрованого регістрового файла, розподіленого регістрового файла, кластерного розподіленого регістрового файла, розподіленого регістрового файла з керованою комутацією, розподіленого регістрового файлу з віконною організацією, ієрархічного регістрового файла. Розглянута динамічна та статична організація даних в регістрових файлах. Описана робота регістрового файла на базі черги з програмованою затримкою.

Описано галузі ефективного використання пам'яті з асоціативним доступом та принципи її роботи, зокрема, як реалізується запис нової інформації та як реалізується зчитування інформації з пам'яті з асоціативним доступом. Показано які види пошуку можна здійснювати в пам'яті з асоціативним доступом. Описано чотири основні елементи організації пам'яті з асоціативним доступом: з повним паралельним асоціативним доступом, з неповним паралельним асоціативним доступом, з послідовним асоціативним доступом, з частково асоціативним доступом.

Описано види запам'ятовуючих пристроїв, які може містити основна пам'ять. Приведено можливі варіанти побудови блокової пам'яті та можливості по скороченню часу доступу до інформації, які надає блокова організація пам'яті. Пояснено чим обумовлена ефективність розшарування пам'яті. Показано як здійснюється нарощування розрядності основної пам'яті. Описано топологію запам'ятовуючих елементів, яка лежить в основі організації напівпровідникових ЗП. Дано пояснення призначення керуючих сигналів в мікросхемі пам'яті. Описано як побудовано ПЗП, що запрограмований при виготовленні, одноразово запрограмований після виготовлення та багаторазово програмовний ПЗП.

Приведено порядок розміщення інформації на магнітному диску. Описані типи сучасних дискових систем. Пояснено для чого використовуються масиви магнітних дисків з надлишковістю та робота шести базових типів дискових масивів RAID: RAID 0,

RAID 1,..., RAID 5 та дискових масивів, створених на їх основі. Описана робота оптичної пам'яті та пам'яті на магнітних стрічках.

9.9. *Література для подальшого читання*

Структура пам'яті комп'ютера та типи пам'яті, які входять до складу внутрішньої та зовнішньої пам'яті комп'ютера приведені в книгах [1-3,5, 7-11]. Тут же приведено основні характеристики пам'яті: ємність, організація, швидкодія, час доступу, період звернення, вартість. Класифікація різних типів пам'яті залежно від способу доступу до даних приведена в [5]. В роботі [6] було запропоновано новий тип пам'яті, яка за способом доступу до даних відноситься до пам'яті з впорядкованим доступом. Принципи побудови та організації роботи пам'яті з довільним доступом детально розглянуті в літературі [8-11].

В праці [26] проведена класифікація та детально розглянуті структури та принципи організації використовуваних у комп'ютерах регістрових файлів. У роботі [22] описано віконну організацію регістрового файла. Структуру ієрархічного регістрового файла запропоновано в роботі [14]. У роботах [14, 15] пропонується дворівнева модель регістрового файла для динамічних суперскалярних архітектур, яка дозволяє зменшити кількість регістрів та кількість портів. Така дворівнева організація зменшує кількість портів у три рази, а також покращує час доступу до даних на 46%. Регістровий файл процесора SPARC є також ієрархічний, оскільки містить безпосередньо регістрові вікна та асоціативний регістровий кеш [16, 23]. В роботі [25] пропонується будувати регістровий файл на основі черг за принципом FIFO.

В роботах [25, 26] пропонується організувати регістровий файл на основі черги з програмованою затримкою.

Асоціативна пам'ять описана в [1]. Існує цілий ряд алгоритмів, що дозволяють організувати впорядковану вибірку даних з цієї пам'яті. Детальний їх опис і порівняльний аналіз можна знайти в [11].

У 1987 році Паттерсон (Patterson), Гібсон (Gibson) і Катц (Katz) з каліфорнійського університету Берклі опублікували статтю [ЗО]. У цій статті описувалися різні типи дискових масивів, що позначаються скороченням RAID.

9. 7 0. *Література до розділу 9*

1. Искусственный интеллект: В 3-х книгах. Кн. 3. Программные и аппаратные средства: Справочник / Под ред. В Н. Захарова, В Ф. Хорошевского. - М.: Радио и связь, 1990. - 191 с

2. Каган Б М. Электронные вычислительные машины и системы. - М.: Энергия, 1979.- 528 с

3. Каган Б М., Каневский М М. Цифровые вычислительные машины и системы. - М.: Энергия, 1974. - 680 с

4. Мельник А О. Програмовані процесори обробки сигналів. - Львів: Видавництво Національного університету "Львівська політехніка", 2000. - 55 с

5. Угрюмов Е П. Цифровая схемотехника. - СПб.: БХВ - Санкт-Петерберг, 2000. - 528 с

6. Мельник А О. Принципи побудови буферної сортувальної пам'яті. Вісник Державного університету "Львівська політехніка" "Комп ютерна інженерія та інформаційні технології", N 307, 1996.- С. 65-71.

7. Б Н Малиновский и др. Справочник по цифровой вычислительной технике. - К Техніка, 1980. - 320 с

8. Шигин А Г., Дерюгин А А. Цифровые вычислительные машины (память ЦВМ). - М.: Энергия, 1975. - 536 с

9. D. Patterson, J. Hennessy. Computer Architecture. A Quantitative Approach. Morgan Kaufmann Publishers, Inc. 1996.

10. Patterson, D. A., & Hennessy, J. L. *Computer Organization and Design, The Hardware/Software Interface,* 2nd ed , San Mateo, CA: Morgan Kaufmann, 1997.

11. Метлицкий Б А., Каверзнев В В. Системы параллельной памяти. Теория, проектирование, применение. Под ред. В И. Тихонова. - Л., 1989.

12. David J. Kuck. The Structure of Computers and Computations. John Wiley & Sons, Pittsburgh, Pennsylvania, 1978.

13. S.Y. Kung, VLSI Array Processors, Prentice Hall, 1988.

14. Marcio Merino Fernandes, Josep Llosa, Nigel Topham. Using Queues for Register File Organization in VLIW Architectures. Technical Report ECS-CSG 29-97, Dept of Computer Science, University of Edinburgh, 1997.

15. Henk Corporaal, Microprocessor Architectures: From VLIW to TTA, John Wiley 8c Sons, Inc., New York, NY, 1997.

16. Greg Blank and Steve Krueger. SuperSPARC: A fully integrated superscalar processor. In Hot Chips III. A Symposium on High-Performance Chips, IEEE, August 1991.

17. CEVA: CEVA-X1620 Datasheet. CEVA, 2005.

18. Texas Instruments: TMS320C64x Technical Overview. 2005. - www.ti.com

19. S. Rixner, W. Dally, B. Khailany, P. Mattson, U. Kapasi. Register organization for media processing. International Symposium on High Performance Computer Architecture (HPCA), pp. 375-386, 2000.

20. Balasubramonian, R., Dwarkadas, S., Albonesi, D. Reducing the Complexity of the Register File in Dynamic Superscalar Processor. In Proceedings of the 34th International Symposium on Microarchitecture, December 2001.

21. R. M. Russell. The CRAY-1 computer system. Communications of the ACM, 21(l):63-72, Jan. 1978.

22. R. Ravlndran, R. Senger, E. Marsman, G. Dasika, M. Guthaus, S. Mah/ke, andR. Brown. Increasing the Number of Effective Registers in a Lo w-Po wer Processor Using a Windo wed Register File. Proc. 2003

23. David L. Weaver and Tom Germond. The SPARC Architecture Manual, Version9. Sparc International and PTR Prentice Hall, Englewood Cliffs, NJ, 1994.

24. Мельник А О. Спеціалізовані системи реального часу: конспект лекцій. - Львів: навч видання, 1996. - 53 с.

25. Мельник А О., Сало А М. Методика проектування паралельного процесора на основі пам'яті з детермінованою вибіркою // Вісник НУ "Львівська політехніка". - № 546,2005. - С. 96-101.

26. Мельник А О., Сало А М. Регістровий файл.// Вісник НУ "Львівська політехніка", 2007. - С. 96-101.

27. Rolf Hakenes. A novel low-power microprocessor architecture, ***www.iccd-conference. org/proceedings/2000/08010141.pdf***

28. Gregory W. A Comparison of Circuits for On-Chip Programmable Crossbar Switches // 10th NASA Symposium on VLSI Design, Albuquerque, NM, March 20-21, 2002.

29. ***www.xilinx. com***

30. Patterson, Gibson, Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID).

9.11. Питання до розділу 9

1. Поясніть принципи організації пам яті з довільною вибіркою
2. Як зв язані адреса і ємність пам'яті?
3. Назвіть операції пам'яті
4. Опишіть структуру пам'яті комп'ютера
5. Які типи пам'яті входять до складу внутрішньої пам'яті комп'ютера?
6. Які типи пам'яті входять до складу зовнішньої пам'яті комп'ютера?
7. Які типи пам'яті є в процесорі?
8. Які операції визначає поняття "звернення до пам'яті"?
9. Назвіть основні характеристики пам'яті
10. Які одиниці вимірювання використовуються для вказівки ємності пам яті?
11. Що таке "організація пам'яті"?
12. Якими характеристиками описується швидкодія пам'яті?
13. Що таке час доступу до пам'яті?
14. Що таке період звернення до пам яті?
15. У чому полягає відмінність між часом доступу і періодом звернення до пам'яті?
16. Назвіть типи пам'яті залежно від технології виготовлення
17. Які одиниці використовуються для оцінки вартості пам'яті?
18. Назвіть типи енергонезалежної пам'яті
19. Назвіть типи енергозалежної пам'яті
20. Приведіть класифікацію пам'яті за методом доступу до даних
21. Що таке регістровий файл процесора?
22. Чи є регістри регістрового файла програмно доступними? Як це розуміти?
23. Наведіть типи регістрових файлів
24. Наведіть структуру інтегрованого регістрового файла
25. Як здійснюється запис даних до інтегрованого регістрового файла?
26. Як здійснюється зчитування даних з інтегрованого регістрового файла?
27. На що впливає збільшення кількості портів інтегрованого регістрового файла?
28. Що таке розподілений регістровий файл?
29. Приведіть організацію кластерного розподіленого регістрового файла
30. Яка перевага кластерного розподіленого регістрового файла в порівнянні з інтегрованим регістровим файлом?
31. Як організовано кластерний розподілений регістровий файл процесора TM5320C6xx?
32. Приведіть організацію розподіленого регістрового файла з керованою комутацією
33. Як працює розподілений регістровий файл з віконною організацією?
34. Що дає застосування ієрархічного регістрового файла?
35. Поясніть різницю між динамічною та статичною організацією даних в регістрових файлах
36. Які переваги в динамічній організації збереження даних в регістрових файлах порівняно із статичною?
37. Поясніть роботу регістрового файла на базі черги з програмованою затримкою
38. Де ефективно використовувати пам'ять з асоціативним доступом?
39. Поясніть принципи роботи пам'яті з асоціативним доступом
40. Для чого використовується регістр збігів у пам'яті з асоціативним доступом?
41. Поясніть призначення маски в пам'яті з асоціативним доступом
42. Як реалізується запис нової інформації в пам'ять з асоціативним доступом?
43. Як реалізується зчитування інформації з пам'яті з асоціативним доступом?
44. Які види пошуку можна здійснювати в асоціативному ЗП?
45. Назвіть чотири основні елементи організації пам'яті з асоціативним доступом

46. Чим відрізняються простий і складний пошуки інформації в пам'яті з асоціативним доступом?

47. Поясніть організацію роботи пам'яті з повним паралельним асоціативним доступом

48. Поясніть організацію роботи пам'яті з неповним паралельним асоціативним доступом

49. Поясніть організацію роботи пам'яті з послідовним асоціативним доступом

50. Поясніть організацію роботи пам'яті з частково асоціативним доступом

51. Які види запам ятовуючих пристроїв може містити основна пам'ять?

52. Охарактеризуйте можливі варіанти побудови блокової пам'яті

53. Які можливості по скороченню часу доступу до інформації надає блокова організація пам'яті?

54. Чим обумовлена ефективність розшарування пам яті?

55. Як здійснюється нарощування розрядності основної пам'яті?

56. Яка топологія запам ятовуючих елементів лежить в основі організації напівпровідникових ЗП?

57. Яку мінімальну кількість ліній повинен містити стовпець МС пам яті?

58. Поясніть призначення керуючих сигналів в мікросхемі пам'яті

59. Чим обумовлена необхідність регенерації вмісту динамічних ОЗП?

60. Охарактеризуйте основні сфери застосування статичних і динамічних ОЗП

61. Який вид ПЗП має найвищу швидкість перепрограмування?

62. Як побудовано ПЗП, що запрограмований при виготовленні?

63. Як побудовано ПЗП, який одноразово запрограмований після виготовлення?

64. Приведіть структуру матриці одноразово програмованого ПЗП

65. Як побудовано багаторазово програмований ПЗП?

66. Назвіть типи багаторазово програмованих ПЗП

67. Як розміщена інформація на магнітному диску?

68. Назвіть типи сучасних дискових систем

69. Для чого використовуються масиви магнітних дисків з надлишковістю?

70. Поясніть роботу шести базових типів дискових масивів RAID: RAID 0, RAID 1,..., RAID 5 та дискових масивів, створених на їх основі

71. Назвіть типи оптичної пам яті

72. Як організована пам'ять на магнітних стрічках?

Розділ 10

Організація пам'яті

В комп'ютері використовується декілька різних типів пам'яті. Це пов'язано з тим, що неможливо одним типом пам'яті вирішити всі завдання, які стоять перед нею, в першу чергу забезпечити велику ємність та високу швидкодію. Разом з тим, постає завдання забезпечення ефективної взаємодії всіх типів пам'яті, щоб система пам'яті в цілому задовольняла всі вимоги з боку інших вузлів комп'ютера. Як буде показано в даному розділі, це можливо здійснити завдяки використанню при побудові системи пам'яті підходу, відомого як принцип ієрархічної організації пам'яті.

Відповідно до цього принципу пам'ять комп'ютера будується на основі пристроїв пам'яті різних типів, які, залежно від характеристик, належать до певного рівня ієрархії. Пам'ять нижчого рівня має меншу ємність, швидша і має більшу вартість в перерахунку на біт, ніж пам'ять вищого рівня. Рівні ієрархії взаємозв'язані: всі дані на одному рівні можуть бути також знайдені на вищому рівні, і всі дані на цьому вищому рівні можуть бути знайдені на наступному вищому рівні і т. д. З рухом вверх по ієрархічній структурі зменшується співвідношення вартість/біт, зростає ємність та час доступу. Однак завдяки принципу локальності за зверненням з рухом вверх по ієрархічній структурі зменшується частота звернення до пам'яті з боку центрального процесора, що веде до зменшення загальної вартості при заданому рівні продуктивності.

Принципи забезпечення ефективної взаємодії між рівнями ієрархії пам'яті розглядаються в даному розділі.

10.1. Ієрархічна організація пам'яті комп'ютера

10.1.1. Різниця між продуктивністю процесора та пам'яті

Одним із вузьких місць комп'ютерів з архітектурою Джона фон Неймана є розрив в продуктивності процесора та пам'яті, причому цей розрив неухильно збільшується. Так, продуктивність процесора зростає вдвічі приблизно кожні 1,5 роки. В табл. 10.1 наведено ріст з роками тактової частоти роботи процесора на прикладі процесорів фірми Intel.

Таблиця 10.1

Роки	1980	1985	1990	1995	2000	2005	2005-1980
Тип процесора	8080	286	386	Pentium	PIП	P-IV	
Тактова частота (МГц)	1	6	20	150	750	3800	3800 разів
Час одного такту (не)	1 000	166	50	6	1.6	0.26	3800 разів

Разом з тим, для пам'яті приріст швидкодії не є таким високим, як у процесорів. В табл. 10.2 наведено зміну з роками ціни зберігання одного МБ інформації, часу доступу до статичної (SRAM) і динамічної (DRAM) напівпровідникової пам'яті та дискової пам'яті, а також ємність основної та зовнішньої дискової пам'яті.

Таблиця 10.2

	Хіфазстеристики*'	1980	1985	1990	1995	2000	2005	Ріст протягом 2005-1980
RAM	Ціна (S МБ)	19200	2900	320	256	100	50	330 разів
	Час доступу (не)	300	150	35	15	2	0.5	300 разів
DRAM	Ціна (S МБ)	8000	880	100	3«	1	0.1	80 000 разів
	Час доступу (не)	375	200	100	70	50	зо	12 разів
	Типовий об'єм ОП на базі DRAM (МБ)	0.064	0.256	4	16	64	258	4 000 разів
DISC	Ціна (S МБ)	500	100	8	0.3	0.05	0 01	5 000 разів
	Час доступу (не)	300	150	35	15	2	0.5	300 разів
	Типовий об'єм дискової пам'яті (МБ)	1	10	160	1 000	9 000	80000	S0000 разів

Виходячи з наведених даних в табл. 10.1 та табл. 10.2, на рис. 10.2 показана зміна з роками часу доступу до статичної T_{SRAM} та динамічної T_{DRAM} напівпровідникової та дискової T_{DISK} пам'яті, а також тактової частоти роботи процесора T_{CPU}. Як видно з наведеного рисунку, існує значна різниця між тактовою частотою процесора і часом доступу до динамічної пам'яті. Причому з роками розрив у значеннях часових характеристик між процесором і динамічною напівпровідниковою пам'яттю та дисковою пам'яттю зростає.

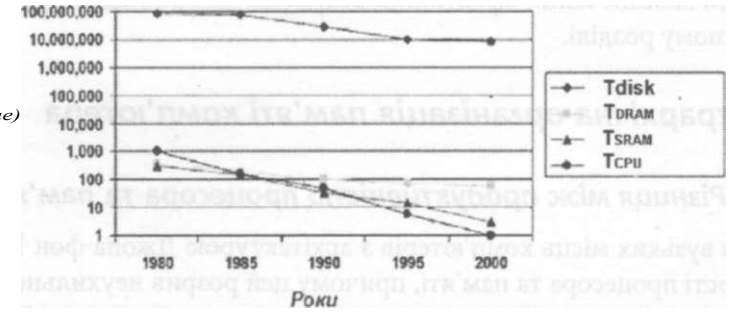


Рис. 10.2. Зміна з роками часових характеристик статичної і динамічної напівпровідникової та дискової пам'яті, а також процесора

Особливо важливим з точки зору архітектури комп'ютера є те, що зростає розрив між тактовою частотою процесора і часом доступу до динамічної напівпровідникової пам'яті DRAM, на основі якої будується основна пам'ять комп'ютера. Адже якраз між цими двома вузлами комп'ютера здійснюється основна частка обмінів інформацією. В той час, коли продуктивність процесора зростає щороку на 60 % (подвоєння за

1,5 року), ріст продуктивності динамічної напівпровідникової пам'яті не перевищує 9 % в рік (подвоєння за 10 років). Це виражається в збільшенні розриву в продуктивності між процесором і динамічною напівпровідниковою пам'яттю на 50 % в рік (рис. 10.3).

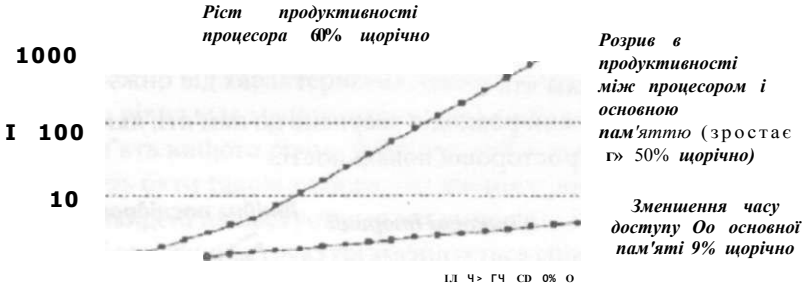


Рис. 10.3. Збільшення розриву в продуктивності між процесором і пам'яттю

Разом з тим, з рис. 10.2 видно, що розрив з роками між продуктивністю процесора і статичної напівпровідникової пам'яті є незначним. Виглядає доцільним побудова основної пам'яті комп'ютера на базі статичної напівпровідникової пам'яті. Однак це не так, оскільки, як видно з табл. 10.2, зберігання 1МБ інформації в такій пам'яті є приблизно в 100 разів дорожчим, аніж в динамічній напівпровідниковій пам'яті. Тому був знайдений наступний вихід з цієї ситуації - включення між основною пам'яттю комп'ютера і процесором додаткової швидкої пам'яті малої ємності, побудованої на основі статичної напівпровідникової пам'яті.

Іще більший розрив існує між продуктивністю зовнішньої дискової і динамічної напівпровідникової пам'яті. Однак, як видно з табл. 10.2, вартість зберігання 1МБ інформації в дисковій пам'яті приблизно в 20 разів менша, ніж в динамічній напівпровідниковій пам'яті. Крім того, вона забезпечує зберігання значно більших об'ємів інформації. Тому і тут виникає потреба пошуку механізму, який би забезпечив прискорення обміну між основною і зовнішньою пам'яттю.

10.1.2. Властивість локальності за зверненням до пам'яті

Якщо розглянути процес виконання більшості програм, то з дуже високою ймовірністю можна спрогнозувати, що адреса чергової команди програми або слідує безпосередньо за адресою, за якою була зчитана поточна команда, або розташована поблизу неї. Таке розташування адрес називається просторовою локальністю команд програми. Аналогічно можна стверджувати і про дані, які, як правило, є структурованими і, зазвичай, зберігаються в послідовних комірках пам'яті. Дана особливість називається просторовою локальністю даних.

Крім того, програми містять безліч невеликих циклів і підпрограм. Це означає, що невеликі набори команд можуть багато разів повторюватися протягом деякого інтервалу часу, тобто має місце часова локальність.

Таким чином, існує дві передбачувані властивості програм при зверненні до пам'яті:

• просторова локальність - якщо відбулося звернення до деякої комірки пам'яті, то з великою ймовірністю можна стверджувати, що в найближчий час відбудеться звернення до сусідньої комірки пам'яті;

• часова локальність - якщо відбулося звернення до деякої комірки пам'яті, то з великою ймовірністю можна стверджувати, що в найближчий час знову відбудеться звернення до тієї ж комірки пам'яті.

На рис. 10.4 наведено типовий розподіл звернень до пам'яті, який ілюструє наведену вище властивість часової та просторової локальності.

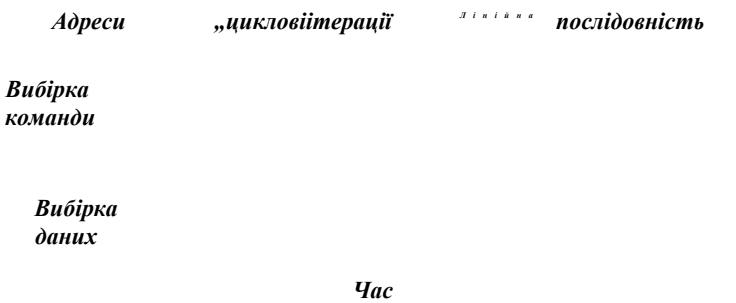


Рис. 10.4. Типовий розподіл звернень до пам'яті

Обидва види локальності об'єднує поняття локальності за зверненням. Властивість локальності можна виразити в чисельній формі і представити у вигляді так званого правила «90/10»: 90 % часу роботи програми пов'язано з доступом до 10 % адресного простору цієї програми.

З властивості локальності витікає, що програму розумно представити у вигляді послідовно оброблюваних фрагментів - компактних груп команд і даних. Поміщаючи такі фрагменти в швидшу пам'ять, можна істотно понизити загальні затримки на звернення до пам'яті, оскільки команди і дані, будучи один раз передані з повільного пристрою пам'яті в швидкий, потім можуть використовуватися багато разів, і середній час доступу до них в цьому випадку визначається вже швидшим пристроєм пам'яті. Це дозволяє зберігати великі програми і масиви даних на повільних, великої ємності, але дешевих пристроях пам'яті, а в процесі обробки активно використовувати швидку пам'ять порівняно невеликої ємності, збільшення якої пов'язане з великими витратами.

10.1.3. Принцип ієрархічної організації пам'яті

З проведеного вище аналізу можна зробити висновки про наступні фундаментальні та стабільні протягом тривалого часу властивості апаратних та програмних засобів сучасного комп'ютера:

• чим менший час доступу до пам'яті, тим менша її ємність та вища вартість зберігання в ній одного біта інформації;

• чим більша ємність пам'яті, тим більший час доступу до неї та нижча вартість зберігання в ній одного біта інформації;

• існує значна різниця між продуктивністю процесора і основної пам'яті, а також між продуктивністю основної та зовнішньої пам'яті;

• комп'ютерні програми наділені властивістю локальності за зверненням до пам'яті.

Ці фундаментальні властивості позитивно доповнюють одна одну з точки зору вирішення завдання забезпечення необхідної ємності та високої швидкодії пам'яті за прийнятну ціну. Вони є підґрунтям доцільності використання при побудові системи пам'яті підходу, відомого як принцип ієрархічної організації пам'яті.

Відповідно до цього принципу пам'ять комп'ютера складається із пристроїв пам'яті різних типів, які, залежно від характеристик, належать до певного рівня ієрархії.

Пам'ять нижчого рівня має меншу ємність, швидша і має велику вартість в перерахунку на біт, ніж пам'ять вищого рівня. Рівні ієрархії взаємозв'язані: всі дані на деякому нижчому рівні можуть бути також знайдені на вищому рівні, і всі дані на цьому вищому рівні можуть бути знайдені на наступному вищому рівні і т. д.

З рухом вверх по ієрархічній структурі зменшується співвідношення вартість/біт, зростає ємність та час доступу. Однак завдяки властивості локальності за зверненням з рухом вверх по ієрархічній структурі зменшується частота звернення до пам'яті з боку нижчих рівнів, що веде до зменшення загальної вартості при заданому рівні продуктивності.

На кожному рівні ієрархії пам'ять розбивається на блоки, які є найменшою інформаційною одиницею, що пересилається між двома сусідніми рівнями ієрархії. Розмір блоків може бути фіксованим або змінним. При фіксованому розмірі блоку ємність пам'яті зазвичай кратна його розміру. Розмір блоків на кожному рівні ієрархії найчастіше різний і збільшується від нижчих рівнів до вищих.

Процесор взаємодіє з пам'яттю найнижчого рівня ієрархії, яка розміщена найближче до нього. При зверненні з боку процесора до пам'яті, наприклад, для зчитування команд або даних, проводиться їх пошук в пам'яті нижнього рівня. Факт виявлення потрібної інформації називають попаданням (hit), інакше говорять про промах (miss). При промаху проводиться пошук потрібної інформації в пам'яті наступного вищого рівня, де також можливі попадання або промахи. Після виявлення необхідної інформації виконується послідовне пересилання вмісту блоку з шуканою інформацією з вищих рівнів на нижчі. Слід зазначити, що незалежно від числа рівнів ієрархії, пересилання інформації може здійснюватися лише між двома сусідніми рівнями, тобто пересилання інформації в обхід деяких рівнів не допускається.

При проведенні обміну інформацією блоками між пристроями пам'яті різних рівнів ієрархії необхідно вирішувати наступні питання:

- вибрати правило заміщення вмісту одних блоків вмістом інших блоків, оскільки через меншу ємність в пам'яті нижчого рівня не може бути така сама кількість блоків, як в пам'яті вищого рівня;
- визначити допустиме місце розташування нового вмісту блоку при записі з дотриманням правила заміщення;
- вибрати та дотримуватись стратегії запису з метою забезпечення узгодженості копій вмісту одних і тих же блоків, розташованих на різних рівнях, при записі нової інформації в копію, що знаходиться на нижчому рівні.

10.1.4. Характеристики ефективності ієрархічної організації пам'яті

При оцінці ефективності ієрархічної організації пам'яті використовують наступні характеристики:

- коефіцієнт попадань відповідного рівня ієрархії пам'яті - відношення числа звернень з пам'яті даного рівня ієрархії до пам'яті наступного вищого рівня, при яких

відбулося попадання, до загального числа звернень з пам'яті даного рівня ієрархії до пам'яті наступного вищого рівня ієрархи. Попадання - факт виявлення потрібної інформації при зверненні до пам'яті наступного вищого рівня;

- коефіцієнт промахів - відношення числа звернень до пам'яті наступного вищого рівня, при яких мав місце промах, до загального числа звернень до пам'яті даного рівня ієрархії. Якщо позначити коефіцієнт попадань через $к_{п}$, а коефіцієнт промахів через $к_{т}$, то залежність між ними можна виразити наступною формулою: $к_{т} = 1 - к_{п}$;
- час звернення при попаданні - час, необхідний для пошуку потрібної інформації в пам'яті нижчого рівня (включаючи з'ясування, чи є звернення попаданням), плюс час на фактичне зчитування даних;
- втрати на промах - час, потрібний для заміни блоку в пам'яті нижчого рівня на блок з потрібними даними, розташований у пам'яті наступного (вищого) рівня.
- середній час доступу до пам'яті, який визначається з виразу: $т_{д} = :_{п} + к_{т} г_{р}$, де ^ - час звернення при попаданні, і - втрати на промах. Імовірність попадання в сучасних комп'ютерах є досить високою (біля 95 % від загальної кількості звернень), і, відповідно, коефіцієнт промахів є низьким. Однак, через велику різницю між швидкодією пам'яті нижчого і вищого рівнів, навіть такий низький коефіцієнт промахів суттєво впливає на середній час доступу до пам'яті. Так, якщо прийняти, що час доступу до пам'яті нижчого рівня складає 1 такт, а час доступу до пам'яті вищого рівня в 10 разів більший, то середній час доступу буде рівним 1 = 1 + 0.05 *10 = 1.5 такти.

10.1.5. Ієрархічна пам'ять сучасного комп'ютера

Структура ієрархічної пам'яті сучасного комп'ютера представлена на рис. 10.5.

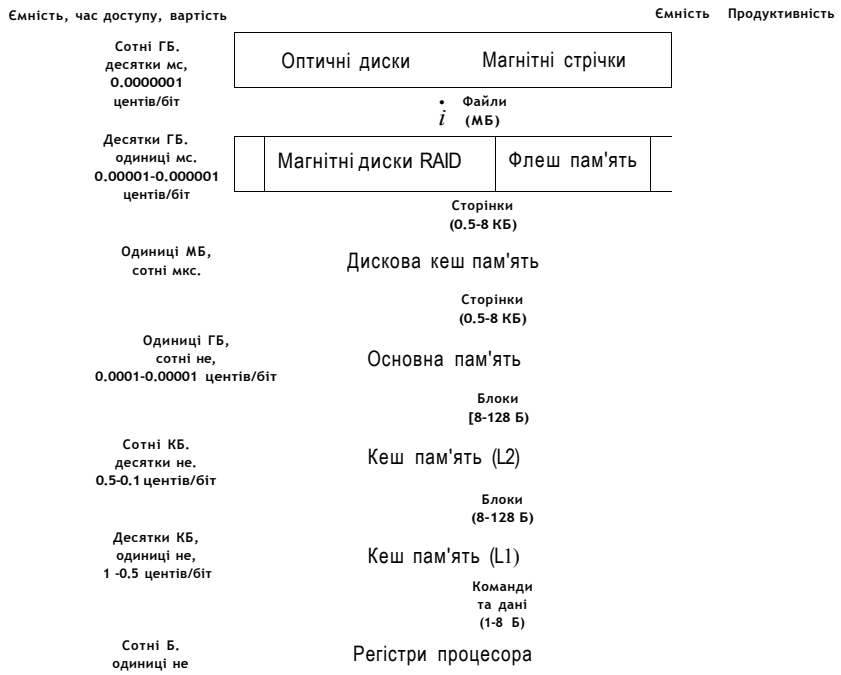


Рис. 10.5. Ієрархічна пам'ять

Найшвидший, але і мінімальний за ємністю, тип пам'яті - це внутрішні регістри процесора, тобто регістровий файл. Як правило, кількість регістрів в регістровому файлі невелика (типово 16-128). У архітектурі комп'ютера із спрощеною системою команд їх число може доходити до декількох сотень. Основна пам'ять має значно більшу ємність. Між регістрами процесора і основною пам'яттю часто розміщують кеш пам'ять, яка за ємністю відчутно програє основній пам'яті, але істотно перевершує останню за швидкодією, поступаючись у той же час регістровому файлу. У більшості сучасних комп'ютерів є декілька рівнів кеш пам'яті, які позначають буквою L з номером рівня кеш пам'яті. На рис. 10.5 показані два таких рівні. В останніх комп'ютерах все частіше з'являється також третій рівень кеш пам'яті (L3), причому розробники комп'ютерів говорять про доцільність введення і четвертого рівня - L4. Кожен наступний рівень кеш пам'яті має більшу ємність, але одночасно і меншу швидкодію в порівнянні з попереднім. За швидкодією будь-який рівень кеш пам'яті перевершує основну пам'ять. Чотири нижні рівні ієрархії утворюють внутрішню пам'ять комп'ютера, а всі вищі за них рівні - це зовнішня або вторинна пам'ять.

Всі види внутрішньої пам'яті реалізуються на основі напівпровідникових технологій і, в основному, є енергозалежними.

Довготривале зберігання великих об'ємів інформації (програм і даних) забезпечується зовнішньою пам'яттю, першим рівнем якої є пам'ять на базі магнітних дисків. Після дискової пам'яті йдуть пристрої архівної пам'яті, серед яких найпоширеніші пристрої на базі оптичних дисків, на магнітних стрічках та флеш пам'ять.

Нарешті, ще один рівень ієрархії може бути доданий між основною пам'яттю і магнітними дисками. Цей рівень носить назву дискової кеш пам'яті і реалізується у вигляді пристрою, що входить до складу магнітного диску. Дискова кеш пам'ять істотно покращує швидкість обміну інформацією між магнітними дисками і основною пам'яттю.

Розглянемо далі принципи обміну інформацією між двома базовими рівнями: між процесором і основною пам'яттю та між основною і зовнішньою пам'яттю.

10.2. Організація обміну інформацією між процесором і основною пам'яттю через кеш пам'ять

10.2.1. Кеш пам'ять в складі комп'ютера

Звернення процесора до основної пам'яті завжди локалізовано в невеликому діапазоні змін її адрес. Застосування кеш пам'яті ґрунтується на обох принципах локальності за зверненням: в ній використовується часова локальність, оскільки зберігається вміст комірок, до яких недавно відбулося звернення, та використовується просторова локальність, оскільки в блоках даних зберігається вміст множини сусідніх комірок.

Кеш пам'ять розташована між процесором і основною пам'яттю. Це швидка буферна пам'ять невеликої ємності. Кеш пам'ять працює на близькій тактовій частоті до процесора і не пригальмовує його роботу. Кеш пам'ять (cache в перекладі з англ. - тайник) залишається прозорою для програміста, тому що система команд процесора, як правило, не містить команд роботи з кеш пам'яттю.

Типовий фрагмент структури сучасного комп'ютера, в якому використовується два рівні кеш пам'яті, подано на рис. 10.6. Для зберігання даних і команд в кеш пам'яті пер-

шого рівня використано розділені кеш пам яті даних і команд. В свою чергу, обидві кеш пам'яті першого рівня зв'язані з єдиною кеш пам'яттю другого рівня, яка взаємодіє з основною пам'яттю. Зрозуміло, що обмін в підсистемі "кеш пам'ять Ь2 - кеш пам'ять даних ІЛ" є двостороннім, а в підсистемі "кеш пам'ять Ь2 - кеш пам'ять команд ІЛ" - одностороннім.

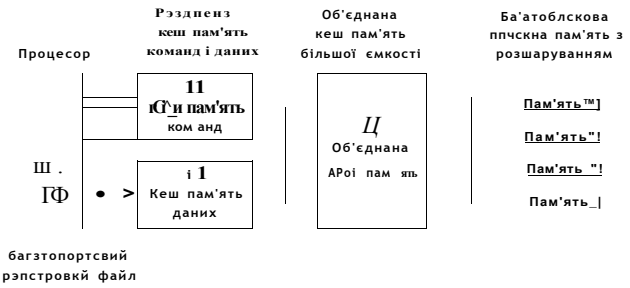


Рис. 10.6. Фрагмент структури сучасного комп'ютера

Наведена структура ефективно поєднала риси Принстонської і Гарвардської архітектур. Такий підхід вигідніший, ніж побудова швидкої основної пам'яті без використання проміжної кеш пам'яті.

10.2.2. Порядок взаємодії процесора і основної пам'яті через кеш пам'ять

Основні правила організації кеш пам'яті та основної пам'яті, які забезпечують їх ефективну взаємодію, є наступними:

- Кеш і основна пам'яті діляться на блоки однакового розміру, тобто вони можуть зміщувати однакову кількість слів.
- Базовою порцією інформації, яка переміщується між основною та кеш пам'яттю, є вміст одного блоку.
- Кожний блок має свій номер.
- Нумерація комірок в кожному блоці однакова.
- В кожний момент часу в блоках кеш пам'яті знаходиться вміст декількох переписаних із основної пам'яті блоків.
- Кожне слово кеш пам'яті супроводжується адресним тегом, що вказує на те, вміст якого блоку основної пам'яті є переписаним до блоку кеш пам'яті, в якому знаходиться слово.
- Ідентичність вмісту блоків кеш пам'яті та основної пам'яті забезпечується використанням спеціальних методів оновлення вмісту блоків основної пам'яті.
- Заміна вмісту одних блоків в кеш пам'яті вмістом інших блоків з основної пам'яті здійснюється за правилами, які називають алгоритмом заміщення.
- Між блоками основної пам'яті та кеш пам'яті встановлюється відповідність, яка задається функцією відображення.

Якщо дотримано вищеназваних правил, то взаємодія між процесором, кеш пам'яттю та основною пам'яттю відбувається наступним чином.

Процесор формує адреси для взаємодії з основною пам'яттю і не враховує наявність кеш пам'яті. Він видає адресу комірки основної пам'яті і сигнали запису або зчитування. Контролер кеш пам'яті визначає, чи вміст блоку з даною коміркою знаходиться в кеш пам'я-

ті. Якщо так (попадання), то при зчитуванні з процесором взаємодіє лише кеш пам'ять, а при записі для забезпечення ідентичності вмісту блоків кеш і основної пам'яті може бути два варіанти. При першому варіанті здійснюється одночасний запис операнда в комірку з тією ж адресою в кеш пам'яті та в основній пам'яті. При другому варіанті одночасний запис не здійснюється, а використовується біт модифікації. Саме ж заміщення вмісту даної комірки основної пам'яті відбувається пізніше, при заміщенні в кеш пам'яті вмісту даного блоку Оскільки питання забезпечення ідентичності вмісту блоків кеш пам яті та основної пам яті є важливим, в наступному пункті воно буде розглянуто детальніше

Якщо ж вміст блоку з заданою коміркою відсутній в кеш пам'яті (промах), то при зчитуванні він поступає із основної пам'яті в кеш пам'ять, а слово з заданої комірки поступає прямо в процесор, або після пересилки вмісту всього блоку. Перший підхід ефективніший але складніший. При операції запису в цьому випадку слово записується прямо в основну пам'ять

Таким чином, кеш пам'ять перехоплює сигнали читання/запису, які процесор надсилає до основної пам'яті, а коли потрібно, то надає процесору копії даних, які тимчасово зберігає у власній робочій пам'яті. Якщо кеш пам'ять спроможна підмінити собою основну пам'ять (у понад 96-98 відсотків випадків), тоді вона за рахунок власних ресурсів задовольняє запит процесора. Процесор не пригальмовується і продовжує працювати на повній швидкості. Коли "підміна" основної пам'яті неможлива (менше від двох-чотирьох відсотків випадків), тоді кеш пам'ять залучає до роботи основну пам'ять, обмін з якою суттєво пригальмовує процесор

Усі задачі, пов'язані з перехопленням запитів від процесора до основної пам'яті, вирішує контролер кеш пам'яті, який є її складовою частиною. Другою частиною кеш пам'яті є невелика робоча пам'ять, де зберігаються копії вмісту блоків основної пам'яті, що брали участь в обслуговуванні останніх запитів процесора

Важливо, що вміст комірок основної пам'яті копіюється до кеш пам'яті разом із їхніми адресами. Саме ці адреси і дозволяють контролеру кеш пам'яті приймати рішення про спроможність задовольнити конкретний процесорний запит без залучення до обміну повільної основної пам'яті

10.2.3. Забезпечення ідентичності вмісту блоків кеш пам'яті і основної пам'яті

Як ми вже бачили вище, в процесі обчислень процесор не лише зчитує з кеш пам'яті наявну інформацію, але і записує нову, оновлюючи тим самим вміст блоків кеш пам яті, які є копіями вмісту відповідних блоків основної пам'яті. Виникає ситуація, коли вміст блоку кеш пам яті та відповідного блоку основної пам яті перестають співпадати, що створює проблему необхідності фіксування та усунення цього неспівпадіння шляхом оновлення вмісту комірок основної пам'яті Для подолання цієї проблеми в комп'ютерах з кеш пам яттю використовуються два методи оновлення вмісту блоків основної пам яті: метод наскрізного запису і метод зворотного запису

За методом наскрізного запису перш за все оновлюється вміст комірки основної пам'яті. Якщо в кеш пам яті існує копія вмісту цієї комірки, то вона також оновлюється. Якщо ж в кеш пам'яті відсутня потрібна копія, то, або з основної пам'яті в кеш пам'ять пересилається вміст блоку, в якому знаходиться оновлене слово (наскрізний запис з відображенням), або цього не робиться (наскрізний запис без відображення).

Метод наскрізного запису, про який вище вже згадувалось, передбачає одночасний запис операнда в комірку з тією ж адресою як кеш, так і основної пам'яті. Основна перевага методу наскрізного запису полягає в тому, що коли вміст деякого блоку в кеш пам яті має бути заміщений, то його можна не повертати в основну пам'ять, оскільки його копія там вже є. Метод достатньо простий в реалізації. На жаль, ефект від використання кеш пам'яті (скорочення часу доступу) стосовно операцій запису тут відсутній. Даний метод застосований в мікропроцесорах i486 фірми Intel.

Певний виграш дає модифікація методу наскрізного запису, відома як метод буферизованого наскрізного запису. Інформація спочатку записується в кеш пам ять і в спеціальний буфер, що працює за схемою FIFO. Запис в основну пам'ять проводиться вже з буфера, а процесор, не чекаючи її закінчення, може відразу ж продовжувати свою роботу. Звичайно, відповідна логіка керування повинна піклуватися про те, щоб своєчасно очищувати заповнений буфер. При використанні буферизації процесор повністю звільняється від роботи з основною пам'яттю

Згідно з методом зворотного запису, слово заноситься лише в кеш пам ять. Якщо вмісту потрібного блоку в кеш пам'яті немає, то він спочатку пересилається з основної пам яті, після чого запис все одно виконується виключно в кеш пам'ять. При заміщенні вмісту блоку його попередній вміст необхідно заздалегідь переслати у відповідне місце основної пам'яті. Для методу зворотного запису, на відміну від методу наскрізного запису, характерним є те, що при кожному зчитуванні з основної пам яті здійснюються два пересилання вмісту блоків між основною і кеш пам'яттю

Ефективність зворотного запису підвищується при використанні біта модифікації. Відповідно до методу зворотного запису з бітом модифікації, коли в якомусь блоці кеш пам яті проводиться заміщення вмісту, встановлюється пов'язаний з цим блоком біт модифікації (прапорець). При заміщенні вміст блоку з кеш пам яті переписується в основну пам'ять лише тоді, коли його біт модифікації встановлений в 1. Такий метод використовується, наприклад, в мікропроцесорах класу i486 і Pentium фірми Cyrix.

В середньому зворотний запис на 10 % ефективніший за наскрізний запис, але для його реалізації потрібні додаткові апаратні витрати. З іншого боку, практика показує, що операції запису складають менше ЗО % від загальної кількості звернень до пам яті. Таким чином, відмінність за швидкодією між розглянутими методами невелика

Крім розглянутих вище методів забезпечення ідентичності вмісту блоків кеш пам яті та основної пам'яті існують й інші методи, пов'язані з тим, що пристрої введення/виведення можуть безпосередньо обмінюватися інформацією з основною пам яттю без участі процесора, а значить і кеш пам'яті. Тобто в основну пам ять з пристрою введення, минувши процесор, заноситься нова інформація і невірною стає копія, що зберігається в кеш пам'яті. Запобігти подібній неузгодженості дозволяють два підходи: забезпечити введення будь-якої інформації в основну пам'ять з проведенням відповідних змін в кеш пам яті, або здійснювати доступ до основної пам'яті лише через кеш пам'ять

10.2.4. Функція відображення

10.2.4.1. Типи функцій відображення

Відображенням називають відповідність між вмістом блоків кеш пам'яті та блоків основної пам'яті.

Існує три основних типи відображення: повністю асоціативне, пряме та частково-асоціативне. Рис. 10.7 дає пояснення кожному типу відображення. При асоціативному відображенні вміст будь-якого блоку основної пам'яті може знаходитись в будь-якому блоці кеш пам'яті. Тобто вміст 12-го блоку основної пам'яті, наведеної на рис. 10.7, може бути записаний до будь-якого з 8 блоків кеш пам'яті. При прямому відображенні вміст кожного блоку основної пам'яті можна копіювати не до будь-якого, а лише до наперед визначеного блоку кеш пам'яті. Наприклад, це може бути відображення вмісту блоку К основної пам'яті в блок (К)тоар кеш пам'яті, де р - кількість блоків в кеш пам'яті. Тобто 12 блок основної пам'яті на рис. 10.7 може бути записаний лише до 4 блоку кеш пам'яті, так як (12)тосI8 = 4. При частково-асоціативному відображенні вміст кожного блоку основної пам'яті можна копіювати не до будь-якого, а до кількох наперед визначених блоків кеш пам'яті. Наприклад, це може бути відображення вмісту блоку К основної пам'яті в довільний блок сектора (К)тосI5 кеш пам'яті, де в - кількість секторів в кеш пам'яті. Тобто вміст 12-го блоку основної пам'яті на рис. 10.7 може бути записаний лише до блоку 0 або блоку 1 сектора 0 кеш пам'яті, так як (12)тосI4 = 0.

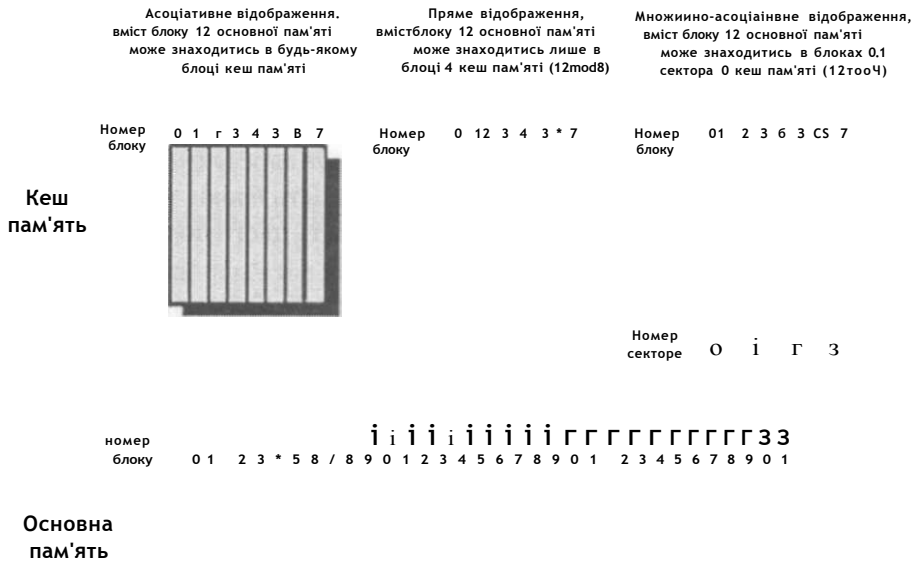


Рис. 10.7. Три основних типи відображення

Для детальнішого пояснення принципів обміну вмістом блоків основної пам'яті та кеш пам'яті при використанні кожного типу відображення приймемо, що основна пам'ять має ємність 1 ГБ, а кеш пам'ять має ємність 16 КБ, вони поділені на блоки ємністю по 32 Б, тобто в основній пам'яті є 2^{25} блоків, а в кеш пам'яті - 512 блоків.

10.2.4.2. Повністю асоціативне відображення

Взаємодія основної пам'яті з кеш пам'яттю з використанням повністю асоціативного відображення показана на рис. 10.8 а.

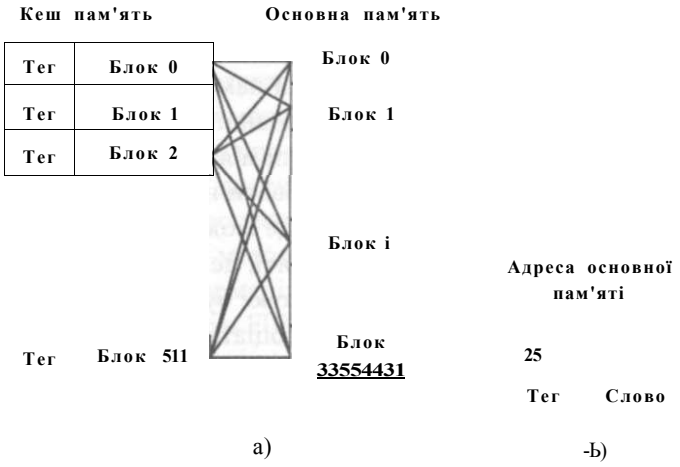


Рис. 10.8. Взаємодія основної пам'яті з кеш пам'яттю з використанням повністю асоціативного відображення а) та формат адреси основної пам'яті Ь)

Відповідно до цього типу відображення вміст будь-якого блоку основної пам'яті може знаходитись в будь-якому блоці кеш пам'яті. При використанні асоціативного відображення адреса, яка поступає з процесора до основної пам'яті, ділиться на два поля: поле тега і поле слова (рис. 10.8 Ь). Вміст поля тега вказує адресу блоку в основній пам'яті. Розрядність цього поля рівна $k = \log_2 K$, де К - кількість блоків в основній пам'яті. А вміст поля слова вказує адресу слова в блоці. Розрядність цього поля рівна $m = \log_2 M$, де М - кількість слів в блоці.

При даному способі відображення кожен рядок кеш пам'яті вміщує наступну інформацію: тег, який вказує вміст якого блоку основної пам'яті переписано до даного блоку кеш пам'яті, розряд достовірності (valid bit V), який вказує, чи вміст даного блоку кеш пам'яті дійсно належить вказаному тегом блоку основної пам'яті, розряд модифікації (dirty bit D), який інформує про внесення змін до вмісту блоку кеш пам'яті, а також вміст вказаного тегом блоку основної пам'яті. Використання розряду модифікації пов'язано з тим, що вміст блоку в кеш пам'яті може змінюватись. Тому, якщо він не змінився, немає необхідності переписувати його назад до основної пам'яті, так як там вже є копія. Якщо ж змінився, потрібно переписати. Тому для фіксації змін в блоках кеш пам'яті до рядка кеш пам'яті і вводиться розряд модифікації D, який дозволяє суттєво зменшити час обміну. Коли процесору потрібний операнд із блоку з певною адресою, контролер кеш пам'яті повинен шукати його шляхом порівняння відповідних розрядів адреси з тегами всіх блоків кеш пам'яті. Якщо таке порівняння здійснювати послідовно, то кеш пам'ять втрачає зміст через низьку швидкодію. Доцільніше таке порівняння робити одночасно зі всіма тегами, як це показано на рис. 10.9. В цьому випадку кожен блок кеш пам'яті повинен містити свій компаратор, тобто це буде пам'ять з асоціативним доступом. При наявності в кеш пам'яті відповідного тега, та при одиничному значенні розряду достовірності, кеш пам'ять видає сигнал підтвердження попадання та надає доступ до відповідного блоку.

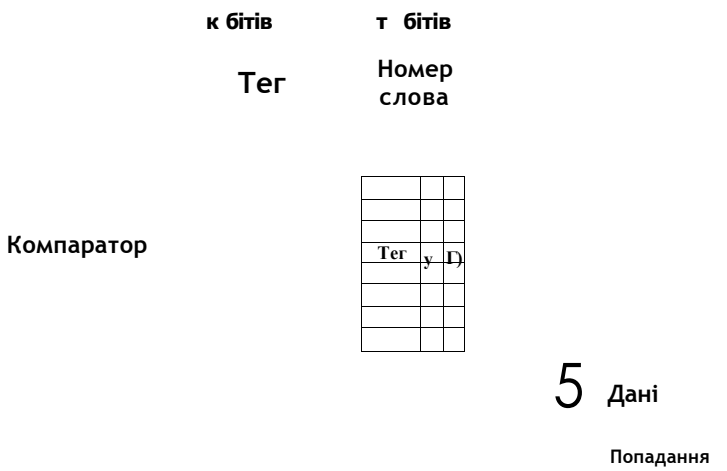


Рис. 10.9. Ідентифікація даних в кеш пам'яті з асоціативним доступом

При побудові кеш пам'яті малої ємності використання пам'яті з асоціативним доступом є доцільним, оскільки дозволяє швидко ідентифікувати потрібну інформацію та реалізувати найефективніший спосіб відображення. При побудові кеш пам'яті великої ємності використання пам'яті з асоціативним доступом стає проблематичним через складність реалізації та великі габарити.

70.2.4.3. *Пряме відображення*

Взаємодія основної пам'яті з кеш пам'яттю з використанням прямого відображення показана на рис. 10.10а. Тут вміст кожного блоку основної пам'яті можна копіювати лише до наперед визначеного блоку кеш пам'яті Одна з можливих реалізацій використовує розрядне відображення. Тут додатково до того, як це було зроблено для асоціативного відображення, к-розрядне поле адреси блоку в основній пам'яті розбивається на дві частини: поле тега г та поле номера блоку в (рис. ІО.ІОЬ). Тим самим основна пам'ять ділиться на секції, за кожною з яких закріплюється відповідна бирка (або ознака), яку називають тегом. Кількість Рч секцій (а значить і кількість тегів) рівна відношенню ємності основної пам'яті до ємності кеш пам'яті, або, що є тим самим, відношенню кількості К блоків в основній пам'яті, до кількості 8 блоків в кеш пам'яті. Розрядність поля тега визначається з виразу г = ^{2} К / 5 = ^{2}(2^{к}/2^{5}) = к - в. Кількість Б блоків в кожній секції рівна кількості блоків в кеш пам'яті. Тобто ємність секції рівна ємності кеш пам'яті.

Наприклад, якщо в адресі блоку основної пам'яті присутні к розрядів, то молодші п з них вибирають, в який блок кеш пам'яті може копіюватись вміст блоку основної пам'яті. Отже, вміст всіх блоків з однаковими молодшими адресами потрапляє в один і той самий блок кеш пам'яті. Інші г=к-в розрядів адреси, що залишилися, використовуються як адресний тег, який використовується для порівняння з адресою, виданою процесором з метою пошуку номера секції в основній пам'яті.

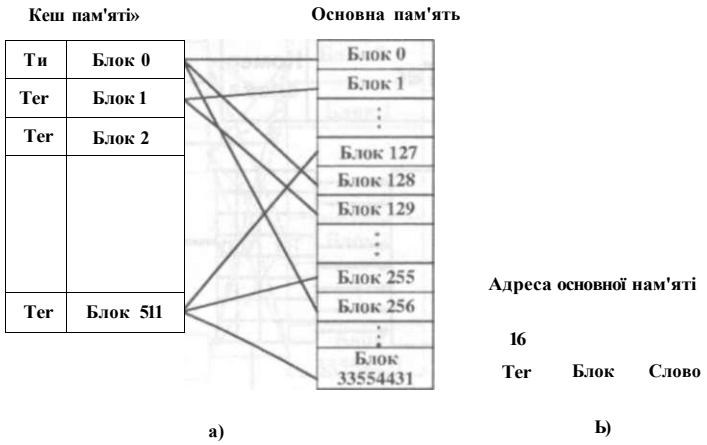


Рис. 10.10. Взаємодія основної пам'яті з кеш пам'яттю з використанням прямого відображення а) та формат адреси основної пам'яті Ь)

Для наведеної вище умови, коли що основна пам ять має ємність 1 ГБ, а кеш пам ять - 16 КБ, вони поділені на блоки ємністю по 32 Б, тобто кількість блоків в основній пам'яті рівна 2^{25}, а в кеш пам'яті - 2^{9}, маємо: повна розрядність адреси п = ЗО, причому поле адреси блоків основної пам'яті займає k = 25 розрядів, поле номера тега (тобто номера секції в основній пам'яті) г = 16 розрядів, поле номера блоку в секції (в кеш пам'яті) п = 9 розрядів, поле адреси слова в блоці m = 5 розрядів (рис. 10.10b). Зауважимо, що процесор не сприймає структурної інтерпретації адреси, наведеної на рис. 10.10b. Таку інтерпретацію адресі надає лише контролер кеш пам'яті, коли перехоплює цю адресу, що призначена основній пам'яті (в гарвардській архітектурі це може бути як пам'ять даних, так і пам'ять команд). Контролер кеш пам'яті за допомогою дев'яти середніх розрядів слова адреси звертається до визначеного цими розрядами блоку власної пам'яті. Зрозуміло, що шуканий в такий спосіб блок завжди присутній в кеш пам'яті. Адже кеш пам'ять вміщує 512 блоків. Але вміст віднайденого блоку кеш пам'яті може бути копією не одного, а одного з декількох дозволених на копіювання блоків основної пам'яті. Наприклад, до нульового блоку кеш пам'яті дозволено копіювати вміст наступних блоків основної пам'яті: 0, 128, 256, 512 і т. д. Усього до кожного блоку кеш пам'яті можна скопіювати вміст одного з 2^{16} = 65536 блоків основної пам'яті, оскільки ємність основної пам'яті в 65536 разів перевищує ємність кеш пам'яті. Постає завдання визначити, чи є поточне наповнення вказаного блоку кеш пам'яті відповідним до запиту процесора. Завдання розв'язують за допомогою вмісту старших 16 бітів адреси основної пам'яті, які утворюють поле із назвою Ter

Основною перевагою тут є те, що кеш пам'ять має структуру звичайної пам'яті з довільним доступом, тому потрібен лише один компаратор (рис. 10.11).

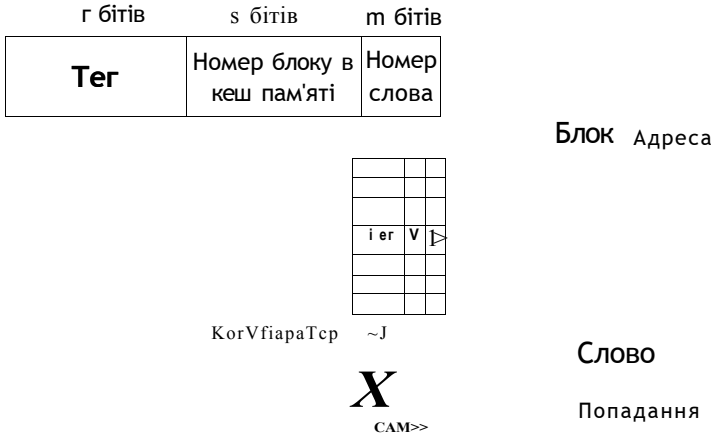


Рис. 10.11. Ідентифікація даних в кеш пам'яті з довільним доступом

При даному способі відображення кожен рядок кеш пам'яті вміщує наступну інформацію: тег, який вказує вміст блоку з якої секції основної пам'яті переписано до даного блоку кеш пам'яті, розряд достовірності (valid bit V), який вказує чи вміст даного блоку кеш пам'яті дійсно належить блоку основної пам'яті, вказаному s розрядами поля адреси, розряд модифікації (dirty bit D), який інформує про внесення змін до вмісту блоку кеш пам'яті, а також вміст блоку основної пам'яті, вказаного s розрядами поля адреси. Коли процесору потрібний операнд із блоку з певною адресою, контролер кеш пам'яті вибирає з пам'яті тегів відповідний номеру блоку в кеш пам'яті тег, та порівнює його з відповідними г розрядами адреси, в яких вказано тег. При наявності в кеш пам'яті відповідного тега та при одиничному значенні розряду достовірності кеш пам'ять видає сигнал підтвердження попадання та надає доступ до відповідного блоку

Отже пам'ять такого типу може мати велику ємність. Основний недолік закріплення за кожним блоком однієї секції основної пам'яті одного блоку в кеш пам'яті, тому в кеш пам'яті може знаходитись обмежене число вмісту блоків основної пам'яті Якщо процесор почергово звертається до даних з двох різних секцій основної пам'яті, які відображаються на той же блок кеш пам'яті, то ймовірність попадання буде низькою. Як наслідок, збільшується кількість промахів і, відповідно, час звернення до пам'яті

10.2.4.4. Частково-асоціативне відображення

Взаємодія основної пам'яті з кеш пам'яттю з використанням частково-асоціативного відображення показана на рис. 10.12а. Тут послідовно розміщені блоки кеш пам'яті об'єднуються в сектори рівної ємності. Зазвичай один сектор містить 2, 4, 8 блоків. Основна пам'ять ділиться на секції, за кожною з яких закріплюється тег, що вказує номер секції. Кількість таких секцій рівна відношенню ємності основної пам'яті до ємності секції. При цьому ємність секції рівна відношенню ємності кеш пам'яті до кількості блоків в секторі кеш пам'яті. Тобто кількість блоків в секції основної пам'яті рівна кількості блоків в кеш пам'яті, поділеній на кількість блоків в секторі кеш пам'яті. Таким чином, в такій реалізації, як і при прямому відображенні, використовується схема вибору сектора в кеш пам'яті за молодшими розрядами адреси блоку основної пам'яті, але в такому секторі кеш пам'яті знаходиться одразу вміст декількох блоків основної пам'яті, вибір між якими здійсню-

ється на основі асоціативного пошуку. Тут так само, як це було зроблено для прямого відображення, к-розрядне поле адреси блоку в основній пам'яті розбивається на дві частини: поле тега г та поле номера блоку Б (рис. 10.12Ь), причому тег вказує номер секції основної пам'яті, а в полі номера блоку вказується номер блоку в секції основної пам'яті та відповідно в кеш пам'яті.

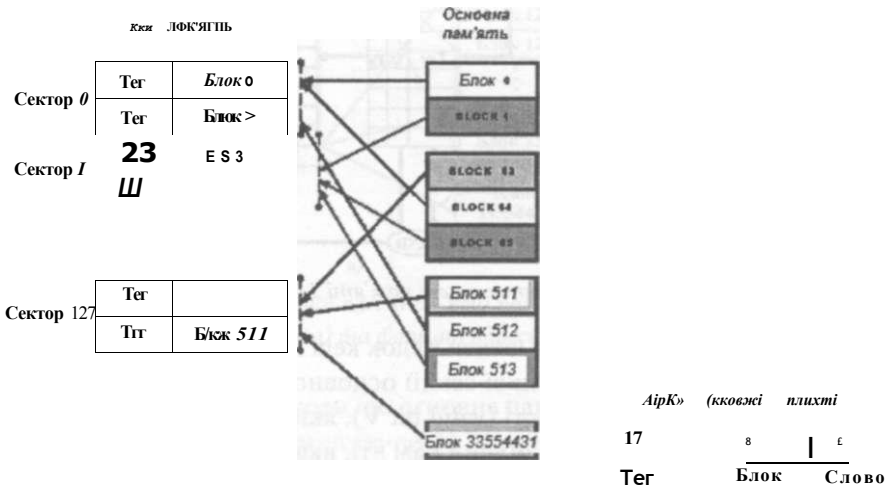


Рис. 10.12. Взаємодія основної пам'яті з кеш пам'яттю з використанням частково-асоціативного відображення а), коли кількість блоків в секторі кеш пам'яті рівна 2, та формат адреси основної пам'яті Ь)

Для наведеного вище прикладу, коли кількість блоків в основній пам'яті рівна 2^{25}, а в кеш пам'яті - 2^{9}, причому кількість блоків в секторі кеш пам'яті рівна 2, маємо: повна розрядність адреси п = ЗО, причому поле адреси блоків основної пам'яті займає k = 25 розрядів, поле номера тега (тобто номера секції в основній пам'яті) г = 17 розрядів, поле номера блоку в секції (в кеш пам'яті) п = 8 розрядів, поле адреси слова в блоці m = 5 розрядів (рис. 10.12b).

Контролер кеш пам'яті за допомогою восьми середніх розрядів слова адреси звертається до визначеного цими розрядами блоку власної пам'яті. Зрозуміло, що шуканий в такий спосіб блок завжди присутній в кеш пам'яті. Адже кеш пам'ять вміщує 512 блоків. Але вміст віднайденого блоку кеш пам'яті може бути копією не одного, а одного з декількох дозволених на копіювання блоків основної пам'яті. Наприклад, до нульового блоку кеш пам'яті дозволено копіювати вміст наступних блоків основної пам'яті: 0, 64, 128, 192, 256 і т. д. Усього до кожного блоку кеш пам'яті можна скопіювати вміст одного з 2^{17}= 131072 блоків основної пам'яті, оскільки ємність основної пам'яті в 131072 разів перевищує ємність кеш пам'яті. Для того, щоб визначити, чи є поточне наповнення вказаного блоку кеш пам'яті відповідним до запиту процесора, використовують вміст старших 17 бітів адреси основної пам'яті.

Потрібно відзначити, що якраз частково-асоціативне відображення найчастіше використовується в сучасних комп'ютерах, причому кількість блоків в одній секції кеш пам'яті зазвичай не перевищує чотирьох. Основною перевагою тут є те, що кеш пам'ять поєднує переваги пам'яті з довільним та з асоціативним доступом (рис. 10.13).

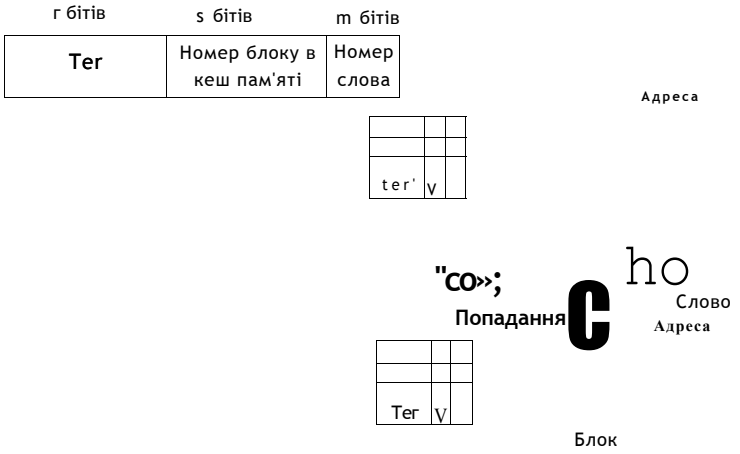


Рис. *10.13. Ідентифікація даних в кеш пам'яті з частково-асоціативним доступом*

При даному способі відображення кожен рядок кеш пам'яті вміщує наступну інформацію: тег, який вказує вміст блоку якої секції основної пам'яті переписано до даного блоку кеш пам'яті, розряд достовірності (valid bit V), який вказує, чи вміст даного блоку кеш пам'яті дійсно належить блоку основної пам'яті, вказаному s розрядами поля адреси, розряд модифікації (dirty bit D), який інформує про внесення змін до вмісту блоку кеш пам'яті, а також вміст блоку основної пам'яті, вказаного s розрядами поля адреси. Коли процесору потрібний операнд із блоку з певною адресою, контролер кеш пам'яті вибирає з секторів пам'яті тегів відповідні номеру блоку в кеш пам'яті теги, та порівнює їх з відповідними г розрядами адреси, в яких вказано тег. При наявності в кеш пам'яті відповідного тега, та при одиничному значенні розряду достовірності, кеш пам'ять видає сигнал підтвердження попадання та надає доступ до відповідного блоку.

10.2.5. Порядок заміщення блоків в кеш пам'яті з асоціативним відображенням

Для запису з основної пам'яті до кеш пам'яті вмісту нового блоку в ній потрібно знайти блок, вміст якого має бути заміщений. При використанні прямого відображення такий блок є наперед визначеним. При використанні повністю асоціативного та частково-асоціативного відображення, коли в деякий блок в кеш пам'яті може бути записаний вміст довільного блоку, або деякої множини блоків основної пам'яті, потрібно використати відповідний алгоритм заміщення.

Основна мета стратегії заміщення - утримувати в кеш пам'яті вміст блоків основної пам'яті, до яких найбільш імовірні звернення в найближчому майбутньому, і заміщувати вміст блоків, доступ до яких відбудеться пізніше, або взагалі не відбудеться. Очевидно, що оптимальним буде алгоритм, який заміщує вміст того блоку, звернення до якого в майбутньому відбудеться пізніше, ніж до будь-якого іншого блоку кеш пам'яті. На жаль, такий прогноз здійснити практично неможливо, тому використовують ряд стратегій, кожна з яких вигідніша в відповідному конкретному випадку. При цьому, для досягнення високої швидкості заміщення, алгоритми заміщення реалізуються апаратними засобами.

Серед безлічі можливих алгоритмів заміщення найпоширенішими є чотири, що розглядаються далі у порядку зменшення їх відносної ефективності.

Найбільш ефективним є алгоритм заміщення LRU (Least Recently Used), який передбачає заміщення блоків з найдавнішим використанням, оскільки він базується на властивості часової локальності. За цим алгоритмом заміщується вміст того блоку кеш пам'яті, до якого найдовше не було звернення. Найвідоміші два способи апаратурної реалізації цього алгоритму. У першому з них за кожним блоком кеш пам'яті закріплюють лічильник. До вмісту всіх лічильників через певні інтервали часу додається одиниця. При зверненні до блоку його лічильник встановлюється в нульовий стан. Таким чином, найбільше число буде в лічильнику того блоку, до якого найдовше не було звернень, і вміст цього блоку є першим кандидатом на заміщення. Другий спосіб реалізується за допомогою черги, куди у порядку заповнення блоків кеш пам'яті заносяться посилання на ці блоки. При кожному зверненні до блоку посилання на нього переміщується в кінець черги. В результаті першим в черзі кожного разу опиняється посилання на блок, до якого найдовше не було звернень. Вміст саме цього блоку перш за все і заміщується.

Інший можливий алгоритм заміщення - алгоритм, що працює за принципом "перший увійшов, перший вийшов" (FIFO - First In First Out). Тут заміщується вміст блоку, що найдовше знаходився в кеш пам'яті. Алгоритм легко реалізується за допомогою розглянутої раніше черги, з тією лише різницею, що після звернення до блоку положення відповідного посилання в черзі не змінюється.

Ще один алгоритм - заміна вмісту блоку, що найменше використовувався (LFU - Least Frequently Used). Заміщується вміст того блоку в кеш пам'яті, до якого було менше всього звернень. Принцип можна реалізувати, пов'язавши кожен блок з лічильником звернень, до вмісту якого після кожного звернення додавати одиницю. Головним претендентом на заміщення є вміст блоку, лічильник якого містить найменше число.

Простий алгоритм заміщення - довільний вибір блоку для заміни його вмісту. Блок, вміст якого замінюється, вибирається випадковим чином. Це може бути реалізовано, наприклад, за допомогою лічильника, вміст якого збільшується на одиницю з кожним тактовим імпульсом, незалежно від того, що мало місце - попадання чи промах. Значення в лічильнику визначає блок, вміст якого буде замінено в повністю асоціативній кеш пам'яті, або воно визначає блок в межах сектора, вміст якого буде замінено в частково-асоціативній кеш пам'яті. Даний алгоритм використовується вкрай рідко.

Частіше попереднього використовують алгоритм випадкового заміщення вмісту блоків кеш пам'яті за значенням лічильника випадкових чисел, оскільки за ефективністю він є близьким до алгоритму LRU та простішим за нього в реалізації.

10.2.6. Підвищення ефективності кеш пам'яті

Характеристики, які використовують при оцінці ефективності ієрархічної організації пам'яті, підходять і для оцінки ефективності кеш пам'яті. До цих характеристик належать наступні:

- коефіцієнт попадань - відношення числа попадань до загального числа звернень процесора до основної пам'яті, де під попаданням розуміється виявлення в кеш пам'яті потрібної інформації, при зверненні до основної пам'яті;
- коефіцієнт промахів - відношення числа промахів до загального числа звернень процесора до основної пам'яті, де під промахом розуміється відсутність в кеш пам'яті

потрібної інформації, при зверненні до основної пам'яті. Якщо позначити коефіцієнт попадань через k_h, а коефіцієнт промахів через k , то залежність між ними можна виразити наступною формулою: $k_m = 1 - k_h$;

- час звернення при попаданні - час, необхідний для пошуку потрібної інформації в кеш пам'яті (включаючи з'ясування, чи є звернення попаданням), плюс час на фактичне зчитування даних;
- втрати на промах - час, потрібний для заміни блоку в кеш пам'яті на блок з потрібними даними, розташований в основній пам'яті;
- середній час доступу до основної пам'яті, який визначається з виразу:

$t_{av} = t_h + k_m t_p$, де t_h - час звернення при попаданні, t - втрати на промах.

Таким чином, чим менше промахів, тим вища ефективність використання кеш пам'яті. Розглянемо типи промахів до кеш пам'яті та підходи до зменшення їх кількості й пов'язаних з ними втрат. Існує три типи промахів: обов'язкові, ємнісні та конфліктні.

Обов'язкові промахи - це промахи, які виникають при початковому зверненні до основної пам'яті. Зрозуміло, що перший доступ до кеш пам'яті обов'язково буде промахом. Для зменшення кількості цього типу промахів потрібно збільшувати розмір блоків в кеш пам'яті.

Ємнісні промахи пов'язані з обмеженим розміром кеш пам'яті. Для зменшення кількості цього типу промахів потрібно збільшувати розмір кеш пам'яті.

Конфліктні промахи пов'язані з ефективністю алгоритму заміщення блоків в кеш пам'яті. Для зменшення кількості цього типу промахів потрібно збільшувати асоціативність кеш пам'яті, аж до використання для її реалізації повністю асоціативної пам'яті.

На рис. 10.14 наведено залежність величини коефіцієнта промахів від ємності кеш пам'яті та від кількості блоків в одному секторі кеш пам'яті з використанням частково-асоціативного відображення при виконанні набору тестових програм SPEC 92.

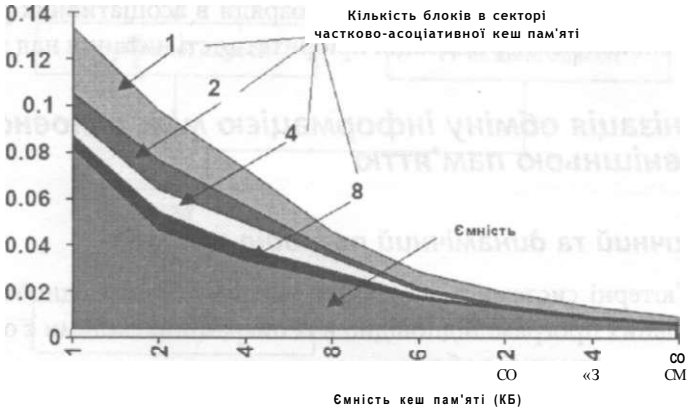


Рис. 10.14. Залежність значення коефіцієнта промахів від ємності та від кількості блоків в одному секторі частково-асоціативної кеш пам'яті

Таким чином, збільшення розміру блоку кеш пам'яті понижує кількість обов'язкових промахів, але збільшує втрати на промах, через зростання часу пересилання блоку інформації. Крім того, якщо ємність кеш пам'яті невелика, то збільшення розміру блоку збільшує кількість конфліктних промахів через зменшення кількості блоків та секторів.

Подібним чином збільшення ємності кеш пам'яті зменшує кількість ємнісних промахів, але збільшує час звернення при попаданні.

Для зменшення кількості конфліктних промахів потрібно збільшувати асоціативність кеш пам'яті, але, як показує практика, більша 4 кількість блоків в одному секторі кеш пам'яті з використанням частково-асоціативного відображення збільшує затрати обладнання без видимого виграшу, та збільшує час звернення при попаданні.

Часто для суттєвого зменшення кількості конфліктних промахів, в першу чергу для кеш пам'яті з прямим відображенням, використовують додаткову кеш пам'ять малого розміру (8-16 блоків), в якій зберігаються останні заміщені в кеш пам'яті блоки.

Для зменшення значення коефіцієнта промахів використовується попередня вибірка з основної пам'яті блоків, до яких може бути звернення. При цьому можуть застосовуватись як апаратні, так і програмні засоби попередньої вибірки. При використанні апаратних засобів зазвичай при промаху додатково до вибірки відсутнього в кеш пам'яті блоку вибирається наступний, або кілька наступних блоків. Це дає виграш, оскільки процесор не чекає на попередньо вибраний блок. Такий підхід може бути використаний як до кеш пам'яті даних, так і команд, але є більш ефективним для кеш пам'яті команд. При використанні програмних засобів до програми, або до компілятора, додаються спеціальні команди попередньої вибірки.

Для зменшення значення коефіцієнта промахів використовуються також спеціальні підходи по оптимізації компіляторів, зокрема об'єднання даних та команд у блоки, об'єднання циклів, черговість циклів, злиття масивів і т. д.

В останніх комп'ютерах для підвищення ефективності кеш пам'яті вводяться більш складні механізми її функціонування за рахунок ускладнення контролера та організації внутрішньої пам'яті. Це, зокрема, створення так званої неблокуючої кеш пам'яті, яка при наявності промаху продовжує взаємодію з процесором, використовуючи механізм невпорядкованого виконання або спеціальні розряди в асоціативних регістрах, будучи при цьому багато блоковою, та реалізація пріоритетності читання над записом.

70.3. *Організація обміну інформацією між основною та зовнішньою пам'яттю*

10.3.1. Статичний та динамічний розподіл пам'яті

Сучасні комп'ютерні системи є багатопрограмними. В них одночасно виконується велика кількість різних програм. Відповідно й їх операційні системи є орієнтованими на забезпечення багатопрограмної роботи.

Через обмежену ємність основної пам'яті, а також розміщення в ній, крім програм користувача, програм операційної системи, всі програми в багатопрограмних комп'ютерних системах не можуть розміщуватись в основній пам'яті.

Але в цьому немає великої необхідності, так як в певний момент часу виконується не вся програма, а певна її частина. Тому основна пам'ять використовується для зберігання активних частин виконуваних програм, а решта інформації зберігається в зовнішній пам'яті. При цьому може виникнути необхідність заміни тих частин виконуваних програм, які стали неактивними. Звідси випливає потреба розподілу пам'яті між програмами.

Якщо вся необхідна основна пам'ять для програм призначається до початку їх виконання - це статичний розподіл пам'яті. Цей тип розподілу основної пам'яті використовувався в перших комп'ютерах. Кожний програміст наперед замовляв потрібну йому ємність основної пам'яті. Зрозуміло, що статичний розподіл основної пам'яті не є досконалим. З одного боку, неефективно використовується пам'ять, так як важко наперед передбачити потрібну ємність основної пам'яті, а з іншого боку, на програміста покладається проблема заміни інформації в основній пам'яті.

В сучасних багатопрограмних комп'ютерних системах використовується динамічний розподіл пам'яті, відповідно до якого основна пам'ять розподіляється між програмами під час їх виконання. Для цього при підготовці цільових програм використовуються умовні адреси, а в процесі виконання програми комп'ютер перетворює умовні адреси в виконавчі, виділяючи програмі місце в пам'яті. Найчастіше це здійснюється з використанням апаратних засобів, щоб прискорити звернення до пам'яті. Існує декілька способів динамічного розподілу пам'яті. Далі розглянемо два методи динамічного розподілу основної пам'яті: за допомогою базових адрес та сторінкової організації.

10.3.2. Розподіл основної пам'яті за допомогою базових адрес

Відповідно до методу динамічного розподілу пам'яті за допомогою базових адрес кожній програмі виділяється деяка область основної пам'яті, і, крім того, програма пишеться з використанням умовних адрес. Для цього кожній програмі ставиться у відповідність базова адреса, що встановлюється операційною системою. При всіх зверненнях до пам'яті базова адреса додається на суматорі СМ (рис. 10.15) до умовної адреси, визначеної програмою, утворюючи виконавчу адресу пам'яті.

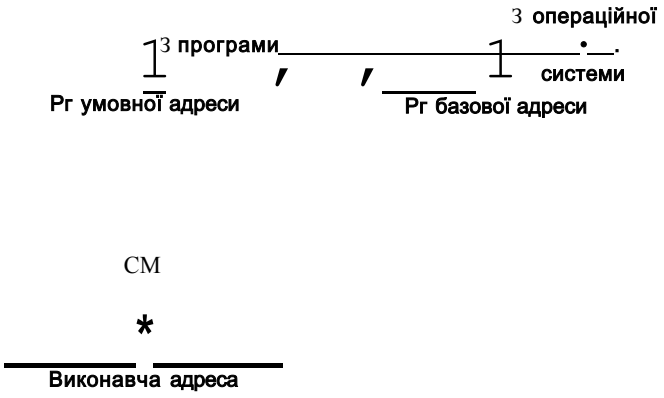


Рис. 10.15. Розподіл пам'яті за допомогою базових адрес

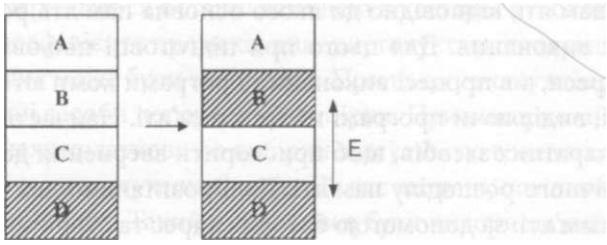
Як тільки деяка програма стає активною, операційна система знаходить для неї місце в пам'яті і встановлює в регістрі базової адреси відповідне число (базову адресу).

Недоліки розподілу пам'яті за допомогою базових адрес:

- необхідність розміщувати програму в послідовних комірках основної пам'яті, що призводить до неефективного її використання;
- необхідність вводити програми в основну пам'ять повністю (або ті її частини, в яких використовується одна базова адреса);

• фрагментація пам'яті, тобто її. поділ на використовувані і не використовувані частини (іншими словами - наявність в пам'яті дірок).

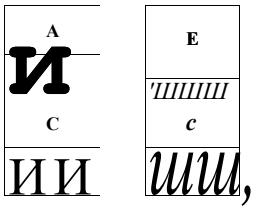
Як приклад на рис. 10.16а наведено розподіл основної пам'яті між трьома програмами А, В, С в деякий момент часу. Після завершення виконання програми В (рис. Ю.ІбЬ) в пам'яті вивільнилося місце, яке разом з областю пам'яті Б (вільні області В та Б виділені штриховими лініями) є більшим за розміром, ніж потрібно програмі Е, однак ця програма не може бути записаною до основної пам'яті, оскільки вона є більшою за кожну з областей В та Е, які розміщені в пам'яті не підряд.



а) Ь)

Рис. 10.16. Області основної пам'яті В та О *є дірками, оскільки програма Е не вставляється на їх місце*

Виходом з цієї ситуації є, наприклад, переміщення програми із області С ближче до області А, що дозволяє вивільнити місце програмі Е, як це показано, наприклад, на рис. 10.17а.



а) Ь)

Рис. 10.17. Переміщення програми із області С а) та очікування місця для програми Е Ь)

Однак це пов'язано з втратами часу на переміщення програми, а часто і з необхідністю редагування програми і завантаження її заново, оскільки змінилися адреси.

Іншим варіантом є чекання, доки не звільниться область необхідної ємності, як це показано на рис. 10.17Ь, коли завершилось виконання програми А і з'явилось достатньо місця для програми Е. Виходить, що програма чекає своєї черги на завантаження до основної пам'яті, хоча є ділянки пам'яті великого розміру, але вони розміщені в різних областях основної пам'яті.

Таким чином, при використанні описаного способу розподілу в основній пам'яті завжди будуть наявні дірки, які в сумі можуть займати значну частину її ємності.

Відмічені недоліки методу динамічного розподілу пам'яті за допомогою базових адрес відсутні в другого методу динамічного розподілу пам'яті, а саме в віртуальній пам'яті з сторінковою організацією.

10.3.3. Віртуальна пам'ять

Принцип віртуальної пам'яті передбачає, що користувач при підготовці програми має справу не з багаторівневою фізичною основною пам'яттю, що є в комп'ютері і має фіксовану ємність, а з віртуальною (уявною) однорівневою пам'яттю, ємність якої дорівнює всьому адресному простору комп'ютера, визначеному розміром адрес в регістрі адреси.

Користувач розпоряджається всім адресним простором комп'ютера, незалежно від ємності основної пам'яті і вимог до неї з боку інших користувачів. На всіх етапах підготовки програма представлена в віртуальних (умовних) адресах. При виконанні програми віртуальні адреси перетворюються у фізичні. Користувач не знає, де знаходиться його програма - в основній чи в зовнішній пам'яті, і в якій області. Це встановлюється автоматично в ході обчислювального процесу, тобто шляхом динамічного розподілу пам'яті. При цьому, оскільки для виконання програми необхідно, щоб та частина активної інформації, на яку вказують віртуальні адреси, знаходилась в основній пам'яті, то в процесі виконання програми необхідно перетворювати віртуальні адреси в фізичні та переписувати активну інформацію до основної пам'яті.

За своєю суттю віртуалізація пам'яті - це спосіб реалізації ієрархічної організації пам'яті на рівні взаємодії основної та зовнішньої пам'яті.

Віртуальна організація пам'яті дозволяє здійснювати керування пам'яттю, коли виконується паралельно багато програм, з забезпеченням захисту даних та наданням в розпорядження кожної програми всього адресного простору комп'ютера.

При віртуальній організації пам'яті адреси, які формуються процесором, є віртуальними. Необхідним атрибутом віртуальної пам'яті є зовнішня пам'ять, яка здатна зберігати всю програму повністю.

Розглянемо приклад. Нехай процесор посилає до основної пам'яті віртуальну адресу 520 004 096, причому ємність основної пам'яті рівна 512МБ, а ємність зовнішньої пам'яті рівна 80 ГБ (рис. 10.18).

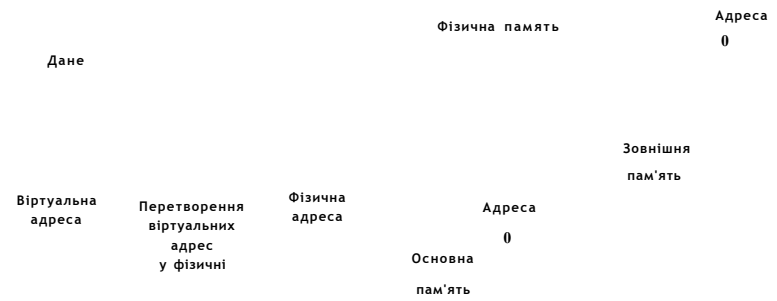


Рис. 10.18. Взаємодія процесора з основною пам'яттю при використанні віртуальної пам'яті

В комп'ютерах без застосування віртуальної пам'яті використовується пряма адресація. Тому звертання за вказаною адресою викликало б програмне переривання з таким повідомленням: "Адресована неіснуюча область пам'яті", оскільки вказана адреса виходить за межі основної пам'яті. В комп'ютерах з віртуальною пам'яттю виконується така послідовність кроків:

- віртуальна адреса перетворюється у фізичну;
- фізична адреса подається до фізичної пам'яті (основної або зовнішньої);
- в фізичній пам'яті знаходиться відповідна фізичній адресі комірка;
- до цієї комірки записується, або з неї зчитується, дане, причому, якщо ця комірка знаходиться в зовнішній пам'яті, то звернення до неї здійснюється через основну пам'ять шляхом перезапису інформації з зовнішньої пам'яті до основної.

10.3.4. Сторінкова організація пам'яті

10.3.4.1. Основні правила сторінкової організації пам'яті

Для спрощення перетворення віртуальних адресів в фізичні і позбавлення від фрагментації пам'яті, а крім того для прискорення обміну між основною та зовнішньою пам'яттю та вирішення питань захисту пам'яті, а також з метою реалізації принципу віртуальної пам'яті, використовується сторінкова організація пам'яті. Основні правила сторінкової організації пам'яті, які забезпечують ефективну взаємодію основної та зовнішньої пам'яті, є наступними:

- Програми представляють в віртуальних (умовних) адресах віртуальної одноріневої пам'яті, ємність якої дорівнює всьому адресному простору комп'ютера, визначеному розміром адрес в регістрі адреси.
- Віртуальна пам'ять та фізична (основна та зовнішня) пам'ять розбиваються на сторінки рівного об'єму.

Сторінка - це деяка множина послідовно розміщених комірок пам'яті.

Сторінкам присвоюються номери.

Кожна фізична сторінка здатна зберігати вміст однієї віртуальної сторінки.

Нумерація комірок в обох сторінках (віртуальних і фізичних) однакова.

Базовою порцією інформації, яка переміщується між основною та зовнішньою ам яттю, є вміст однієї сторінки.

Ідентичність вмісту відповідних сторінок основної пам'яті і зовнішньої пам'яті забезпечується використанням спеціальних методів оновлення вмісту сторінок основної пам'яті.

- Заміщення вмісту сторінок в основній пам'яті вмістом сторінок з зовнішньої пам'яті здійснюється за правилами, які називають алгоритмом заміщення.
- Взаємодія між сторінками віртуальної і фізичної пам'яті задається сторінковою таблицею.
- Між різними процесами (програмами) розподіляється фізична (а не віртуальна) пам'ять.
- Основна пам'ять динамічно розподіляється між різними процесами (програмами) посторінково.

Переваги сторінкової організації пам'яті:

- не вимагаються додаткові переміщення програм в пам'яті, пов'язані з потребою звільнення місця через фрагментацію;
- скорочується кількість пересилань між основною та зовнішньою пам'яттю, так як вміст сторінки завантажується лише при необхідності;

• зменшуються вимоги до ємності основної пам'яті, так як до неї завантажуються лише активні частини програми.

Про сторінкову організацію пам'яті говорять, що вона є прозорою, тобто програміст може працювати, не звертаючи уваги на факт її існування.

10.3.4.2. *Реалізація сторінкової організації пам'яті*

Відповідно до наведених вище правил сторінкової організації пам'яті спочатку потрібно розділити віртуальну та фізичну пам'ять на сторінки рівного об'єму, присвоїти цим сторінкам номери, та встановити відповідність між цими сторінками, як це для прикладу показано на рис. 10.19, де об'єм сторінки вибрано рівним 4КБ, а відповідні віртуальним сторінкам А, В, С, *В* фізичні сторінки розміщені як в основній, так і в зовнішній пам'яті. При цьому відразу необхідно зауважити, що довільній віртуальній сторінці може відповідати довільна фізична сторінка, тобто відповідність між сторінками є повністю асоціативною.

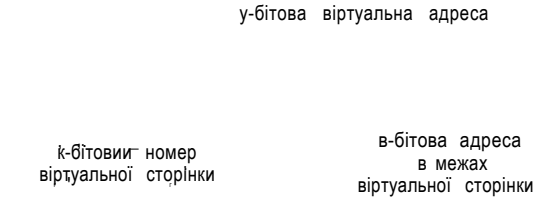


Рис. 10.19. Поділ віртуальної та фізичної пам'ятей

Нехай віртуальна пам'ять має ємність $У=2^{у}$ слів, та поділена на $2^{к}$ віртуальних сторінок, кожна з яких має об'єм $2^{в}$ слів, тобто розрядність віртуальної адреси **V** = к + в, де к - адреса (номер) віртуальної сторінки, в - номер слова в сторінці. Тоді формат віртуальної адреси буде мати вид, представлений на рис. 10.20.

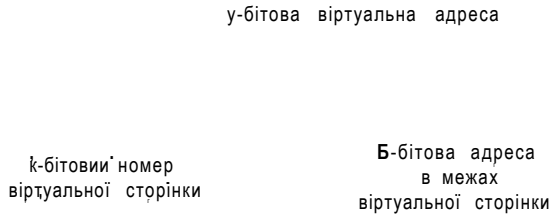


Рис. 10.20. Формат віртуальної адреси

Аналогічно, нехай основна пам'ять має ємність $М=2^{т}$ слів, та поділена на $2^{і}$ віртуальних сторінок, кожна з яких має об'єм $2^{і}$ слів, тобто розрядність адреси основної пам'яті т = 1 + Б, де 1 - адреса (номер) сторінки основної пам'яті, в - номер слова в сторінці. Тоді формат адреси основної пам'яті буде мати вид, представлений на рис. 10.21.

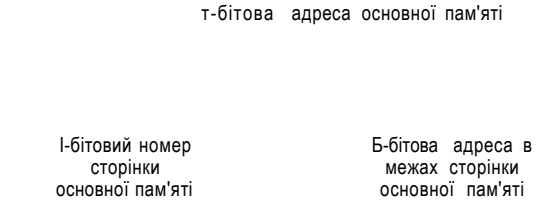


Рис. 10.21. Формат адреси основної пам'яті

Подібним чином, нехай зовнішня пам'ять має ємність $E=2^e$ слів, та поділена на 2^p віртуальних сторінок, кожна з яких має об'єм 2^r слів, тобто розрядність адреси основної пам'яті e = p + в, де p - адреса (номер) сторінки зовнішньої пам'яті, в - номер слова в сторінці. Тоді формат адреси зовнішньої пам'яті буде мати вид, представлений на рис. 10.22.

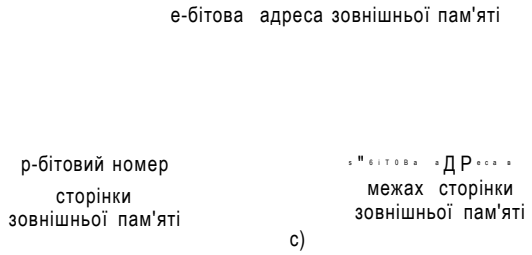


Рис. 10.22. Формат адреси зовнішньої пам'яті

Порядок перетворення адрес (номерів) віртуальних сторінок у фізичні задається у вигляді сторінкової таблиці, що вказує відповідність віртуальних і фізичних сторінок. Сторінкова таблиця формується, і при необхідності змінюється, операційною системою.

Процедура звернення до пам'яті така: номер віртуальної сторінки вибирається з віртуальної адреси і використовується як вхід в сторінкову таблицю, яка вказує номер фізичної сторінки (рис. 10.23).

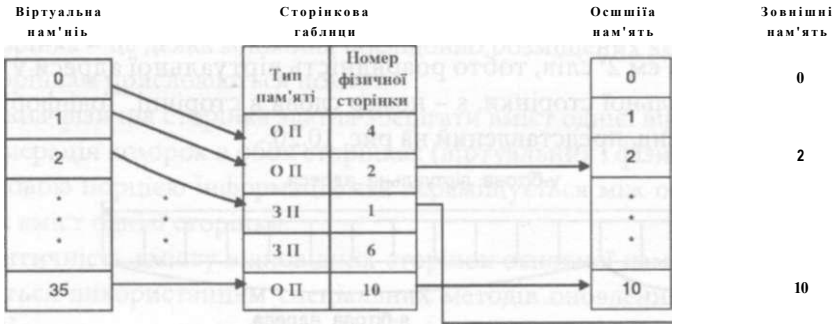


Рис. 10.23. Процедура звернення до пам'яті

Слово сторінкової таблиці складається з трьох основних полів (про допоміжні поля буде сказано пізніше):

- перше поле, яке займає один розряд, вказує тип пам'яті. Коли в цьому полі записано 0, це говорить про те, що сторінка відсутня в основній пам'яті. Коли записано 1 - це говорить про те, що сторінка наявна в основній пам'яті. Цей розряд зазвичай називають розрядом наявності (РН, або valid bit V).
- адреса сторінки в основній пам'яті (коли РН = 1). Якщо сторінка знаходиться в зовнішній пам'яті - це поле ігнорується.
- адреса сторінки в зовнішній пам'яті (коли РН = 0). За цією адресою вибирають вміст сторінки із зовнішньої пам'яті та загружають до основної пам'яті. Крім того, робиться відповідний запис в сторінковій таблиці, а після використання ця сторінка буде повернута назад в зовнішню пам'ять.

Перетворення віртуальної адреси у фізичну виконується так, як показано на рис. 10.24. Тобто номер віртуальної сторінки за допомогою сторінкової таблиці замінюється на номер фізичної сторінки, а номер слова в межах сторінки залишається без змін, причому залежно від значення розряду наявності (РН) формується або адреса основної пам'яті, або адреса зовнішньої пам'яті.

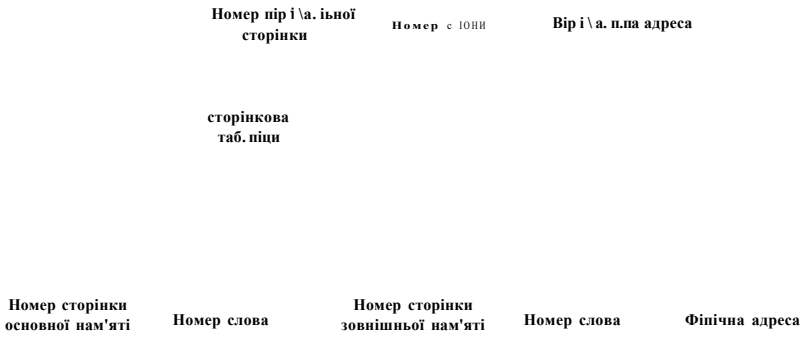


Рис. 10.24. Перетворення віртуальної адреси у фізичну

Вище було розглянуто питання розподілу пам'яті, коли виконується лише одна програма. Насправді сучасні комп'ютери є багатопрограмними. В цьому випадку при сторінковій організації пам'яті кожна програма має свою віртуальну пам'ять з адресацією від нуля і до максимальної адреси, яка допускається розрядною сіткою. Так як написання різних програм проводиться незалежно одна від одної, вони можуть виконуватись в одних і тих самих віртуальних адресах, які повинні належати до різних фізичних сторінок. Отже, сторінкова таблиця повинна враховувати також належність віртуальної сторінки різним програмам. Тобто номер фізичної сторінки є функцією двох змінних - номера віртуальної сторінки, та номера програми, як це показано на рис. 10.25. Іншими словами, для кожної програми має бути своя сторінкова таблиця.

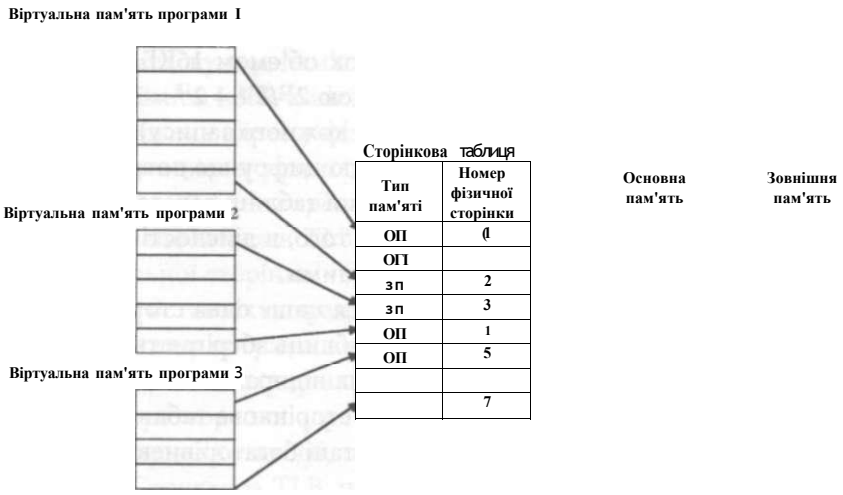


Рис. 10.25. Процедура звернення до пам'яті в багатопрограмних комп'ютерах

Перетворення віртуальної адреси у фізичну тут здійснюється відповідно до рис. 10.26.

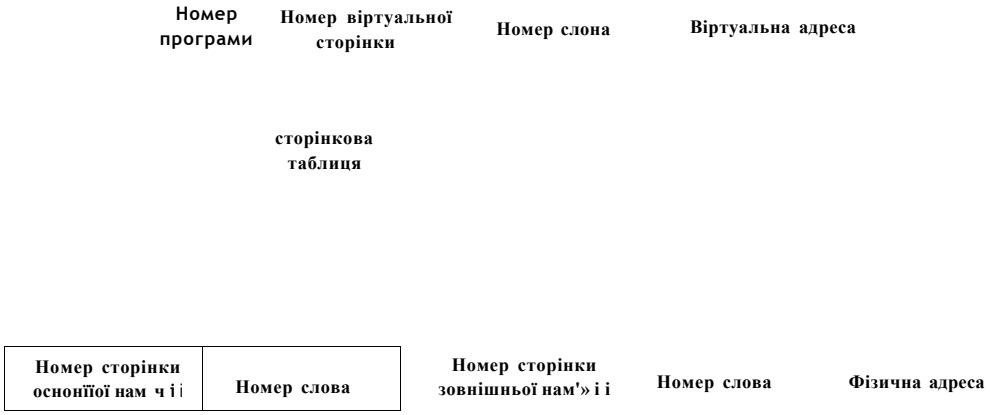


Рис. 10.26. Перетворення віртуальної адреси у фізичну в багатопрограмних комп'ютерах

10.3.4.3. Апаратна реалізація сторінкової таблиці

Вибір слова з пам'яті при сторінковій організації вимагає двох циклів звернення: спочатку до сторінкової таблиці, а потім до фізичної адреси пам'яті. Якщо сторінкова таблиця зберігається в основній пам'яті, це в два рази сповільнює вибірку слова. Тому часто сторінкову таблицю зберігають в швидкій пам'яті невеликої ємності. Склад і можливості цієї пам'яті залежать від вибраного правила побудови сторінкової таблиці:

- кожній віртуальній сторінці відповідає одна стрічка сторінкової таблиці. В стрічці є номер фізичної сторінки, який зберігає дану віртуальну сторінку;
- кожній фізичній сторінці відповідає одна стрічка сторінкової таблиці, що зберігається в даній фізичній сторінці.

В першій структурі входом в сторінкову таблицю є номер віртуальної сторінки і номер програми, як це показано на рис рис. 10.26, і вона будується як пам'ять з довільним доступом на основі мікросхем напівпровідникової пам'яті.

Кількість програм та віртуальних сторінок може бути досить великою. Наприклад, при використанні 32-розрядної адреси та сторінок об'ємом 16КБ, кількість записів в сторінковій таблиці для однієї програми буде рівною $2^{32}/2^{14} = 2^{18} = 256К$. Тоді об'єм сторінкової таблиці буде рівним (враховуючи формат кожного запису) 256К*4Б = 1 МБ, що складає 64 сторінки основної пам'яті, причому і цю цифру ще потрібно помножити на кількість програм. Тобто ємність пам'яті сторінкової таблиці такого типу також буде великою, причому ця пам'ять буде повільною. Крім того, в дійсності необхідно зберігати значно менше записів, так як записи є малорозрядними.

В деяких комп'ютерах в цій пам'яті зберігається лише одна сторінкова таблиця, яка належить активній програмі. Решта сторінкових таблиць зберігаються в основній пам'яті. При зміні циклу активності операційна система відправляє в основну пам'ять дану сторінкову таблицю і завантажує нову. Крім того, сторінкова таблиця також може бути поділена на сторінки, тобто можуть бути використані багаторівневі сторінкові таблиці, як це показано на рис. 10.27.

т

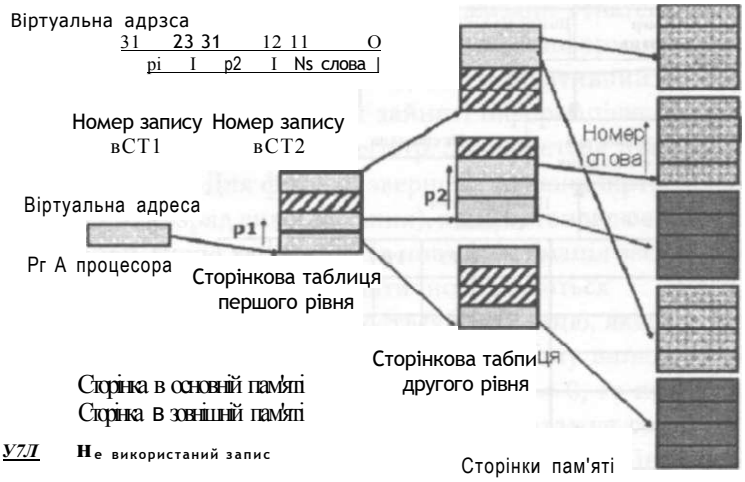


Рис. 10.27. Використання багаторівневих сторінкових таблиць

Тут спочатку іде звернення до сторінкової таблиці першого рівня, а з неї до сторінкової таблиці другого рівня. Як видно з рисунка, взамін однієї сторінкової таблиці об'ємом 2^{20} записів (кількість записів рівна кількості сторінок в пам'яті), тут потрібна одна сторінкова таблиця першого рівня з 1024 записами, та 1024 сторінкові таблиці другого рівня з такою ж кількістю записів. Однак при цьому додається ще одне звернення до пам'яті.

В другій структурі довжина сторінкової таблиці визначається лише кількістю фізичних сторінок в основній пам'яті і не залежить від числа цільових програм. Ємність пам'яті зменшується, проте ускладнюється структура - застосовується асоціативна пам'ять, що дозволяє вибирати дані за їх змістом, а не за місцезнаходженням. Але, як було показано вище, кількість фізичних сторінок в сучасних комп'ютерах є досить великою, що робить проблематичним застосування асоціативної сторінкової таблиці. В зв'язку з цим, в сучасних комп'ютерах поступають наступним чином: виділяють сторінковій таблиці область в основній пам'яті, хоча це приводить до подвоєння часу звернення, та використовують додаткову кеш пам'ять, яку називають буфером перетворення з передісторією (TLB - Translation Look-aside Buffer). Використання TLB дозволяє зменшити час перетворення віртуальних адрес у фізичні. Він будується як кеш пам'ять та може бути організованим за принципом прямого, асоціативного та частково-асоціативного відображення. При кожному перетворенні номера віртуальної сторінки в номер фізичної сторінки результат заноситься в TLB: номер фізичної сторінки до асоціативного регістра, а номер віртуальної сторінки - до регістрів тегів. Таким чином, до TLB попадають результати декількох останніх операцій перетворення адрес. При кожному зверненні до основної пам'яті спочатку потрібна віртуальна сторінка шукається в TLB, і при попаданні номер відповідної фізичної сторінки береться з цього буфера. Якщо зафіксовано промах, то процедура перетворення адрес здійснюється за допомогою сторінкової таблиці, яка зберігається в основній пам'яті, після чого номер віртуальної та фізичної сторінок заноситься до TLB. Структура TLB, побудованої як повністю асоціативна кеш пам'ять, наведена на рис. 10.28.

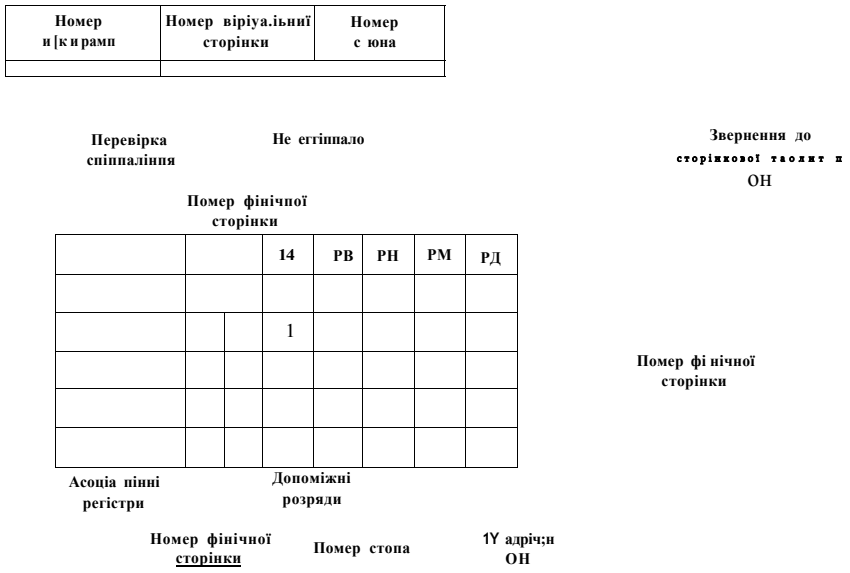


Рис. 10.28. Структура асоціативної кеш пам'яті буфера перетворення з передісторією

Зазвичай число записів (входів) в TLB є невеликим (64-256), наприклад в комп'ютері Pentium III є 64 входи, при розмірі сторінки 4 КБ, що дозволяє отримати швидкий доступ до пам'яті ємністю 256 КБ.

Структура TLB, побудованої як частково-асоціативна кеш пам'ять, наведена на рис. 10.29.

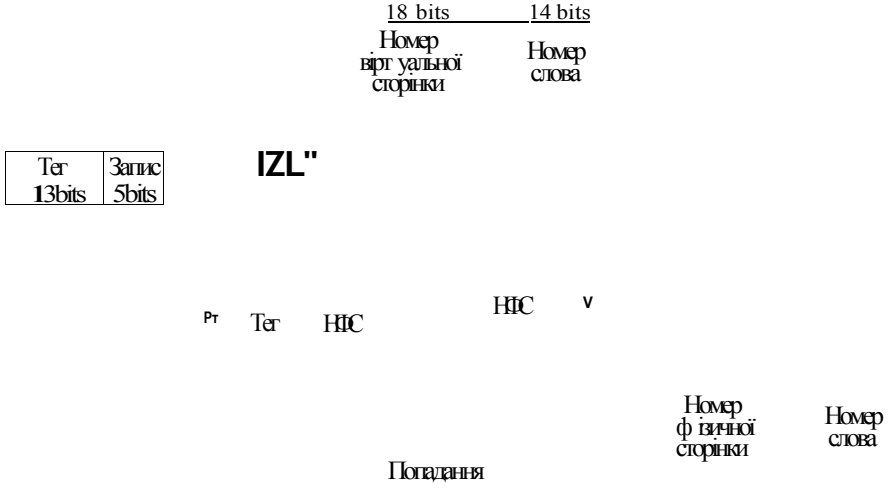


Рис. 10.29. Структура частково-асоціативної кеш пам'яті буфера перетворення з передісторією

Крім номерів програми, віртуальної сторінки та фізичної сторінки слово сторінкової таблиці включає п'ять ознак: РЗ, РВ, РН, РМ, та РД.

Для керування занесення інформації в асоціативні регістри до їх складу вводять додаткові розряди: розряд зайнятості (РЗ) і розряд використання (РВ).

Ознака РЗ (розряд зайнятості) потрібна для реалізації стратегії заміщення вмісту асоціативних регістрів. РЗ встановлюється в 1 при записі інформації в асоціативні регістри. РЗ говорить про те, вільний чи зайнятий даний асоціативний регістр.

Так як зазвичай всі асоціативні регістри зайняті інформацією, то для запису нової необхідно звільнити якийсь асоціативний регістр. Звільняється той регістр, інформація в якому не використовувалась Для фіксації звернень до даної віртуальної сторінки використовується ознака РВ (розряд використання), який встановлюється в 1 при звертанні до віртуальної сторінки. Якщо всі РЗ = 1, то нова інформація засилається в регістр, в якому РВ = 0. Якщо всі РВ = 1, то вони автоматично скидаються

Ознака РН (розряд наявності) встановлюється в одиницю, якщо віртуальна сторінка в даний момент знаходиться в основній пам'яті. В цьому випадку в асоціативному регістрі знаходиться номер фізичної сторінки. Якщо РН = 0, то при спробі звернення до даної віртуальної сторінки операційна система завантажує сторінку з зовнішньої пам'яті до основної. При цьому в асоціативному регістрі наявна інформація про місце розміщення сторінки в зовнішній пам'яті. Завантаження сторінки з зовнішньої пам'яті супроводжується записом до відповідного асоціативного регістра номера фізичної сторінки, до якої буде записана віртуальна сторінка

Сторінка в ОП може змінюватись. Тому, якщо вона не змінилася, немає необхідності її переписувати в ЗП, так як там вже є копія. Якщо ж змінилася, потрібно переписати. Для фіксації змін в сторінках ОП вводиться в сторінкову таблицю розряд модифікації РМ, який дозволяє суттєво зекономити час обміну

Ознака прав доступу ОД вказує вид доступу до сторінки: запис, читання чи дозволені або заборонені обидві операції, і призначений для вирішення питань захисту інформації

Потрібно відзначити, що при сторінковій організації пам'яті виникають ті ж проблеми з заміщенням сторінок в основній пам'яті, як це було для кеш пам'яті. Зазвичай в ОП немає вільних сторінок і при завантаженні нової інформації якусь сторінку доводиться видаляти з ОП. Необхідний алгоритм визначення сторінки, що підлягає знищенню. Тут використовуються ті ж самі методи заміщення сторінок, що і в кеш пам'яті: LRU - знищення сторінки з максимальним простоєм, LFU - знищення сторінки з мінімальною частотою звернення, FIFO - знищення сторінки згідно з чергою поступлення, RAND - знищення сторінки випадковим чином При цьому потрібно відзначити, що плата за промах (коли вміст потрібної сторінки відсутній в основній пам'яті) є досить великою (десятки мілісекунд), оскшьки потрібно робити звернення до повільної зовнішньої пам'яті

Важливим питанням сторінкової організації пам'яті є вибір об ему сторінки. На користь малого об'єму сторінки говорять такі аргументи:

- зменшуються втрати через фрагментацію за рахунок неповного використання об'єму виділеної сторінки програмою;
- зменшується об'єм пересилань.

На користь великого об'єму говорять наступні аргументи:

- зменшуються затрати на сторінкову таблицю;
- зменшується час підготовчих операцій в ЗП.

Зазвичай вибирають об'єм сторінки в діапазоні 8КБ - 32КБ

Таким чином використання сторінкової організація пам^яті дає наступні переваги:

- Віртуальна пам'ять представляється для програміста як пам^ять одного рівня, що спрощує програмування.

* Об єм віртуальної пам яті визначається довжиною адресного поля і може набагато перевищувати ємність основної пам'яті.

• Виключаються протиріччя, пов'язані з розподілом основної пам'яті між системними і прикладними програмами Немає обмеження на кількість програм.

• Виключається фрагментація основної пам'яті.

• При мультипрограмуванні не потрібне послідовне розташування сторінок однієї програми в основній пам'яті.

• Не потрібно переміщення інформації, як це є при розподілі з допомогою базових регістрів.

• В основну пам'ять загружаються активні сторінки за потребою, неактивні сторінки залишаються в зовнішній пам'яті.

Разом з тим, використання сторінкової організації пам яті вимагає додаткових апаратних та програмних засобів, та ускладнює роботу комп'ютера.

10.3.5. *Сегментна організація віртуальної пам'яті*

Зазвичай програма складається з декількох частин, розміри яких наперед невідомі та можуть змінюватись в процесі виконання програми. Для кожної з цих частин повинна бути відведена область в просторі віртуальних адрес, оскільки користуватися віртуальною пам яттю з неперервною нумерацією байтів всіх частин не завжди зручно. Більш зручно, коли кожна частина має свою нумерацію байтів починаючи з нуля. Бажано також, щоб складена таким чином програма могла працювати при динамічному розподілі пам яті, не вимагаючи від програміста зусиль по об єднанню різних її частин в єдиний масив. Це завдання розв'язується в багатьох комп'ютерах шляхом використання особливого методу перетворення віртуальних адрес в фізичні та називається сегментною організацією пам'яті.

Принципи сегментної організації пам'яті є наступними:

• віртуальна пам'ять кожної програми ділиться на частини, що називаються сегментами-

• всередині сегменту адресація байтів є незалежною, починаючи від нуля до якогось максимального значення;

• різні сегменти можуть мати різну довжину;

• довжина сегмента може змінюватись в процесі роботи;

• так як кожний сегмент займає незалежний адресний простір, сегменти можуть рости і скорочуватися незалежно один від одного.

Як приклад на рис. 10.30 наведено сегменти пам яті деякої програми.

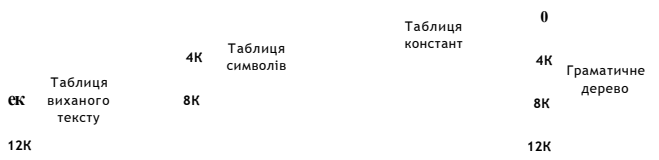


Рис. 10.30. Сегменти пам'яті із чотирьох таблиць

Щоб виконати звернення до такої сегментованої, або двовимірної пам'яті, програма повинна видати адресу, яка складається з двох частин: номера сегмента і внутрішньої

адреси сегмента. Тобто до віртуальної адреси необхідно додати додаткові розряди лівіше номера сторінки, які визначають номер сегмента.

Крім спрощення обробки змінних за об'ємом структур даних, сегментована пам'ять значно спрощує зв'язок процедур, компіляція яких виконана окремо. Якщо процедура в деякому сегменті зазнала змін і повторної компіляції, то решта процедур змінювати не потрібно. Сегментація спрощує і спільне використання даних локальними процесами.

Так як з точки зору програміста сегменти є самостійними логічними об'єктами, припустимо застосування для них різних видів захисту: дозволяється лише читання, запис і т. п.

Таким чином виникає певна ієрархія в організації програм, яка складається з чотирьох ярусів: програма, сегмент, сторінка, слово. Цій ієрархії програм відповідає й ієрархія таблиць перетворення віртуальних адрес у фізичні, як це показано на рис. 10.31.

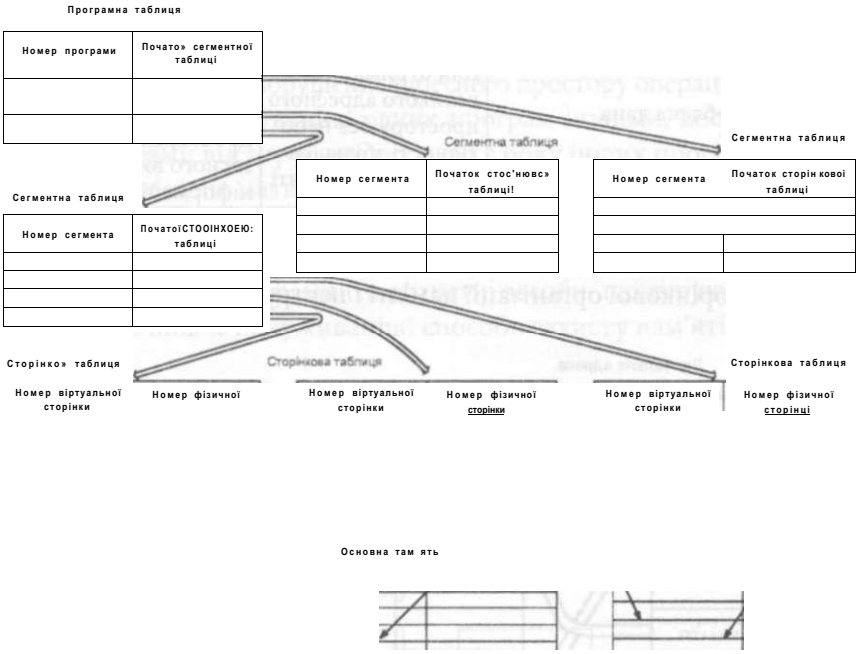


Рис. 10.31. Ієрархія таблиць перетворення віртуальних адресу фізичні

Програмна таблиця включає всі завантажені в комп'ютер програми. Кожній програмі відповідає своя сегментна таблиця. Сегментна таблиця включає сегменти даної програми. Кожному сегменту відповідає своя сторінкова таблиця. Сторінкова таблиця визначає фізичне розташування кожної сторінки в пам'яті (основній і зовнішній).

Хоча в логічному відношенні сегментна і сторінкова організація пам'яті тісно пов'язані між собою та подібні в реалізації, цілі їх застосування різні. Порівняння сегментної і сторінкової організації пам'яті наведено в табл. 10.1.

Таблиця 10.1

п/п	Характеристика	Сторінкова організація пам'яті	Сегментна організація пам'яті
1.	Чи повинен програміст знати, який вид організації пам'яті використовується?	ні	так
2.	Скільки є лінійних адресних просторів?	1	багато
3.	Чи може адресний простір перевищувати ємність пам'яті?	так	так
4.	Чи можуть бути розпізнані та окремо захищені процедури і дані?	ні	так
5.	Чи можна розміщувати таблиці змінного об'єму?	ні	так
6.	Чи можливе спільне застосування процедур декількома користувачами?	ні	так
7.	Для чого була розроблена дана організація пам'яті?	Для отримання великого адресного простору без необхідності збільшення фізичної пам'яті	Для отримання можливості розміщувати програми і дані в логічно незв'язаних адресних просторах і для полегшення сумісного використання і захисту інформації

Формат віртуальної адреси та процес її перетворення в фізичну адресу при використанні сегментної та сторінкової організації пам'яті ілюструє рис. 10.32.

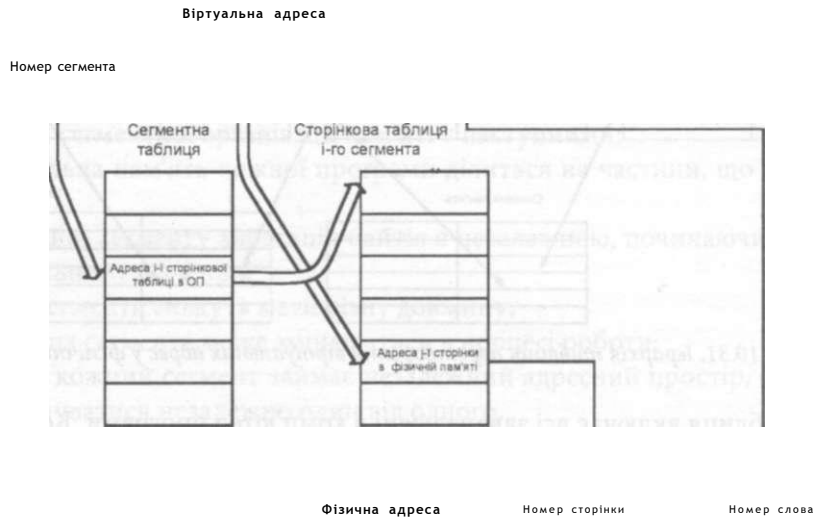


Рис. 10.32. Перетворення віртуальної адреси у фізичну адресу при використанні сегментної та сторінкової організації пам'яті

Як видно з рисунка, для виконання перетворення необхідно два додаткових звернення до пам'яті. Для скорочення кількості звернень для побудови сегментної таблиці може бути застосована, як і при сторінковій організації пам'яті, асоціативна пам'ять.

Ї0.4. *Захист пам'яті від несанкціонованих звернень*

10.4.1. Задачі захисту пам'яті

Сучасні комп'ютери, як правило, одночасно виконують багато завдань для багатьох користувачів, коли в основній пам'яті одночасно знаходяться програми, що належать як до різних користувачів, так і до різних завдань одного користувача. Крім того, в основній пам'яті завжди присутні фрагменти операційної системи. Кожному завданню в основній пам'яті виділяється свій адресний простір. Такі простори, якщо це спеціально не передбачено, зазвичай є незалежними. В той же час в програмах можуть міститися помилки, що може призвести до вторгнення в адресний простір інших завдань. В результаті може бути спотворена інформація, що належить іншим програмам. Отже, у комп'ютері обов'язково повинні бути передбачені заходи запобігання несанкціонованій дії програми одного користувача на роботу програм інших користувачів і на операційну систему. Особливо небезпечні наслідки таких помилок при порушенні адресного простору операційної системи.

Щоб перешкодити руйнуванню одних програм іншими, досить захистити область пам'яті даної програми від спроб запису в неї з боку інших програм (захист від запису). У ряді випадків необхідно мати можливість захисту і від читання з боку інших програм, наприклад при обмеженнях на доступ до системної інформації.

Захист від вторгнення програм в чужі адресні простори реалізується цілим рядом способів, які поєднують програмні та апаратні засоби, що керуються операційною системою. Розглянемо нижче найуживаніші способи захисту пам'яті.

10.4.2. Захист пам'яті за допомогою регістра захисту

Цей спосіб захисту зазвичай використовують при відлагодженні нових програм паралельно з функціонуванням інших програм. Він передбачає захист окремих комірок або блоків пам'яті (рис. 10.33).

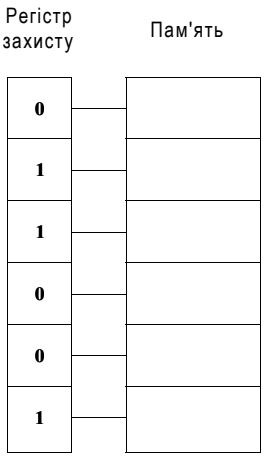


Рис. 10.33. Захист пам'яті з допомогою регістра захисту

Коли розряд захисту рівний 1, то запис до пам'яті заборонено. Може бути декілька розрядів, які вказують режими роботи.

Реалізувати цей спосіб можна також шляхом виділення в кожній комірці пам'яті спеціального розряду захисту і під'єднання його до пристрою керування записом в пам'ять. Встановлення цього розряду в 1 блокує запис в дану комірку. Подібний спосіб використовувався в обчислювальних машинах попередніх поколінь. Зокрема він був застосований в системі Atlas.

10.4.3. Захист пам'яті за граничними адресами

Даний метод захисту є найпоширенішим. Метод припускає наявність в процесорі двох граничних регістрів, вміст яких визначає нижню і верхню межі області пам'яті, куди програма має право доступу (рис. 10.34). Заповнення граничних регістрів проводиться операційною системою при завантаженні програм. При кожному зверненні до пам'яті перевіряється, чи потрапляє використовувана адреса у встановлені межі. Таку перевірку, наприклад, можна організувати на етапі перетворення віртуальної адреси у фізичну. При порушенні межі доступ до пам'яті блокується і формується запит переривання, що викликає відповідну процедуру операційної системи. Нижню межу дозволеної області пам'яті визначає сегментний регістр. Верхня межа підраховується операційною системою відповідно до розміру розміщуваного в пам'яті сегменту.

У розглянутій схемі необхідно, щоб у комп'ютері підтримувалися два режими роботи: привілейований і призначений для користувача. Запис інформації в граничні регістри можливий лише в привілейованому режимі.

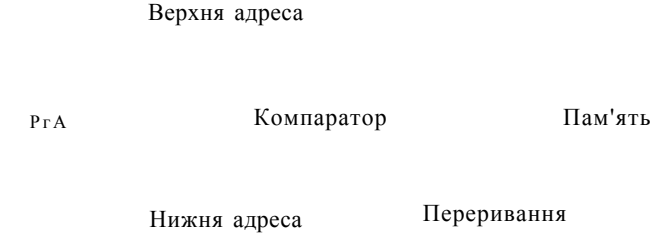


Рис. 10.34. Захист пам'яті за граничними адресами

Як вже було вказано, значення граничних адрес встановлюється операційною системою. Цей спосіб вимагає займання комірок пам'яті підряд. Застосований в системі Stretch і IBM7090.

10.4.4. Захист пам'яті за значеннями ключів

Метод дозволяє організувати захист несуміжних областей пам'яті. Пам'ять умовно ділиться на блоки однакового розміру. Кожному блоку ставиться у відповідність деякий код, який називають ключем захисту пам'яті. Кожній програмі, у свою чергу, присвоюють код захисту програми. Умовою доступу програми до конкретного блоку пам'яті служить збіг ключів захисту пам'яті і програми, або рівність одного з цих ключів нулю. Нульове значення ключа захисту програми надає доступ до всього адресного простору і використовується лише програмами операційної системи. Розподілом ключів захисту програми відає операційна система. Ключ захисту програми зазвичай представлений у

вигляді окремого поля слова стану програми, що зберігається в спеціальному регістрі. Ключі захисту пам'яті зберігаються в спеціальній пам'яті. При кожному зверненні до основної пам'яті схема порівняння проводить порівняння ключів захисту пам'яті і програми. При збігу доступ до пам'яті дозволено. Дії у разі неспівпадіння ключів залежать від того, який вид доступу заборонений: при записі, при читанні або в обох випадках. Якщо з'ясувалося, що даний вид доступу заборонений, то так само як і в методі граничних адрес формується запит переривання і називається відповідна процедура операційної системи.

Описаний спосіб є більш гнучким порівняно з попереднім, так як дозволяє звертатися до областей пам'яті, розміщених не підряд. Він був застосований в системі IBM 360. Схема захисту пам'яті за описаним способом в системі IBM 360 наведена на рис. 10.35.

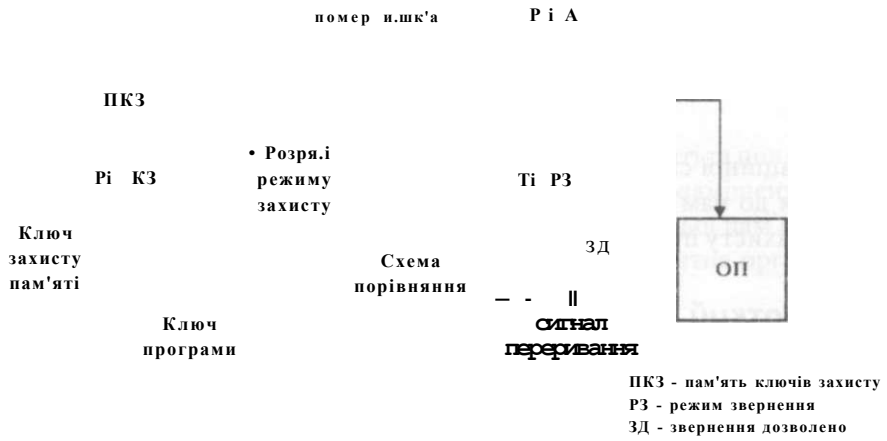


Рис. 10.35. Захист пам'яті за значеннями ключів

В цій системі пам'ять складається з блоків. Кожний блок має код - ключ захисту пам'яті (КЗП). Кожна програма теж має код - ключ програми (КП). Ключі захисту пам'яті зберігаються в пам'яті ключів захисту (ПКЗ). Крім того, є іще тригер режиму захисту (ТгРЗ), в якому зберігається розряд режиму захисту (РРЗ). Доступ дозволений, якщо КЗП = КП. Коли КЗП = КП, а РРЗ = 0, то дозволено лише зчитування. Коли ж КЗП = КП, а РРЗ = 1, то доступ до пам'яті заборонений. Об'єм блоку рівний 2048 байт. Одночасно обробляється 16 програм і, відповідно, використовується 16 варіантів КЗП і КП, тобто коди є 4-розрядни.ми. КП вказується в спеціальному полі слова стану програми (ССП) або каналу (ССК). Коди КП і КЗП встановлює операційна система.

10.4.5. Кільцева схема захисту пам'яті

Захист адресного простору операційної системи від несанкціонованого вторгнення з боку призначених для користувача програм зазвичай організовують шляхом надання системного і призначеного для користувача рівнів привілеїв. Таку структуру прийнято називати кільцевою системою захисту. На рис. 10.36 ця система зображена у вигляді концентричних кіл, де призначений для користувача режим представлений зовнішнім кільцем, а системний - внутрішнім колом. У системному режимі програмі доступні всі

ресурси комп'ютера, а можливості призначеного для користувача режиму істотно обмежені. Перемикання з призначеного для користувача режиму в системний здійснюється спеціальною командою. Вперше описаний підхід був застосований в системі MULTICS на комп'ютері GE 645, де крім ключів захисту використовувалась кільцева схема захисту з 32 рівнями привілеїв. У більшості сучасних комп'ютерів число рівнів привілеїв (кілець захисту) невелике, зокрема в процесорах фірми Intel передбачено чотири рівні привілеїв.

Рис. 10.36. Кільцева схема захисту

В ядрі операційної системи знаходяться програми ініціалізації комп'ютера та керування доступом до пам'яті. Сегменти в внутрішніх кільцях більш захищені, ніж в зовнішніх. Рівні захисту привілеїв задаються двома бітами у відповідних регістрах.

10.5. Короткий зміст розділу

В розділі проведено аналіз характеристик та особливостей апаратних та програмних засобів сучасного комп'ютера, що дозволило виділити наступні їх фундаментальні властивості, які позитивно доповнюють одна одну з точки зору вирішення задачі забезпечення необхідної ємності і високої швидкодії пам'яті за прийнятну ціну:

- чим менший час доступу до пам'яті, тим менша її ємність та вища вартість зберігання в ній одного біта інформації;
- чим більша ємність пам'яті, тим більший час доступу до неї та нижча вартість зберігання в ній одного біта інформації;
- існує значна різниця між продуктивністю процесора і основної пам'яті, а також між продуктивністю основної та зовнішньої пам'яті;
- комп'ютерні програми володіють властивістю локальності за зверненням до пам'яті.

Ці властивості є підґрунтям доцільності використання при побудові системи пам'яті підходу, відомого як принцип ієрархічної організації пам'яті.

Відповідно до цього принципу пам'ять комп'ютера складається із пристроїв пам'яті різних типів, які, залежно від характеристик, належать до певного рівня ієрархії. Пам'ять нижчого рівня має меншу ємність, швидша і має велику вартість в перерахунку на біт, ніж пам'ять вищого рівня. Рівні ієрархії взаємозв'язані: всі дані на деякому нижчому рівні можуть бути також знайдені на вищому рівні, і всі дані на цьому вищому рівні можуть бути знайдені на наступному вищому рівні і т. д. З рухом вверх по ієрархічній структурі зменшується співвідношення вартість/біт, зростає ємність та час доступу. Однак завдяки властивості локальності за зверненням з рухом вверх по ієрархічній структурі зменшу-

ється частота звернення до пам'яті з боку нижчих рівнів, що веде до зменшення загальної вартості при заданому рівні продуктивності. На кожному рівні ієрархії пам'ять розбивається на блоки, які є найменшою інформаційною одиницею, що пересилається між двома сусідніми рівнями ієрархії. Розмір блоків може бути фіксованим або змінним. При фіксованому розмірі блоку ємність пам'яті зазвичай кратна його розміру. Розмір блоків на кожному рівні ієрархії найчастіше різний і збільшується від нижчих рівнів до вищих.

Виходячи з принципу ієрархічної організації пам'яті в розділі описано організацію взаємодії між рівнями ієрархічної пам'яті. Зокрема, наведено принципи обміну інформацією між рівнями ієрархічної пам'яті, характеристики, які використовуються для оцінки ефективності ієрархічної пам'яті, пояснено місце кеш пам'яті в складі комп'ютера, призначення і логіку роботи кеш пам'яті, описана вигода від поділу кеш пам'яті першого рівня на кеш пам'ять даних та кеш пам'ять команд. Обґрунтована різноманітність методів відображення основної пам'яті на кеш пам'ять, показано, якими методами забезпечується узгодженість вмісту основної і кеш пам'яті, та чим обумовлене введення додаткових рівнів кеш пам'яті, і які чинники впливають на вибір ємності кеш пам'яті та розміру блоку.

Введено поняття статичного та динамічного розподілу пам'яті та поняття віртуальної пам'яті та сторінкової організації пам'яті. Описані алгоритми заміщення, які використовуються при завантаженні в основну пам'ять вмісту зовнішньої пам'яті, призначення буфера швидкого перетворення адреси (TLB). Наведена сегментна організація пам'яті. Розглянуті питання захисту пам'яті.

10.6. Література для подальшого читання

Обґрунтування потреби в ієрархії пам'яті і сам термін, а також принципи керування двома рівнями пам'яті вперше запропоновано в роботі [17]. Віртуальна пам'ять була реалізована в комп'ютерах системи IBM 370, в якій вперше був використаний буфер швидкого перетворення адреси, інформацію про що можна знайти в роботі [6].

Питання побудови кеш пам'яті, організацію взаємодії між рівнями ієрархічної пам'яті, принципи обміну інформацією між рівнями ієрархічної пам'яті, характеристики, які використовуються для оцінки ефективності ієрархічної пам'яті, різноманітність методів відображення основної пам'яті на кеш пам'ять, методи забезпечення узгодженості вмісту основної і кеш пам'яті, та які чинники впливають на вибір ємності кеш пам'яті і розмір блоку, розглянуті в роботах [1-5,7,11-16,18-34].

Питання сторінкової організації пам'яті, зокрема як це було зроблено в системі Атлас, та захисту в ній інформації на прикладі систем IBM 360, Intel 80286 та Multics описані в роботах [10, 35-37].

10.7. Література до розділу 10

1. AGARWAL, A. [1987]. *Analysis of Cache Performance for Operating Systems and Multiprogramming,* Ph.D. Thesis, Stanford Univ., Tech. Rep. No. CSL-TR-87-332 (May).

2. AGARWAL, A. AND S. D. PUDAR [1993J. "Column-associative caches: A technique for reducing the miss rate of direct-mapped caches," 20th Annual Int'l Symposium on Computer Architecture ISCA '20, San Diego, Calif, May 16-19. *Computer Architecture News* 21:2 (May), 179-90.

3. BAER, J.-L. AND W.-H. WANG f 1988]. "On the inclusion property for multi-level cache hierarchies", *Proc. 15th Annual Symposium on Computer Architecture* (May-June), Honolulu, 73-80.

4. BHANDARKAR, D. P. [1995], *Alpha Architecture Implementations,* Digital Press, Newton, Mass.

5. BORG, A., R. E. KESSLER, AND D. W. WALL [1990]. "Generation and analysis of very long address traces", *Proc. 17th Annual Int'l Symposium on Computer Architecture* (Cat. No. 90CH2887-8), Seattle, May 28-31, IEEE Computer Society Press, Los Alamitos, Calif, 270-9.

6. CASE, R. P. AND A. PADEGS [1978]. "The architecture of the IBM System/370", *Communications of the ACM* 21:1, 73-96. Also appears in D. P. Siewjorek, C. G. Bell, and A. Newell, *Computer Structures: Principles and Examples* (1982), McGraw-Hill, New York, 830-855.

7. CLARK, D. W. [1983]. "Cache performance of the *VAX-W780", ACM Trans, on Computer Systems* 1:1,24-37.

8. CONTI, C, D. H. GIBSON, AND S. H. P1TKOWSKY [1968J. "Structural aspects of the System/360 Model 85, Part I: General organization", *IBM Systems }.* 7:1, 2-14.

9. CRAWFORD, J. H. AND P. P. GELSINGER [1987]. *Programming the 80386,* Sybex, Alameda, Calif.

10. FABRY, R. S. [1974]. "Capability based addressing", *Comm. ACM* 17:7 (July), 403-412.

11. FARKAS, K. I. AND N. P. JOUPPI [1994]. "Complexity/performance tradeoffs with non-blocking loads", *Proc. 21st Annual Int'l Symposium on Computer Architecture,* Chicago (April).

12. GAO, Q. S. [1993]. "The Chinese remainder theorem and the prime memory system", 20th Annual Int'l Symposium on Computer Architecture ISCA '20, San Diego, May 16-19, 1993. *Computer Architecture News* 21;:2 (May), 337—40.

13. GEE, J. D., M. D. HILL, D. N PNEVMATIKATOS, AND A. J. SMITH [1993]. "Cache performance of the SPEC92 benchmark suite", *IEEE Micro* 13:4 (August), 17-27.

14. HILL, M. D. [1987]. *Aspects of Cache Memory and Instruction Buffer Performance,* Ph.D. Thesis, University of Calif, at Berkeley, Computer Science Division, Tech. Rep. UCB/CSD 87/381 (November).

15. HILL, M. D. [1988]. "A case for direct mapped caches", *Computer* 21:12 (December), 25-40.

16. JOUPPI, N. P. [1990]. "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers", *Proc. 17th Annual Int'l Symposium on Computer Architecture* (Cat. No. 90CH2887-8), Seattle, May 28-31, 1990. IEEE Computer Society Press, Los Alamitos, Calif, 364-73.

17. KILBURN, T., D. B. G. EDWARDS, M. J. LANIGAN, AND F. H. SUMNER [1962], "One-level storage system", *IRE Trans, on Electronic Computers* EC-11 (April) 223-235. Also appears in D. P. Siewiorek, C. G. Bell, and A. Newell, *Computer Structures: Principles and Examples* (1982), McGraw-Hill, New York, 135-148.

18. KROFT, D. [1981]. "Lockup-free instruction fetch/prefetch cache organization", *Proc. Eighth Annual Symposium on Computer Architecture* (May 12-14), Minneapolis, 81-87.

19. LAM, M. S., E. E. ROTHBERG, AND M. E. WOLF [1991]. "The cache performance and optimizations of blocked algorithms", Fourth Int'l Conf. on Architectural Support for Programming Languages and Operating Systems, Santa Clara, Calif, April 8-11. *SIGPLAN Notices* 26:4 (April), 63-74.

20. LEBECK, A. R. AND D. A. WOOD [1994]. "Cache profiling and the SPEC benchmarks: A case study", *Computer* 27:10 (October), 15-26.

21. LIPTAY, J. S. [1968]. "Structural aspects of the System/360 Model 85, Part II: The cache", *IBM Systems }.* 7:1, 15-21.

22. MCFARLING, S. [1989]. "Program optimization for instruction caches", *Proc. Third Int'l Conf. on Architectural Support for Programming Languages and Operating Systems* (April 3-6), Boston, 183-191.

23. MOWRY, T. C, S. LAM, AND A. GUPTA [1992]. "Design and evaluation of a compiler algorithm for prefetching", Fifth Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-V), Boston, October 12-15 , *SIGPLAN Notices* 27:9 (September), 62-73.

24. PALACHARLA, S. AND R. E. KESSLER [1994]. "Evaluating stream buffers as a secondary cache replacement" *Proc. 21st Annual Int'l Symposium on Computer Architecture,* Chicago, April 18-21, IEEE Computer Society Press, Los Alamitos, Calif., 24-33.

25. PRZYBYLSKI, S. A. [1990]. *Cache Design: A Performance-Directed Approach,* Morgan Kaufmann Publishers, San Mateo, Calif.

26. PRZYBYLSKI, S. A., M. HOROWITZ, AND J. L. HENNESSY [1988], "Performance tradeoffs in cache design", *Proc. 15th Annual Symposium on Computer Architecture* (May-June), Honolulu, 290-298.

27. SAAVEDRA-BARRERA, R. H. [1992]. *CPU Performance Evaluation and Execution Time Prediction Using Narrow Spectrum Benchmarking,* Ph.D. Dissertation, University of Calif, Berkeley (May).

28. SAMPLES, A. D. AND P. N. HILFINGER [1988]. "Code reorganization for instruction caches", Tech. Rep. UCB/CSD 88/447 (October), University of Calif, Berkeley.

29. SITES, R. L. (ED.) [1992]. *Alpha Architecture Reference Manual,* Digital Press, Burlington, Mass.

30. SMITH, A. J. [1982]. "Cache memories", *Computing Surveys* 14:3 (September), 473-530.

31. SMITH, J. E. AND J. R. GOODMAN [1983]. "A study of instruction cache organizations and replacement policies", *Proc. 10th Annual Symposium on Computer Architecture* (June 5-7), Stockholm, 132-137.

32. STRECKER, W. D. [1976]. "Cache memories for the PDP-11?," *Proc. Third Annual Symposium on Computer Architecture* (January), Pittsburgh, 155-158.

33. TORRELLAS, J., A. GUPTA, AND J. HENNESSY [1992], "Characterizing the caching and synchronization performance of a multiprocessor operating system", Fifth Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-V), Boston, October 12-15, *SIGPLAN Notices* 27:9 (September), 162-74.

34. WANG, W.-H., J.-L. BAER, AND H. M. LEVY [1989]. "Organization and performance of a two-level virtual-real cache hierarchy", *Proc. 16th Annual Symposium on Computer Architecture* (May 28-June 1), Jerusalem, 140-148.

35. WILKES, M. [1965]. "Slave memories and dynamic storage allocation", *IEEE Trans. Electronic Computers* EC-14:2 (April), 270-271.

36. WILKES, M. V. [1982]. "Hardware support for memory protection: Capability implementations", *Proc. Symposium on Architectural Support for Programming Languages and Operating Systems* (March 1-3), Palo Alto, Calif, 107-116.

37. WULF, W. A., R. LEVIN, AND S. P. HARBISON [1981], *Hydra/C.mmp: An Experimental Computer System,* McGraw-Hill, New York.

10.8. *Питання до розділу 10*

1. Чим пояснюється різниця між продуктивністю процесора та пам'яті?

2. Що показує аналіз зміни з роками швидкодії статичної і динамічної напівпровідникової та дискової пам'яті і процесора?

3. Як зростає з роками розрив між швидкодією процесора і динамічної пам'яті? На що це впливає?

4. Як зростає з роками розрив між швидкодією динамічної і дискової пам'яті? На що це впливає?

5. Чим викликана необхідність побудови системи пам'яті за ієрархічним принципом?

6. Що включає поняття «локальність за зверненням»?

7. Що таке просторова локальність?

8. Що таке часова локальність?

9. Які висновки витікають з властивості локальності?

10. Наведіть принцип ієрархічної організації пам'яті

11. Завдяки чому середній час доступу в ієрархічній системі пам яті визначається більш швидкодіючими видами пам'яті?

12. Що в ієрархічній системі пам'яті визначають терміни «промах» і «попадання»?

13. На які питання необхідно відповісти, щоб охарактеризувати певний рівень ієрархічної пам яті?

14. Наведіть принципи обміну інформацією між рівнями ієрархічної пам'яті.

15. Які характеристики використовуються для оцінки ефективності ієрархічної пам'яті?

16. Поясніть місце кеш пам'яті в складі комп'ютера,

17. Поясніть призначення і логіку роботи кеш пам'яті.

18. В чому вигода від поділу кеш пам'яті першого рівня на кеш пам'ять даних та кеш пам'ять команд?

19. Які проблеми породжує включення кеш пам'яті в ієрархію пам'яті?

20. Чим обумовлена різноманітність методів відображення основної пам'яті на кеш пам'ять?

21. Якій вимозі повинен відповідати «ідеальний» алгоритм заміщення вмісту кеш пам'яті?

22. Якими методами забезпечується узгодженість вмісту основної і кеш пам'яті?

23. Чим обумовлене введення додаткових рівнів кеш пам'яті?

24. Які чинники впливають на вибір ємності кеш пам'яті та розміру блоку?

25. Як співвідносяться характеристики звичайної і дискової кеш пам'яті?

26. Якими засобами забезпечується віртуалізація пам яті?

27. Чи існує обмеження на розмір віртуального простору?

28. Що визначає об'єм сторінкової таблиці?

29. Якими прийомами досягають скорочення об'єму сторінкових таблиць?

30. Які алгоритми заміщення використовуються при завантаженні в основну пам ять нової віртуальної сторінки?

31. Поясніть призначення буфера швидкого перетворення адреси (ТЬВ).

32. Чим мотивується розбиття віртуальних секторів на сторінки?

33. Яка частина віртуальної адреси залишається незмінною при його перетворенні у фізичну адресу?

34. Наведіть способи апаратної реалізації сторінкової таблиці.

35. Наведіть алгоритми заміщення сторінок в основній пам'яті.

36. Для чого призначена сегментна організація пам'яті?

37. Дайте порівняння сторінкової та сегментної організації пам'яті.

38. Наведіть ієрархію таблиць перетворення віртуальних адрес у фізичні.

39. Чим обумовлена необхідність захисту пам яті?

40. Назвіть способи захисту пам'яті.

41. Поясніть спосіб захисту пам'яті за допомогою ключів захисту.

Розділ 11

В даному розділі будуть розглянуті питання взаємодії з комп'ютером пристроїв, призначених для введення інформації в комп'ютер та виведення інформації з комп'ютера. Оскільки в комп'ютері може бути багато пристроїв введення-виведення, при зверненні до них їх потрібно розпізнати. Спосіб розпізнавання залежить від способу їх підключення до процесора. В розділі буде наведено пояснення способів розпізнавання пристроїв введення-виведення з використанням шини введення-виведення, лінії активації та скритого пам'яттю введення-виведення. Буде наведено методи керування введенням-виведенням та пояснена суть, переваги та недоліки програмно керованого введення-виведення.

Оскільки важливою складовою частиною засобів керування введенням-виведенням є система переривання програм, яка призначена для реакції на програмно незалежні події, в розділі буде надано її детальний опис. Переривання програм - це властивість комп'ютера тимчасово переривати виконання біжучої програми при виконанні деяких подій і передавати керування програмі, яка спеціально передбачена для даної події. Буде введено основні поняття та характеристики системи переривання програм: запити переривання, переривальні програми, перериванні програми, час реакції, час обслуговування переривання, глибина переривання. Будуть розглянуті програмно-апаратні засоби, які забезпечують визначення допустимого моменту переривання і початкової адреси переривальної програми при поступленні запиту переривання, а для забезпечення повернення до перериваної програми забезпечують збереження вмісту елементів пам'яті комп'ютера в момент її переривання, та, після завершення виконання переривальної команди, відновлюють цей вміст та надають комп'ютеру можливість продовжити виконання перериваної програми.

Крім того, буде наведено опис прямого доступу до пам'яті, його переваги та недоліки, та описана організація введення-виведення під керуванням периферійних процесорів (каналів), причини появи каналів введення-виведення, їх функції, керуюча інформація каналів введення-виведення, а також функції та склад селекторного та мультиплексного каналів.

Т 7.7. *Під'єднаний зовнішніх пристроїв до комп'ютера*

Зовнішні (периферійні) пристрої під'єднуються до комп'ютера за допомогою відповідних інтерфейсів. Інтерфейси визначають типи роз'ємів, множину сигналів та протоколи обміну ними. По відношенню до комп'ютера зовнішніми пристроями можуть бути інші комп'ютери та пристрої введення-виведення.

Пристрої введення-виведення призначені для введення інформації в комп'ютер та виведення інформації з комп'ютера.

До числа пристроїв введення належать наступні:

- Клавіатура.
- Оптичні пристрої, зокрема зчитувачі перфокарт, паперових стрічок та штрихових кодів, цифрова відеокамера, оптичний пристрій зчитування міток.
- Магнітні пристрої, наприклад, пристрій для зчитування коду, нанесеного в вигляді магнітних стрічок.
- Екранні пристрої, до числа яких входять сенсорні екрани, світлове перо, мишка.
- Широкий спектр аналогових пристроїв з цифровим виходом.

До числа пристроїв виведення належать наступні:

- Електронно-променева трубка.
- Рідкокристалічний дисплей.
- Принтер (ударного типу, струменевий, лазерний, точково-матричний).
- Графопобудовувач.
- Перфоратори карт та стрічок.
- Широкий спектр аналогових пристроїв з цифровим входом.
- Звукові пристрої.

Пристрої введення-виведення та з'єднані з ними внутрішні пристрої комп'ютера, крім вузлів прямого функціонального призначення, містять елементи, які переміщують інформацію між ними та процесором і основною пам'яттю. До цих елементів належать:

- Блоки, призначені для зберігання інформації, яка підлягає введенню-виведенню.
- Шини, по яких здійснюється обмін інформацією між процесором та пам'яттю комп'ютера і керуючими вузлами зовнішніх пристроїв.
- Керуючі вузли внутрішніх та периферійних пристроїв (контролери введення-виведення).
- Пристрої зв'язку (інтерфейсні схеми, які далі скорочено будемо називати інтерфейсами) зовнішніх пристроїв, які забезпечують зв'язок між зовнішніми і внутрішніми пристроями комп'ютера відповідно до протоколів обміну.

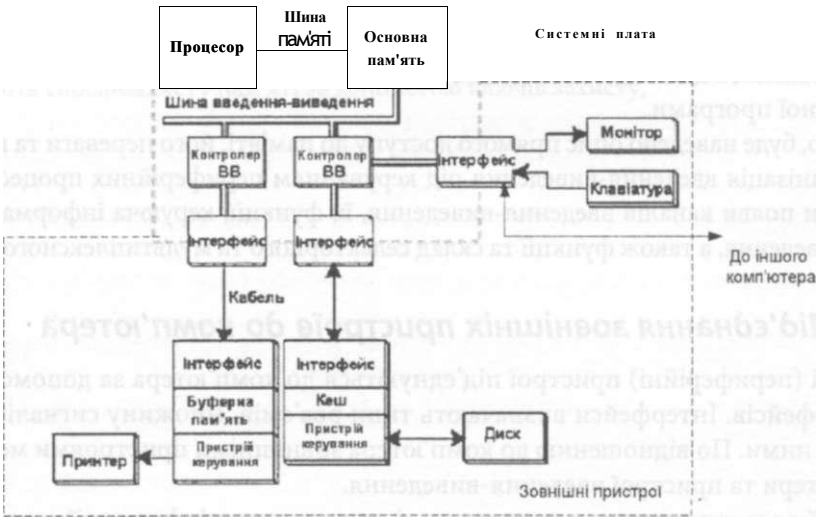


Рис. 11.1. Варіант під'єднання зовнішніх пристроїв до комп'ютера

Рис. 11.1 показує, як всі ці компоненти можуть бути з'єднані між собою, щоб сформувати об'єднану підсистему введення-виведення. Контролери введення-виведення призначені для переміщення даних між основною пам яттю та інтерфейсом відповідного пристрою. Інтерфейси проектуються спеціально під конкретний пристрій введення-виведення, щоб зв'язатися з певними видами пристроїв, як наприклад клавіатурою, дисками або принтерами. Інтерфейси повідомляють контролери про готовність зовнішніх пристроїв до приймання наступної партії даних, або, повідомляють зовнішні пристрої, що внутрішні пристрої комп'ютера готові одержати від них наступну партію даних.

Порядок обміну інформацією між передавачем і приймачем задається в протоколі обміну. Протоколи вказують порядок поступлення сигналів керування, як наприклад, скидання принтера сигналів статусу, як наприклад готовність принтера, або сигналів передачі даних, як наприклад "тут є байти, які ви запросили."

В комп'ютерах використовуються два принципи передачі інформації - синхронний і асинхронний. При синхронному принципі передавач інформації утримує на своєму виході інформацію деякий наперед визначений час, достатній для її фіксування в приймачі. При асинхронному принципі приймач отримавши інформацію повідомляє про це приймача, який лише після цього може змінити інформацію на своєму виході. Тому існують два методи передачі даних. Перший передбачає передачу разом з даним тактового сигналу, призначеного для фіксування даного. Цей вид протоколів обміну називають протоколами з стробуванням. У більшості протоколів обміну використовується другий метод, згідно з яким приймач повинен підтвердити отримання відправлених йому команд і даних, або вказати, що готовий одержати дані. Цей вид протоколів обміну називають протоколами з квітуванням.

Зовнішні пристрої, які оперують великими блоками даних, як наприклад, принтери і драйвери магнітних дисків і стрічок часто обладнуються буферною пам'яттю. Буферна пам'ять дозволяє операційній системі комп'ютера відправляти або приймати великі об'єми даних до/від зовнішніх пристроїв найшвидше. Додаткова спеціалізована пам ять на дисках є зазвичай різновидністю кеш пам'яті, тоді як принтери обладнуються повільнішою пам'яттю з довільною вибіркою. Пристрої керування здійснюють керування відповідним зовнішнім пристроєм та зчитують дані з буферної пам'яті і вказують місце їх поступлення.

Магнітні диски та стрічки належать до типів пам'яті, призначених для тривалого зберігання інформації. Проте, зрозуміло, що і для них існує відповідний термін зберігання. Він становить приблизно п'ять років для магнітних пристроїв пам'яті та до 100 років для оптичних пристроїв пам'яті.

Інтерфейси введення-виведення забезпечують узгодження передачі інформації між внутрішніми (як наприклад пам'ять і регістри процесора) та зовнішніми пристроями введення-виведення. Зовнішні пристрої часто є електромеханічними, тоді як процесор і пам ять - електронні пристрої. Тому швидкість передачі даних від зовнішніх пристроїв є зазвичай, повільнішою.

11.2. Розпізнавання пристроїв введення-виведення

Оскільки в комп'ютері може бути багато пристроїв введення-виведення, при зверненні до них їх потрібно розпізнати. Спосіб розпізнавання залежить від способу їх підключення до процесора.

На рис. 11.2 для пристроїв введення-виведення виділена окрема шина введення-виведення, яка з'єднує їх з процесором.

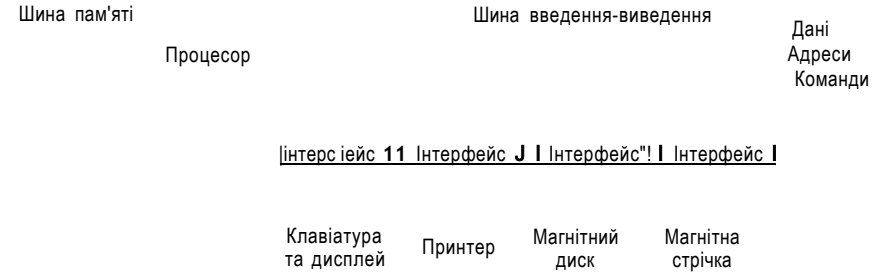


Рис. 11.2. Використання шини введення-виведення

В цьому випадку процесор звертається до пристрою введення-виведення за його номером і незалежно від пам'яті. Наприклад, він може вказувати тип операції введення-виведення та номер пристрою введення-виведення.

На рис. 11.1 наведено варіант взаємодії внутрішніх та зовнішніх пристроїв комп'ютера з використанням двох окремих шин - шини пам'яті та шини введення-виведення. Можливе використання об'єднаної шини пам'яті та введення-виведення. Якщо використовується спільна шина для основної пам'яті та пристроїв введення-виведення, то може бути використана спеціальна лінія активації, по якій вибирається для взаємодії з процесором або пам'ять, або пристрій введення-виведення (рис. 11.3).

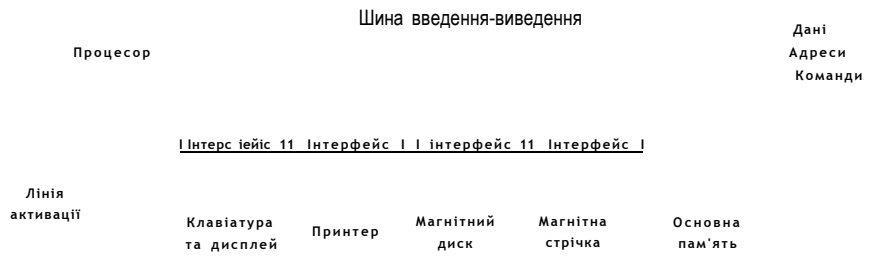


Рис. 11.3. Використання спільної шини

Коли на лінії активації "0" - проводиться обмін процесора з основною пам'яттю, коли "1" - з пристроями введення-виведення.

В цьому випадку об'єднана шина розподіляється в часі між пам'яттю та пристроями введення-виведення. Хоча при такому підході продуктивність комп'ютера дещо нижча, ніж при використанні двох окремих шин, він часто застосовується завдяки таким перевагам як простота організації взаємодії пристроїв та нарощування їх кількості. Прикладом об'єднаних шин є шини VME bus, Multibus II, Nubus.

Інший спосіб організації взаємодії процесора з пам'яттю та пристроями введення-виведення - це включення номерів пристроїв введення-виведення в адресний простір пам'яті. Це так зване скрите пам'яттю введення-виведення. Тут не потрібна лінія акти-

вації. Зв'язок процесора з пристроями введення-виведення здійснюється так само, як з комірками пам'яті. Цей спосіб використовується частіше за інші.

Для взаємодії з процесором пристрої введення-виведення мають спеціальні вузли (рис. 11.4), які називають інтерфейсами або інтерфейсними схемами.

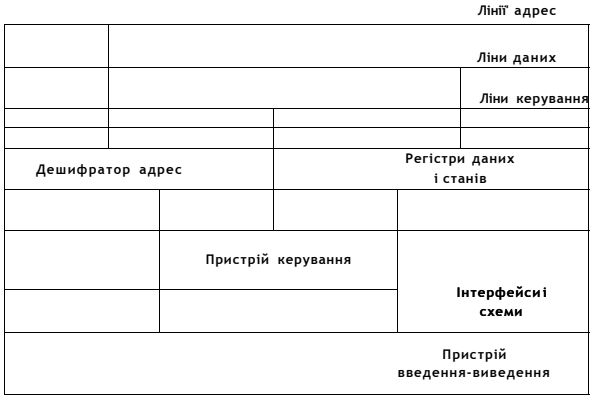


Рис. 11.4. Інтерфейсні схеми пристроїв введення-виведення

Кожен пристрій введення та виведення має інтерфейсну схему (рис. 11.1), яка виконує наступні функції:

- декодує адресу пристрою (номер пристрою), який поступає з процесора;
- декодує команди (визначає потрібну дію);
- забезпечує взаємодію з контролером пристрою введення-виведення;
- синхронізує потік даних і контролює швидкість передачі даних між пристроєм введення-виведення і процесором або пам'яттю.

1 1.3. Методи керування введенням-виведенням

В комп'ютерах використовують чотири загальних методи керування введенням-виведенням. Це наступні методи:

- програмно кероване введення-виведення,
- кероване перериваннями введення-виведення (введення-виведення за перериваннями),
- прямий доступ до пам'яті,
- введення-виведення під керуванням периферійних процесорів (каналів).

Тип методу керування введенням-виведенням суттєво впливає на структуру та продуктивність комп'ютера. Тому потрібно знати, в якому випадку доцільно використовувати конкретний метод введення-виведення в деякому комп'ютері, та як він використовується.

11.4. Програмно-кероване введення-виведення

В комп'ютерах, що використовують програмно-кероване введення-виведення, в інгерфейсній схемі кожного пристрою введення-виведення наявний регістр команд і станів РКС (рис. 11.5). В цьому регістрі є розряд прапорця Е який при потребі обміну з боку пристрою введення-виведення встановлюється в "1".

Рис. 11.5. Взаємодія між процесором і пристроями введення-виведення

Процесор безупинно опитує регістр команд і станів кожного пристрою введення-виведення, чекаючи на надходження даних (рис. 11.6). Тому програмно-кероване введення-виведення іноді називають введенням-виведенням за запитами. Як тільки процесор виявляє умову готовності даних, він діє згідно з вказівками команд з програми відповідного пристрою введення-виведення.

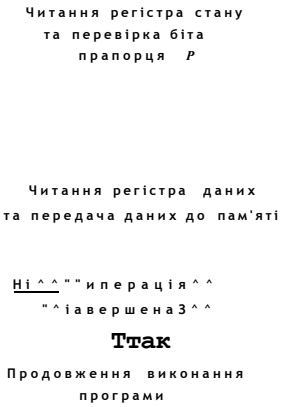


Рис. 11.6. Опитування пристрою введення-виведення та виконання операції введення даних

Вигода від використання цього підходу полягає в тому, що забезпечується програмний контроль над поведінкою кожного пристрою. Постійний опит регістра, проте, є проблемою. Процесор знаходиться в безперервній петлі "активного очікування", поки він почне обслуговувати запит введення виведення. Він не виконує інших функцій, поки є процес введення виведення. Тобто продуктивність процесора знижується аж до рівня продуктивності пристроїв введення виведення. Внаслідок цих обмежень, програмно-кероване введення виведення найкраще підходить для спеціальних систем, де вимоги до продуктивності процесора невисокі.

Розглянемо програмно кероване введення виведення на прикладі відеотерміналу в складі клавіатури і дисплею. Швидкість передачі даних від клавіатури в комп ютер визначається в основному швидкістю роботи оператора, яка рівна декільком символам за секунду. Швидкість передачі даних з комп'ютера на відеотермінал значно вища. Вона визначається швидкістю передачі даних по шині, і типово знаходиться біля показника 1000 символів за секунду. Але це значно менше, ніж може забезпечити процесор (мільярди операцій за секунду). Проблема узгодження швидкостей вирішується наступним чином. Процесор посилає перший символ, чекає на підтвердження відображення символу на дисплеї, посилає другий символ і т. д. При взаємодії з клавіатурою процесор чекає сигнал, який свідчить про те, що клавіша натиснута і можна виконувати ввід коду символу.

Один з варіантів з'єднання процесора і відеотерміналу показаний на рис. 11.7.

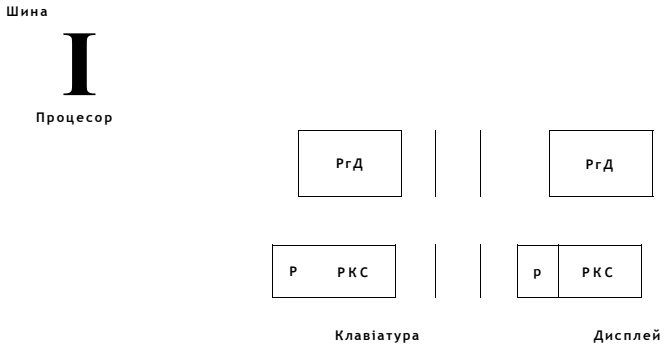


Рис. 11.7. Підключення відеотерміналу до процесора

При пересиланні даних з клавіатури послідовність операцій має бути наступною. Процесор постійно перевіряє вміст прапорця Б регістра РКС, аж поки він не встановиться в одиницю, після чого відбувається пересилання даного з регістра даних РгД в один з регістрів процесора, номер якого вказується в команді.

Аналогічно при пересиланні даних на дисплей процесор постійно перевіряє вміст прапорця Б регістра РКС, аж поки він не встановиться в одиницю, після чого відбувається пересилання даного з одного з регістрів процесора, номер якого вказується в команді, до регістра даних РгД.

11.5. Система переривання програм та організація введення-виведення за перериваннями

11.5.1. Функції системи переривання програм

Роботу комп'ютера можна представити як послідовність подій: виконання операцій, запити на введення-виведення, зміна станів зовнішніх об'єктів, якими керує комп'ютер, і т. д. Частина цих подій є програмно визначеною, тобто передбаченою в програмі, інша частина подій є програмно незалежною, тобто моменти виникнення подій наперед невідомі. До програмно незалежних належать:

- зупинення виконання програми, пов'язане з перевищенням виділеного часу для її виконання;
- запити оператора, який вирішив внести зміни до програми під час її виконання;
- запити периферійних пристроїв за причини завершення операцій введення-виведення або потреби проведення додаткових операцій по їх обслуговуванню;
- переповнення розрядної сітки;
- ділення на нуль;
- вихід за дозволені границі пам'яті і т. д.

Останні три названі вище та подібні події, які ініційовані апаратними засобами та фіксуються програмою, виділяють в окремий клас подій зважаючи на їх важливість та те, що вони не є передбачуваними. Наприклад, в багатопрограмних комп'ютерних сис-

темах, до яких на даний час належать всі універсальні комп'ютери, необхідна фіксація таких подій з метою захисту однієї програми від модифікації іншою. Програми взаємодіють з операційною системою комп'ютера через ці події.

Комп'ютер реагує на програмно визначені події відповідно до вказівок програми. Для реакції на програмно незалежні події в комп'ютер введено спеціальні апаратно-програмні засоби, які дістали назву системи переривання програм (СПП). Ці засоби є невід'ємною частиною сучасних комп'ютерів. Без них поява будь-якої програмно незалежної події приведе до необхідності повторного запуску програми.

Переривання програм - це властивість комп'ютера тимчасово переривати виконання поточної програми на час виконання деяких подій і передавати керування програмі, яка спеціально передбачена для даної події.

Сигнали, які сповіщають про появу програмно незалежних подій, називають запитами переривання.

Програми, на виконання яких є запити, називають переривальними програмами, тобто такими, що переривають інші програми. А програми, які виконувались до появи запитів, називають перериваними програмами, тобто такими, що перериваються.

Часова діаграма процесу переривання наведена на рис. 11.8.

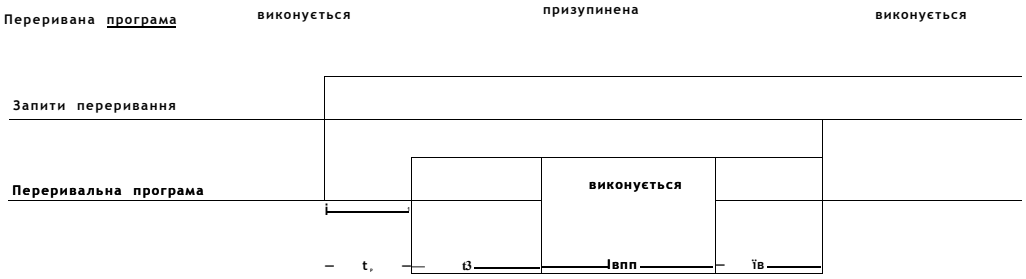


Рис. 11.8. Часова діаграма процесу переривання

Тут tp - час реакції системи переривання на запит переривання; t3 - час запам'ятовування стану перериваної програми, tenn - час виконання переривальної програми, **tB** - час виходу з переривальної програми та повернення до перериваної програми (відновлення її стану).

11.5.2. Характеристики системи переривання програм

До основних характеристик системи переривання програм належать наступні:

- час реакції **tp,** який показує як швидко система реагує на запит переривання;
- час обслуговування переривання - затрати часу на вхід в переривальну програму та вихід з переривальної програми to = із + **tB;**
- глибина переривання - максимальне число програм, які можуть переривати одна одну;
- кількість входів приймання переривань.

На рис. 11.9 показано переривання двох програм запитами, причому перша програма має глибину переривання 2, а друга - 3.

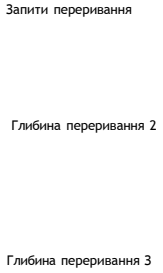


Рис. 11.9. Переривання програм запитами

В першому випадку, коли поступив запит переривання 2, виконання першої програми переривається і виконується друга програма, яка виконується до завершення не дивлячись на те, що під час її виконання поступив запит 3. Це пояснюється тим, що система переривання програм не дозволяє виконання наступного переривання. Після закінчення виконання програми 2 виконується програма 3 (звичайно, якщо вона має вищий пріоритет, ніж програма 1), а тоді відбувається повернення до виконання програми 1. В другому випадку з поступленням кожного запиту переривання він обслуговується без затримки. Чим більша глибина переривання, тим краще враховуються пріоритети запитів переривання. Якщо tp або to більше за час надходження запиту переривання від того ж джерела (події), то виникає так зване насичення системи переривання, чого не можна допускати.

11.5.3. Вхід в перериваль ну програму

При поступленні запиту переривання системи переривання програм повинна спочатку визначити допустимий момент переривання і початкову адресу переривальної програми. При цьому, для забезпечення повернення до перериваної програми, вміст елементів пам'яті комп'ютера, в першу чергу лічильника команд та регістра станів, в момент її переривання повинен бути збережений з тим, щоб після завершення виконання переривальної команди відновити цей вміст та продовжити виконання перериваної програми.

Існують наступні способи визначення допустимого моменту переривання:

- включення в кожну команду програми розряду дозволу переривання. В цьому випадку програміст сам слідкує за тим, щоб переривання не зашкодило виконанню перериваної програми. Недолік цього способу - великий час реакції tp, а також створення додаткових клопотів програмісту;
- переривання дозволено після закінчення будь-якої команди. Недоліки цього способу: необхідність запам'ятовувати стани програмно доступних регістрів процесора та комірок пам'яті, включаючи і кеш пам'ять, а також малий час реакції tp, який є рівним часу виконання команди;
- переривання дозволено після кожного такту команди. Тут час реакції tp є мінімальним. Недолік цього способу: необхідність збереження станів не тільки програмно доступних регістрів та комірок пам'яті, включаючи і кеш пам'ять, а й недоступних програм і вузлів, зокрема вмісту лічильника тактів.

Другий і третій способи використовуються в комп'ютерах найчастіше. При цьоіну потрібно зазначити, що для їх реалізації в процесор необхідно ввести спеціальні апаратні та програмні засоби для проведення операцій із збереження та відновлення після завершення переривальної програми вмісту програмно доступних регістрів та комірок основної пам'яті, в яких зберігалася програма, включаючи і кеш пам'ять. Як ми вже наголошували раніше, для обробки переривань в пам'яті мікрокоманд пристрою керування для цього записані спеціальні мікропрограми.

Після визначення допустимого моменту переривання потрібно визначити початкову адресу переривальної програми з тим, щоб приступити до її виконання. Зрозуміло, що джерел переривання може бути декілька. Це, наприклад, запити від пристроїв введення-виведення, від джерела живлення, від засобів захисту пам'яті, від АЛП. Тому система переривання програм повинна вміти розпізнати причину переривання та вказати на початкову адресу програми, яка обробляє це переривання. Існують наступні способи визначення початкової адреси переривальної програми:

- програмне розпізнавання причин переривання. Запит переривання спочатку фіксується, а потім переривальна програма проводить опитування джерел переривання;
- апаратне розпізнавання причин переривання. Тут кожному джерелу перериванння, або групі джерел переривання, ставиться у відповідність своя адреса початку переривальної програми. Кількість початкових адрес рівна кількості рівнів системи переривання. Для реалізації цього способу потрібні додаткові апаратні засоби.

Зазвичай використовують комбінацію обох описаних способів, тобто апаратно-програмне розпізнавання причин переривання.

Нарис. 11.10 наведено пристрій фіксування запитів переривання та формування початкової адреси переривальної програми.

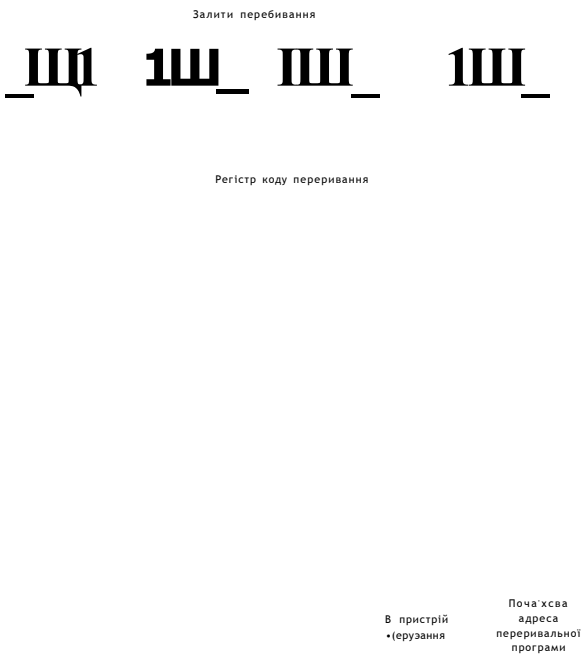


Рис. 11.10. Пристрій фіксування запитів переривання та формування початкової адреси переривальної програми

Запити поступають в систему переривання по спеціальній шині запиту переривання. Запити з однаковим пріоритетом поступають на ті ж самі логічні схеми АБО. Сигнал переривання сигналізує про те, що даний пристрій потребує переривання програми. Всі сигнали переривання зазвичай фіксуються в регістрі коду переривання РгКП. Оскільки сигнали переривання є асинхронними, будь-який пристрій може послати переривання в довільний момент часу, включаючи і моменти, коли обробляється попереднє переривання. Для запобігання виконання переривання програми до збереження необхідної інформації, потрібної для повернення до виконання перериваної програми, до складу системи переривання програм вводиться тригер блокування переривання (ТБП). Крім того, в системі переривання існує тригер очікування переривання (ТОП), який інформує пристрій керування про потребу проведення переривання. Тригер блокування переривання блокує схеми І, та не дозволяє фіксацію запиту переривання в тригері очікування переривання. За значенням вмісту регістра коду переривання дешифратор формує початкову адресу переривальної програми. Мікронакази скидання регістра РгКП та тригерів ТБП і ТОП формує пристрій керування комп'ютера.

11.5.4. Пріоритетне обслуговування переривання

Джерел переривання в комп'ютері є багато, і запити від них можуть поступати одночасно, тому потрібно встановити порядок їх обслуговування, тобто пріоритетність.

В системі переривання програм є два типи пріоритетів:

- пріоритет між запитами переривань;
- пріоритет між переривальними програмами.

Перший пріоритет потрібний головним чином для селекції: вибирається один запит з багатьох, оскільки комп'ютер не завжди може відразу прийняти декілька запитів. Він може бути реалізований методом послідовного пошуку, або шляхом паралельного дешифрування всіх запитів, та вибору з них одного за якимось правилом, наприклад, з найменшим номером серед прийнятих запитів.

Другий пріоритет визначає фактичний порядок, в якому переривальні програми будуть виконуватись.

Вибір пріоритету програм в багаторівневій системі переривання програм є досить складною задачею. Часто пріоритетність програм потрібно поміняти. Тому встановлен ня пріоритетів має бути програмно керованим.

В сучасних комп'ютерах використовується метод встановлення програмно-керованого пріоритету, який забезпечує присвоєння пріоритету програмістом, який називають маскою переривань. Маска переривань - це двійкове число, кожний розряд якого ставиться у відповідність одному із рівнів переривання і дозволяє (якщо його значення рівне 1), або забороняє (якщо його значення рівне 0) переривання від запитів, які належать до даного рівня. Існує регістр маски переривань, вміст якого змінюється програмним шляхом. Значення розрядів цього регістра поступають на логічні схеми І пристрою фіксування запитів переривання та формування початкової адреси переривальної програми, як це показано на рис. 11.11, та дозволяють, або забороняють, фіксацію запиту переривання в тригері очікування переривання ТОП. Для того, щоб значення в тригері очікування переривання не було змінене до його фіксації, в пристрій введено тригер блокування переривань ТБП.

Рис. 11.11. Пристрій фіксування запитів переривання та формування початкової адреси переривальної програми з врахуванням маски переривання

Для кожної переривальної програми може бути встановлена своя маска. Маска для всіх програм зберігається в основній пам'яті та засилається в регістр маски, якщо викликається до виконання відповідна програма.

Крім індивідуальних масок можуть бути і групові, в яких один розряд належить до декількох рівнів. Вищим ступенем в такій ієрархії масок може бути головний тригер, який виключає систему переривання повністю.

Якщо запит переривання заборонений маскою, він або ігнорується, або запам'ятовується відповідною апаратурою, про що може бути відповідна вказівка в масці переривання.

11.5.5. Організація повернення до перериваио) програми

Повернення до перериваної програми полягає у відновленні стану цієї програми, в якому вона була в момент переривання і який був збережений в момент входу в переривальну програму.

Інформація про біжучий стан перериваної програми ділиться на дві частини:

- основну, яка повинна запам'ятовуватись завжди при вході в переривальну програму;
- додаткову інформацію, необхідність запам'ятовування якої залежить від змісту переривальної програми.

До складу першої частини інформації про біжучий стан перериваної програми належать:

- вміст лічильника команд;
- стан тригера роботи комп'ютера (робота/зупинка);
- маска переривання;
- код переривання - двійковий номер джерела переривання;
- вміст регістрів захисту пам'яті.

Перераховані дані компонуються в так званий вектор переривання, який займає декілька комірок основної пам'яті.

В деяких комп'ютерах, наприклад в системі IBM 370, для збереження вектора переривання використовується регістр слова стану програми РгССП. В ньому зберігаються код переривання, маска переривання та інша важлива інформація. Початкова частина переривальної програми проводить запам'ятовування ССП перериваної програми і встановлює ССП переривальної програми. Заключна частина переривальної програми відновлює ССП перериваної програми. Під час заміни ССП будь-яке нове переривання забороняється.

До складу другої частини інформації про біжучий стан перериваної програми належать вміст індексних та інших програмно доступних регістрів. В більшості випадків сам програміст визначає, що тут необхідно зберігати.

11.5.6. Введення-виведення за перериваннями

При програмно-керованому введенні-виведенні процесор чекає, коли зовнішній пристрій буде готовий до обміну. В цей час він не зайнятий роботою. Інший підхід - процесор постійно працює, а при необхідності введення-виведення з пристрою введення-виведення до нього по спеціальній лінії поступає запит переривання. Це і є введення-виведення за перериваннями, реалізація якого покладена на систему переривання програм, описану вище. Тому організація введення-виведення за перериваннями не відрізняється від виконання інших програм, які виконуються в комп'ютері з за діянням системи переривання програм.

Коли поступає переривання від зовнішнього пристрою, процесор припиняє виконання поточної перериваної програми і переходить до програми обслуговування цього переривання, тобто до переривальної програми, наприклад, це може бути програма друкування, як це показано на рис. 11.12.

Рис. 11.12. Обслуговування переривання підпрограми друкування

Якщо сигнал переривання поступав при виконанні і-1 команди перериваної програми, її виконання закінчується, запам'ятовується вміст програмного лічильника і процесор переходить до виконання переривальної програми друкування, після закінчення

якої знову загружає в програмний лічильник раніше запам'ятоване значення, і продовжує виконання поточної перериваної програми.

Коли процесор приступає до обслуговування переривання, він повідомляє про це відповідний пристрій введення-виведення, який після цього знімає запит.

Введення-виведення за перериваннями можна представити як інверсію програмованого введення-виведення. Замість того, щоб процесор безупинно опитував приєднані до нього пристрої чи вони хочуть вводити інформацію, самі ці пристрої говорять процесору, коли вони мають дані для посилання. Процесор обслуговує інші задачі поки пристрій введення-виведення не виставить переривання. Про наявність переривання повідомляє відповідний прапорець в регістрі стану процесора, який називається прапорцем переривання.

Як тільки прапорець переривання встановлений, операційна система перериває виконання поточної програми, зберігаючи стан програми і її змінну інформацію. Система переривання вибирає адресний вектор, який указує на адресу службової програми введення-виведення. Після того, як процесор завершив обслуговування введення-виведення, він відновлює інформацію, яка була збережена від перерваної програми, і виконання програми поновлюється.

11.6. Прямий доступ до пам'яті

Для звільнення процесора від організації введення-виведення з синхронними пристроями введення-виведення, передача даних з яких здійснюється з частотою, незалежною від процесора і не зв'язаною з його частотою, використовується прямий доступ до пам'яті, коли передача блоків даних здійснюється прямо між пам'яттю і пристроями введення-виведення без участі процесора. Наприклад, магнітний диск не можна зупинити після записування символу.

В цьому випадку необхідний контролер прямого доступу до пам'яті, який керує взаємодією пам'яті та пристроїв введення-виведення без участі процесора.

На рис 11.13 показаний двоканальний контролер прямого доступу до пам'яті, який керує зв'язком основної пам'яті з дисковою пам'яттю і високошвидкісним принтером.

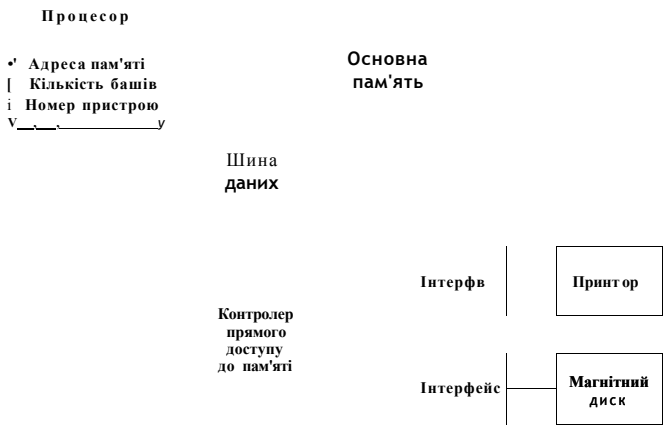


Рис. 11.13. Двоканальний контролер прямого доступу до пам'яті

До складу контролера входять:

- регістр даних;
- регістр адреси;
- лічильник слів;
- регістр команд;
- регістр станів;
- логіка керування.

Ці вузли мають свої адреси для засилання в них інформації з процесора. Кількість таких вузлів кратна кількості каналів.

Для початку передавання даних між пам'яттю і пристроєм введення-виведення в контролер записується наступна інформація:

- адреса пам'яті;
- кількість слів;
- адреса даних в пристрої введення-виведення (на диску);
- виконувана функція (запис/зчитування).

Ця інформація поступає в контролер з процесора. Коли передача закінчена, цей факт реєструється в регістрі станів контролера. Тут також записується інформація про помилки при передачі даних. Після закінчення введення-виведення, чи при наявності помилки, система прямого доступу до пам'яті повідомляє про це процесор відповідним сигналом переривання.

Як це видно з рисунку, контролер прямого доступу до пам'яті та процесор розділяють шину пам'яті. Тільки один з них в той же час може мати контроль над шиною. Загалом, пристрої введення-виведення мають перевагу над процесором при взаємодії з пам'яттю, тому що багато пристроїв введення-виведення діють з суттєвими часовими обмеженнями. Якщо вони не проводять ніякої діяльності в межах вказаного періоду, вони припиняють роботу і переривають процес введення-виведення. Оскільки пересилання інформації від пристроїв введення-виведення здійснюється блоками не досить довгої тривалості, це не суттєво впливає на продуктивність процесора.

7 7.7. Введення-виведення під керуванням периферійних процесорів

11.7.1. Принципи введення-виведення під керуванням периферійних процесорів

Програмно-кероване введення-виведення передбачає передачу одного байта за один раз. Кероване перериваннями введення-виведення також може маніпулювати даними по одному байту за один раз або малими блоками, залежно від виду пристрою, що бере участь у введенні-виведенні. Повільніші пристрої, як наприклад клавіатура, генерують більшу кількість переривань відносно кількості переданих байтів, ніж диски або принтери. Методи прямого доступу до пам'яті є блочно-орієнтованими. Тут переривання роботи процесора відбувається лише після завершення (або помилки) передачі групи байтів. Після того, як система прямого доступу до пам'яті сигналізує про завершення введення-виведення, процесор може надати їй адресу наступного блоку пам'яті, для зчитуван-

ня або запису. У випадку невдачі, обов'язок по вживанню відповідних заходів лежить на процесорі. Тому, введення системи прямого доступу до пам'яті вимагає лише трохи меншої участі процесора, ніж при керованому перериваннями введенні-виведенні. Такі підходи до організації введення-виведення допустимі для малих систем, орієнтованих на одного користувача, проте, вони не є ефективними для великих систем, орієнтованих на багатьох користувачів, як наприклад, великі універсальні комп'ютери - мейнфрейми. Більшість мейнфреймів використовують інтелектуальний вид інтерфейсу прямого доступу до пам'яті, відомого як канальне введення-виведення.

При використанні канального введення-виведення один або більше процесорів введення-виведення, які ще називають каналами, керують різними трактами введення-виведення. Тракти для "повільних" пристроїв, як, наприклад, термінали і принтери, можуть комбінуватися (мультиплексуватися), дозволяючи керування деякою кількістю цих пристроїв тільки одним процесором введення-виведення. В мейнфреймах фірми IBM процесор введення-виведення, який керує мультиплексованим канальним трактом, називають мультиплексним каналом (МПК). Канали для дисководів й інших "швидких" пристроїв називають селекторними каналами (СК).

Процесори введення-виведення оптимізують для організації виконання операцій введення-виведення. На відміну від схем прямого доступу до пам'яті, процесори введення-виведення мають здатність виконувати програми, які включають арифметично-логічні команди та команди переходу. На рис. 11.14 наведена спрощена конфігурація канального введення-виведення.

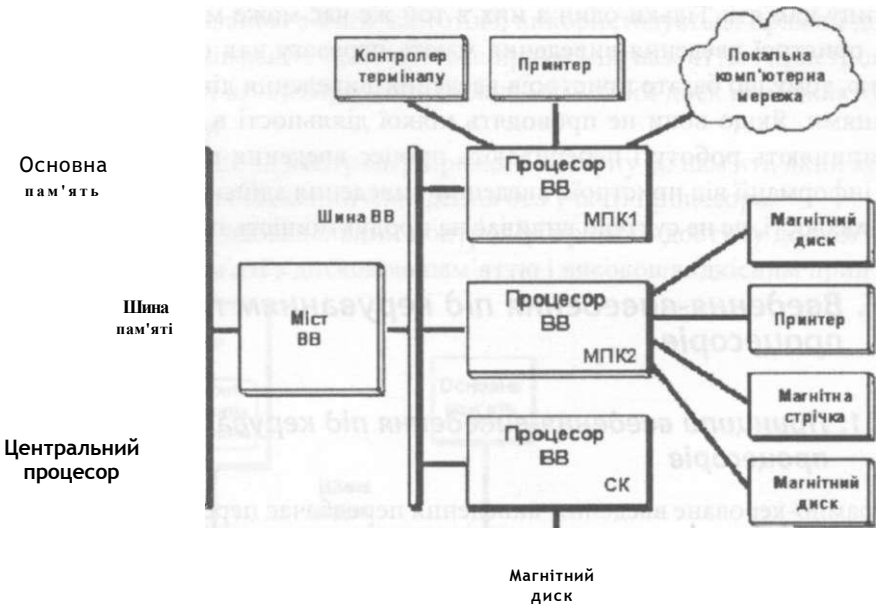


Рис. 11.14. Спрощена конфігурація канального введення-виведення

Процесори введення-виведення виконують програми, які розміщені центральним процесором в основній пам'яті. Ці програми, складаючись з серій слів команд каналу, включають не тільки фактичні команди пересилання, але і команди, які керують при-

строями введення-виведення. Названі команди включають різні види ініціалізацій пристрою, відторгнення сторінки принтером, команди перемотування стрічки та інші. Як тільки програма введення-виведення була розміщена в пам'яті, процесор комп'ютера видає команду старту підканалу, інформуючи процесор введення-виведення про місце розташування програми в пам'яті. Після того, як процесор введення-виведення завершив свою роботу, він розміщує інформацію про завершення в пам'яті та відправляє переривання центральному процесору комп'ютера. Центральний процесор тоді одержує інформацію про завершення і вживає заходи по виходу з переривання.

Головна відмінність між автономним прямим доступом до пам'яті і канальним введенням-виведенням полягає в більш широких можливостях процесора введення-виведення. Процесор введення-виведення забезпечує дотримання протоколів, видає команди пристрою, транслює коди зовнішньої пам'яті до кодів основної пам'яті, і може перенести повні файли або групи файлів незалежно від центрального процесора. Центральному процесору комп'ютера належить лише створити команди програми для операцій введення-виведення і вказати процесору введення-виведення, де їх знайти.

Подібно до автономного прямого доступу до пам'яті, процесор введення-виведення повинен забрати від центрального процесора цикли звернення до пам'яті. На відміну від автономного прямого доступу до пам'яті, системи введення-виведення обладнані окремими шинами введення-виведення, які допомагають звільнити центральний процесор від виконання введення-виведення. Копіюючи файл з диску на стрічку, наприклад, процесор введення-виведення використовує системну шину пам'яті тільки для вибору його команд з основної пам'яті. Остання частина передачі проводиться використовуючи лише шину введення-виведення. Завдяки широким функціональним можливостям та шинній ізоляції, канальне введення-виведення використовується в середовищах діалогової високопродуктивної обробки запитів, де його вартість і складність можуть бути виправдані.

7 7.7.2. *Причини застосування каналів введення-виведення*

Поява каналів введення-виведення викликана необхідністю:

- створення комп'ютерних систем із змінним складом обладнання;
- забезпечення паралельної роботи пристроїв введення-виведення по відношенню до інших пристроїв введення-виведення і до процесора;
- уніфікація програмування введення-виведення;
- забезпечення реакції обчислювальних засобів на багатоманітність ситуацій, які виникають в пристроях введення-виведення (готовність, збій і т. п.);
- узгодження швидкості роботи процесора і пристроїв введення-виведення;
- розвантаження процесора від виконання функцій керування обміном.

Канал (пристрій обміну, процесор введення-виведення) - спеціальний уніфікований пристрій керування вводом-виводом, який забезпечує обмін інформацією між ядром комп'ютера (процесором і основною пам'яттю), та периферійними пристроями з здійсненням паралельної роботи змінного набору периферійних пристроїв різних типів.

При цьому в комп'ютері повинні бути уніфіковані шини, схеми підключення, сигнали та алгоритми керування обміном, формат і множина команд процесора.

11.7.3. Функції каналів введення-виведення

Функції каналів введення-виведення залежать від типу комп'ютера. В великих комп'ютерних системах канал - самостійний в логічному відношенні пристрій, який працює за відповідною програмою. Він звільняє процесор від багатьох допоміжних операцій введення-виведення. Функції каналів у наведеному на рис. 11.15 форматі керуючого слова (КС) каналу.

КОП	ВЛК	ВЛД	ВПІ	ВПКП	ПА	N

Рис. 11.15. Формат керуючого слова каналу

До функцій каналів введення-виведення належать наступні:

1. Визначення типу операції введення-виведення. Тип операції введення-виведення задається кодом операції КОП.

2. Організація ланцюга операцій. Ланцюг операцій - це послідовність КС, кожне з яких описує нову операцію введення-виведення над новим масивом інформації. Для організації ланцюга операцій до складу КС вводять розряд вказівника ланцюга операцій ВЛО. Якщо ВЛО = 1, то після виконання КС вибирається нове КС, якщо ВЛО = 0, то дане КС є останнім в ланцюгу.

3. Визначення області пам'яті. Область пам'яті задається початковою адресою ПА і розміром області Х, тобто кількістю слів, починаючи від початкового слова.

4. Організація ланцюга даних. Ланцюг даних організується при роботі з масивами даних. Ланцюг даних - це послідовність з декількох КС. Після закінчення введення-виведення одного масиву даних, канал переходить до іншого КС. Для керування ланцюгом даних використовується двійковий розряд - вказівник ланцюга даних ВЛД, який приймає одне з двох значень - 0 або 1. Якщо вказівник ланцюга рівний 1, то канал вибирає з пам'яті наступне КС, якщо вказівник ланцюга рівний 0, то операція закінчується.

5. Забезпечення пропуску інформації. Записуються/зчитуються окремі частини масиву та пропускаються дані, які не є потрібними в конкретному випадку. Для реалізації пропуску інформації в керуюче слово каналу вводиться розряд вказівника пропуску інформації ВПІ. Якщо ВПІ = 0, то виконується КС, якщо ВПІ = 1, то запис/зчитування не проводиться, лише підраховуються дані.

Утворення ланцюгів даних і операцій можна комбінувати. В загальному випадку ланцюг КС - це програма каналу.

6. Організація переривання введення-виведення.

Канал інформує центральний процесор про хід введення-виведення, посилаючи запити переривання. Два основних види переривань - програмно-кероване переривання і переривання, яке викликане закінченням введення-виведення. На наявність програмно-керованого переривання вказує відповідний розряд в КС, який називають вказівником програмно-керованого переривання ВПКП. Якщо ВПКП = 1, то апаратура каналу формує запит переривання, виконуючи одночасно поточну операцію введення-виведення. Переривання, зв'язані з закінченням введення-виведення, формуються, якщо виконані всі операції в ланцюгу КС, або виявлені помилки.

11.7.4. Керуюча інформація каналу введення-виведення

Існує 4 види керуючої інформації каналу введення-виведення: команди центрального процесора, керуюче слово каналу, накази периферійним пристроям та коди стану периферійних пристроїв і каналів. На рис. 11.16 наведено закріплення керуючої інформації каналу за пристроями комп'ютера.

Рис. 11.16. Керуюча інформація каналу

Команди введення-виведення керують роботою центрального процесора. Послідовність команд введення-виведення - це програма центрального процесора по введенню-виведенню інформації. Формат команди введення-виведення наведено на рис. 11.17.

КОП	НК	НПВВ	АКС

Рис. 11.17. Формат команди введення-виведення

До складу команди центрального процесора входять наступні поля: КОП - поле коду операції, яке задає операції запуску введення-виведення, припинення введення-виведення, опит стану каналу, опит стану пристрою введення-виведення; поле номера каналу НК; поле номера пристрою введення-виведення НПВВ; поле адреси керуючого слова каналу АКС.

Код операції керуючого слова каналу, який наведено на рис. 11.15, задає операцію введення інформації в основну пам'ять; операцію зчитування інформації з основної пам'яті в периферійний пристрій, опит стану периферійного пристрою, операцію видачі в периферійний пристрій наказу типу "перехід в каналі".

Наказ - це частина КС (або код, який адресується КС) - код, який приймається з каналу і виконується периферійним пристроєм. Наказ містить інформацію, специфічну для даного периферійного пристрою (магнітного диску, магнітної стрічки, оптичної пам'яті, давана і т. д.).

Коди стану інформують центральний процесор про стан каналу та периферійного пристрою (справність, готовність і т. д.).

11.7.5. Мультиплексный та селекторний канали введення-виведення

За здатністю до одночасного обслуговування декількох периферійних пристроїв розрізняють два види каналів: мультиплексний і селекторний.

Мультиплексний канал дозволяє одночасно обслуговувати декілька паралельно працюючих периферійних пристроїв. Кожний периферійний пристрій працює з каналом в визначений сеанс зв'язку. Для різних периферійних пристроїв існують свої пріоритети. Тому при одночасному запиті на обслуговування може бути черга. Звідси зрозуміло, що

мультиплексний канал призначений для роботи з повільними пристроями без втрати інформації.

Апаратні засоби мультиплексного каналу можна розділити на дві частини. До першої частини належать апаратні засоби, призначені для обслуговування окремих периферійних пристроїв. Ці засоби називають підканалом. До другої частини належать апаратні засоби, загальні для всіх периферійних пристроїв, які розділяються між ними в часі. Кількість підканалів визначає максимальна кількість одночасно працюючих периферійних пристроїв.

В цілому мультиплексний канал - це процесор з усіма властивими йому елементами. Тобто до його складу входять пристрій керування, АЛП, набір регістрів та швидка оперативна пам'ять. При цьому пам'ять мультиплексного каналу ділиться на підканали (рис. 11.18). Підканал це - пам'ять, що зберігає команди введення-виведення та дані для окремих периферійних пристроїв. Загальні апаратні засоби (обладнання) - це всі інші засоби процесора. До загальних засобів належать пристрій керування та регістри мультиплексного каналу, до яких належать наступні: регістр номера підканалу, регістр команди введення-виведення, регістр керуючого слова, регістр адреси керуючого слова, регістри зв'язку з периферійними пристроями та основною пам'яттю, лічильник слів.

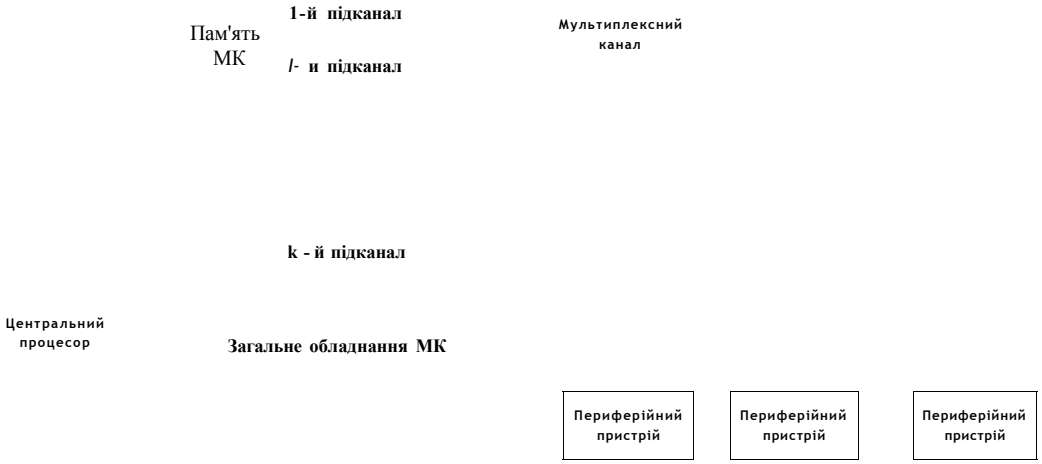


Рис. 11.18. Мультиплексний канал

В мультиплексному каналі може бути можливість монопольного обслуговування одного периферійного пристрою. В цьому режимі один з підканалів повністю займає канал. Це дозволяє підключити до мультиплексного каналу швидкодіючі периферійні пристрої.

Селекторний канал призначений для монопольного обслуговування одного периферійного пристрою. Він обслуговує швидкодіючі пристрої. Фактично селекторний канал має один підканал. Тому він є простішим, ніж мультиплексний канал. До його складу входять, крім пам'яті, пристрій керування та регістри, до яких належать наступні: регістр номера периферійного пристрою, регістр команди введення-виведення, регістр керуючого слова, регістр адреси керуючого слова, регістри зв'язку з периферійними пристроями та основною пам'яттю, лічильник слів.

В мейнфреймах зазвичай є один мультиплексний та кілька селекторних каналів, причому кількість підканалів в мультиплексному каналі рівна 256 і більше.

Залежно від фізичної реалізації канали ділять на автономні і вбудовані. Вбудований канал не має власних апаратних засобів і реалізує свої функції за допомогою апаратних засобів процесора. Виділяється тільки вузол зв'язку з периферійними пристроями. Вбудовані канали діляться на канали з затриманим і негайним доступом. Канал з затриманим доступом виконує зв'язок з периферійним пристроєм тільки в проміжках між командами процесора. З негайним - між мікрооперaціями процесора. В цьому випадку необхідно зберігати вміст регістрів процесора.

11.8. Короткий зміст розділу

Пристрої введення-виведення призначені для введення інформації в комп'ютер та виведення інформації з комп'ютера. Існує широкий спектр пристроїв введення-виведення, зокрема клавіатура, цифрова відеокамера, сенсорні екрани, світлове перо, мишка, аналогові пристрої з цифровим входом та виходом, електронно-променева трубка, рідкокристалічний дисплей, принтер та інші.

Оскільки в комп'ютері може бути багато пристроїв введення-виведення, при зверненні до них, їх потрібно розпізнати. Спосіб розпізнавання залежить від способу їх підключення до процесора. В розділі наведено пояснення способів розпізнавання пристроїв введення-виведення з використанням шини введення-виведення, лінії активації та скритого пам'яттю введення-виведення. Подана схема та описані функції інтерфейсної схеми пристроїв введення-виведення.

Наведено чотири загальних методи керування введенням-виведенням та пояснена суть, переваги та недоліки програмно керованого введення-виведення.

Оскільки важливою складовою частиною засобів керування введенням-виведенням є система переривання програм, яка призначена для реакції на програмно-незалежні події, в розділі наведений її детальний опис. Переривання програм - це властивість комп'ютера тимчасово переривати виконання біжучої програми при виконанні деяких подій і передавати керування програмі, яка спеціально передбачена для даної події. Введено основні поняття та характеристики системи переривання програм: запити переривання, переривальні програми, перериванні програми, час реакції, час обслуговування переривання, глибина переривання. Розглянуті програмно-апаратні засоби, які забезпечують визначення допустимого моменту переривання і початкової адреси переривальної програми при поступленні запиту переривання, а для забезпечення повернення до перериваної програми забезпечують збереження вмісту елементів пам'яті комп'ютера в момент її переривання, та, після завершення виконання переривальної команди, відновлюють цей вміст та надають комп'ютеру можливість продовжити виконання перериваної програми.

Описаний прямий доступ до пам'яті, його переваги та недоліки, та описана організація введення-виведення під керуванням периферійних процесорів (каналів). Наведені причини появи каналів введення-виведення, їх функції, керуюча інформація каналів введення-виведення. Описані функції та склад селекторного та мультиплексного каналів.

11.9. Література для подальшого читання

Питанням організації введення-виведення інформації присвячена значна кількість літератури. В першу чергу слід назвати навчальну літературу, зокрема підручники з архітектури комп'ютера [1-8]. Крім того, ці питання відображені в великій кількості спеціальної літератури, де описані серійні комп'ютери різних фірм-виробників, яку можна знайти на веб-сторінках цих фірм.

11.10. Література до розділу 11

1. Каган Б. М. Электронные вычислительные машины и системы. - М.: Энергия, 1979. - 528 с

2. Каган Б. М., Каневский М. М. Цифровые вычислительные машины и системы. - М.: Энергия, 1974. - 680 с

3. Угрюмов Е. П. Цифровая схемотехника. - СПб.: БХВ - Санкт-Петерберг, 2000. - 528 с

4. Справочник по цифровой вычислительной технике. Б. Н Малиновский и др. - К.: Техніка, 1980.-320 с

5. D. Patterson, J. Hennessy. Computer Architecture. A Quantitative Approach. Morgan Kaufmann Publishers, Inc. 1996.

6. Patterson, D. A., & Hennessy, J. L. *Computer Organization and Design, The Hardware/Software Interface,* 2nd ed , San Mateo, CA: Morgan Kaufmann, 1997.

7. Stallings, W. *Computer Organization and Architecture,* 5th ed., New York, NY: Macmillan Publishing Company, 2000.

8. Tanenbaum, Andrew. *Structured Computer Organization,* 4th ed., Upper Saddle River, NJ: Prentice Hall, 1999.

11.11. Питання до розділу 11

1. Назвіть пристрої введення інформації в комп'ютер
2. Назвіть пристрої виведення інформації з комп'ютера
3. ′ .
4.
5. Поясніть спосіб розпізнавання пристроїв введення-виведення з використанням шини введення-виведення
6. Поясніть спосіб розпізнавання пристроїв введення-виведення з використанням лінії активації
7. Поясніть суть скритого пам яттю введення-виведення
8. Наведіть схему та назвіть функції інтерфейсної схеми пристроїв введення-виведення
9. Які є чотири загальних методи керування введенням-виведенням?
10. Поясніть суть, переваги та недоліки програмно-керованого введення-виведення
11. Поясніть суть, переваги та недоліки керованого перериваннями введення-виведення
12. Які події належать до програмно незалежних?
13. Які події належать до програмно залежних?
14. Як комп'ютер реагує на програмно визначені події?
15. Як комп'ютер реагує на програмно незалежні події?
16. Для чого є в комп'ютері система переривання програм?
17. Як називають сигнали, які сповіщають про появу програмно незалежних подій?
18. Як називають програми, на виконання яких є запити?
19. Як називають програми, які виконувались до появи запитів?

20. Наведіть часову діаграму процесу переривання.

21. Наведіть основні характеристики системи переривання програм.

22. Назвіть способи визначення допустимого моменту переривання.

23. Для чого і як визначається початкова адреса переривальної програми?

24. Назвіть способи визначення початкової адреси переривальної програми.

25. Як працює пристрій фіксування запитів переривання та формування початкової адреси переривальної програми?

26. Для чого до складу системи переривання програм вводиться тригер блокування переривання?

27. Де фіксуються переривання?

28. Для чого потрібно встановлювати порядок обслуговування запитів переривання?

29. Для чого потрібний пріоритет між запитами переривань?

30. Для чого потрібний пріоритет між переривальними програмами?

31. Що таке маска переривань та як вона використовується?

32. В чому полягає повернення до перериваної програми?

33. Поясніть суть, переваги та недоліки прямого доступу до пам'яті.

34. Поясніть суть, переваги та недоліки введення-виведення під керуванням периферійних процесорів (каналів).

35. Що таке канал введення-виведення?

36. Які причини появи каналів введення-виведення?

37. Назвіть функції каналів введення-виведення.

38. Керуюча інформація каналів введення-виведення?

39. Назвіть функції та склад селекторного та мультиплексного каналів.

40. Яка різниця між автономними та вбудованими каналами?

Розділ 12

[ашшшшШ[ЩшІ[ті[ві[ві[ші[в][ШШШШ

Тіаралеміші шмпу'юшєрпі системи/

До цих пір ми знайомилися в основному з однопроцесорними комп'ютерами з архітектурою Джона фон Неймана. Однопроцесорні комп'ютери досягли високих показників продуктивності завдяки технологічним та архітектурним досягненням, які були детально розглянуті в попередніх розділах. Але ріст їх продуктивності стримується як фізичними, технологічними обмеженнями, так і обмеженими можливостями покращення традиційної архітектури Джона фон Неймана. Як ми бачили з попередніх розділів, такі архітектурні рішення, як спрощена система команд, конвеєрна обробка команд та даних, суперконвеєрна та суперскалярна обробка команд, використання довгого формату команди, векторна обробка команд та природній паралелізм, практично виводять однопроцесорні комп'ютери до граничної межі, коли використовуються всі потенційні можливості паралелізму архітектури комп'ютера Джона фон Неймана.

У цьому розділі будуть розглянуті питання подальшого підвищення продуктивності комп'ютерів шляхом створення паралельних комп'ютерних систем. До таких систем належать як багатопроцесорні комп'ютерні системи, в яких використовується паралелізм розподілу задач на велику кількість вузлів обробки, так і комп'ютерні мережі, які мають іншу форму паралелізму - мережу структурно автономних комп'ютерів. Деякі з розглянутих підходів уже знайшли місце в сучасних комп'ютерах. Інші підходи розглядаються як основа для покращення характеристик перспективних комп'ютерів.

12.1. Використання принципів паралельної обробки інформації в архітектурі комп'ютера

Сьогодні використання принципів паралельної обробки інформації в архітектурі комп'ютерів є звичним. Практично всі сучасні комп'ютери використовують той або інший вид паралельної обробки інформації. На презентаціях нових мікросхем і в інформаційних матеріалах виробників це подається як останнє слово техніки і передовий край науки, і це дійсно так, якщо розглядати реалізацію цих принципів у мініатюрних рамках одного кристалу.

Водночас, ідеї паралельної обробки інформації з'явилися досить давно. Вони впроваджувалися в найбільш передових, дорогих, а тому одиничних, комп'ютерах свого часу. Пізніше, після належного відпрацювання технології і здешевлення виробництва, вони використовувалися в комп'ютерах^середнього класу, і нарешті сьогодні вони в повному обсязі втілюються у робочих станціях і персональних комп'ютерах.

Прослідкуємо за впровадженням значущих нововведень в архітектурі комп'ютера з точки зору паралельної обробки інформації, починаючи практично з часу створення перших комп'ютерів.

Паралельна пам'ять та паралельна арифметика. Всі найперші комп'ютери (EDSAC, EDVAC, UNIVAC) мали послідовну пам'ять, із якої слова зчитувалися послідовно біт за бітом. Першим комерційно доступним комп'ютером, що використовував паралельну пам'ять і паралельну арифметику, став комп'ютер IBM 701, який поступив на ринок у 1953 році. В більш популярній моделі IBM 704, що поступила на ринок у 1955 році, крім названого, була вперше застосована пам'ять на феритових сердечниках і апаратний арифметичний пристрій з рухомою комою.

Незалежні процесори введення-виведення. В перших комп'ютерах процесори не лише обробляли інформацію, а й керували введенням-виведенням. Проте швидкість роботи найшвидшого зовнішнього пристрою, а в ті часи це була магнітна стрічка, була в 1000 разів меншою швидкості процесора, тому під час операцій введення/виведення процесор фактично простоював. У 1958 р. до комп'ютера IBM 704 приєднали 6 незалежних процесорів введення/виведення, що після одержання команд від центрального процесора могли працювати паралельно з ним, а сам комп'ютер перейменували в IBM 709. Дана модель була досить вдалою, підтвердженням чого є те, що було продано біля 400 її екземплярів, які знаходились в експлуатації близько 20 років.

Передбачення та розшарування пам'яті. У 1961 році фірма IBM закінчила розробку комп'ютера STRETCH, що мав дві принципово важливі особливості: передбачення для вибірки команд і розшарування пам'яті на два блоки для узгодження низької швидкості вибірки з пам'яті та швидкості виконання операцій.

Конвеєр команд. Вперше конвеєрний принцип виконання команд був використаний у комп'ютері ATLAS, розробленому в Манчестерському університеті в 1962 році. Виконання команд розбито на 4 стадії: вибірка команди, обчислення адреси операнда, вибірка операнда і виконання операції. Конвеєризація дозволила зменшити час виконання команд із 6 мкс до 1,6 мкс. Створення даного комп'ютера зробило величезний вплив як на подальший розвиток архітектури комп'ютера, так і на розвиток системного програмного забезпечення: у ньому вперше використана мультипрограмна операційна система, що використовувала віртуальну пам'ять і систему переривань.

Незалежні операційні пристрої. В 1964 році фірма Control Data Corporation (CDC) при особистій участі одного з її фундаторів Сеймура Крея випускає комп'ютер CDC-6600 - перший комп'ютер, у якому використовувалося декілька незалежних операційних пристроїв. Для порівняння із сьогоднішнім днем приведемо деякі характеристики цього комп'ютера:

- час такту рівний 100 не;
- продуктивність рівна 2-3 млн операцій за секунду;
- основна пам'ять розбита на 32 блоки по 4096 60-розрядних слів;
- цикл пам'яті рівний 1 мкс;
- 10 незалежних операційних пристроїв.

Комп'ютер CDC-6600 мав великий успіх на ринку, суттєво потіснивши комп'ютери фірми IBM.

Конвеєрні незалежні операційні пристрої. В 1969 році фірма CDC випускає комп'ютер CDC-7600 із вісьмома незалежними конвеєрними операційними пристроями, тим

самим забезпечивши поєднання паралельної та конвеєрної обробки даних. Основні характеристики цього комп'ютера:

- такт рівний 27,5 не;
- продуктивність рівна 10-15 млн операцій за секунду;
- 8 конвеєрних операційних пристроїв;
- дворівнева пам'ять.

Матричні процесори. В 1967 році були розпочаті роботи над створенням матричного процесора ILLIAC IV, проект якого передбачав досягнення наступних характеристик: 256 процесорних елементів (ПЕ) - 4 квадранти по 64 ПЕ, з можливістю реконфігурації на 2 квадранти по 128 ПЕ, або на 1 квадрант із 256 ПЕ, такт рівний 40 не, продуктивність рівна 1 млрд. операцій з рухомою комою за секунду. В якості пристрою керування був використаний універсальний комп'ютер із невеликою продуктивністю. Кожний ПЕ мав власний АЛП з повним набором команд та пам'ять ємністю 2К 64-розрядних слів з циклом 350 не. Комп'ютер був введений в експлуатацію у 1974 році в складі лише 1 квадранта, такт 80 не, реальна продуктивність наблизилась до 50 млн операцій з рухомою комою за секунду. Однак даний проект суттєво вплинув на архітектуру наступних комп'ютерів, побудованих за схожим принципом, які дістали назву масивно-паралельних комп'ютерів із розподіленою пам'яттю. Ідея побудови комп'ютерів цього класу наступна: беруться серійні мікропроцесори зі своєю локальною пам'яттю та з'єднуються за допомогою деякого комунікаційного середовища. Переваги такої архітектури наступні: якщо потрібно підвищити продуктивність, то збільшується кількість процесорів, якщо обмежені фінанси або заздалегідь відома необхідна продуктивність, то легко підібрати оптимальну конфігурацію і т. п.

Проте є і суттєвий недолік. Справа в тому, що міжпроцесорна взаємодія в комп'ютерах цього класу йде набагато повільніше, ніж відбувається локальне опрацювання даних самими процесорами. Саме тому написати ефективну програму для таких комп'ютерів дуже складно.

За цим принципом були побудовані комп'ютери PEPE, BSP, ICL DAP. До даного класу можна також віднести комп'ютери Intel Paragon, IBM SP1, Parsytec, у якійсь мірі IBM SP2 і CRAY T3D/T3E. До цього ж класу можна віднести і мережі комп'ютерів.

Векторно-конвеєрні комп'ютери. У 1972 році С. Крей залишає фірму CDC і засновує власну компанію Cray Research, що у 1976 р. випускає перший векторно-конвеєрний комп'ютер CRAY-1 з наступними характеристиками: час такту рівний 12,5 не, 12 конвеєрних операційних пристроїв, пікова продуктивність - 160 мільйонів операцій за секунду, основна пам'ять до 1Мслова (слова 64-розрядні), цикл пам'яті рівний 50 не. Головним нововведенням є використання векторних команд, що працюють із масивами цілих чисел і дозволяють ефективно використовувати конвеєрні операційні пристрої, а також ієрархія пам'яті. Ієрархія пам'яті прямого відношення до паралелізму не має, проте, безумовно, належать до тих особливостей архітектури комп'ютерів, що мають велике значення для підвищення їхньої продуктивності (згладжування різниці між тактом роботи процесора і часом вибірки з пам'яті). Основні рівні: регістри, кеш пам'ять, основна пам'ять, дискова пам'ять. Час вибірки по рівнях пам'яті від дискової до регістрів зменшується, вартість у перерахунку на 1 слово (байт) росте. В даний час подібна ієрархія підтримується і на персональних комп'ютерах.

Паралельні комп'ютери із спільною пам'яттю. З розвитком архітектур та технологій виробництва компонентів комп'ютера, в тому числі пам'яті, з'явився новий клас комп'ютерів - паралельні комп'ютери із спільною пам'яттю. Вся основна пам'ять таких комп ютерів розділяється декількома однаковими процесорами. Хоча кількість процесорів, що мають доступ до спільної пам'яті, за чисто технічними причинами не можна зробити великим, до даного напрямку входить багато сучасних багатопроцесорних систем, наприклад, комп'ютери HP Exemplar і Sun StarFire.

Зрозуміло, що в багатьох випадках у комп'ютерах використовується комбінація описаних технічних рішень. З декількох процесорів (традиігійних або векторно-конвеєрних) і спільної для нихпам'яті формується обчислювальний вузол. Якщо отриманої обчислювальної потужності недостатньо, то об'єднується декілька таких вузлів швидкісними каналами зв'язку. Подібну архітектуру називають кластерною, і за таким принципом побудовані комп'ютери CRAY SV1, HP Exemplar, Sun StarFire, NEC SX-5, останні моделі IBM SP2 й інші.

Системи з масовою паралельною обробкою інформації. Після початкового домінування систем, що використовували повільні процесори з складною системою команд та розшаровану пам'ять, фірми Sun, HP, DEC, SGI, NCR/ATT, Tandem, Pyramid, IBM та CRI розпочали випуск багатопроцесорних систем, здатних до масштабування, що базуються на процесорах із складною системою команд. Наведемо приклади сучасних систем цього класу.

Суперкомп'ютер Intel Paragon. На рис. 12.1. подано організацію суперкомп'ютера Intel Par^ agon із так званою архітектурою на основі пересилання повідомлень. Суперкомп'ютер побудовано як тривимірне об'єднання множини окремих вузлів (топологія гіперкуба). Кожен вузол містить два або більше процесорів і860, які утворюють систему з спільною пам'яттю, та вбудовану підтримку зовнішньої мережі із швидкодією обміну з мережею 175 МВ за секунду. Кожен вузол має ще індивідуальний зв'язок із власною підсистемою пам'яті з використанням прямого доступу та шини кеш пам'яті. Це дозволяє ефективно пересилати великі пакети інформації від кожного вузла до мережі та зворотно. Продук-

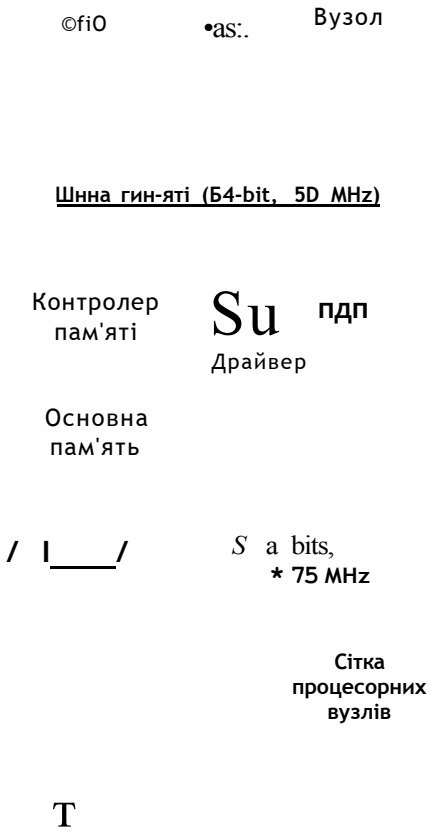


Рис. 12.1. Суперкомп'ютер Intel Paragon (подано двовимірну проекцію тривимірної мережі)

тивність системи складає 143 GFLOPS на тесті LINPACK (матричні тести лінійної алгебри), що на порядок перевищує продуктивність машин CRAY.

Суперкомп'ютер Blue Gene фірми IBM. Найшвидший на 2007 рік суперкомп'ютер, створення якого фірма IBM проголосила у грудні 1999 року, отримав назву Blue Gene. Він спроможний виконувати понад один квадрильйон операцій на секунду (1PFLOPS = 1000 TFLOPS= 1000000 GFLOPS). Blue Gene є майже в мільйон разів продуктивнішим від ПК. Ієрархічний дизайн Blue Gene наведено на рис. 12.2.

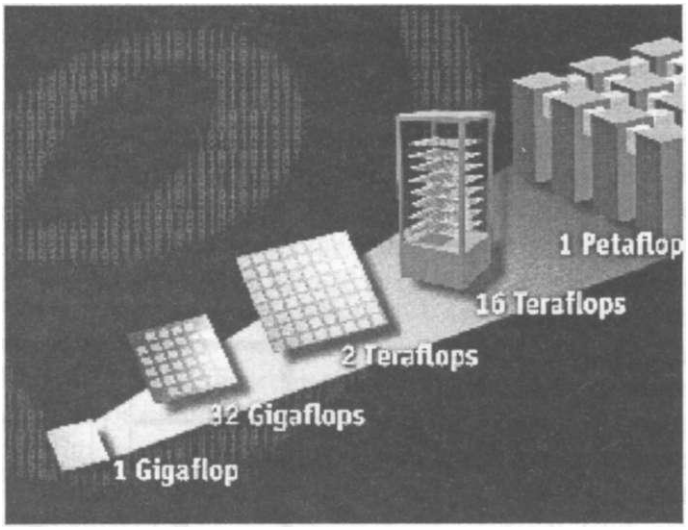


Рис. 12.2. Ієрархічний дизайн Blue Gene

Blue Gene складено з мільйона процесорів, кожний з яких має продуктивність 1 GFLOPS. 32 таких надшвидких процесори інтегровано до одного кристалу (32 GFLOPS). Компактна (два фути на два фути) плата вміщує 64 щойно зазначених кристали (2 TFLOPS). Апаратна частина такого друкованого вузла є продуктивнішою від відомого рекордсмена, суперкомп'ютера ASCI міністерства енергетики СІЛА, який займає 8000 квадратних футів площі. 11 друкованих вузлів збирають до одної монтажної 6-ти футової шафової конструкції, яка має продуктивність 16 TFLOPS. Фінальну конструкцію, що займає площу, меншу від 2000 кв. футів, утворено сполученням 64 шаф. Разом отримується продуктивність 1 PFLOPS= 1000 TFLOPS.

12.2. *Вибір кількості процесорів у багатопроцесорній системі*

Потрібно відзначити, що збільшення кількості процесорів у багатопроцесорній системі не завжди пропорційно зменшує час вирішення задачі. На рис. 12.3 наведено залежність прискорення від кількості задіяних процесорів при розв'язанні однієї задачі молекулярної динаміки на багатопроцесорній системі Intel Paragon. Параметром є версія програмного коду. Бачимо, що монотонне зростання прискорення досягнуто лише на другій версії паралельної програми.

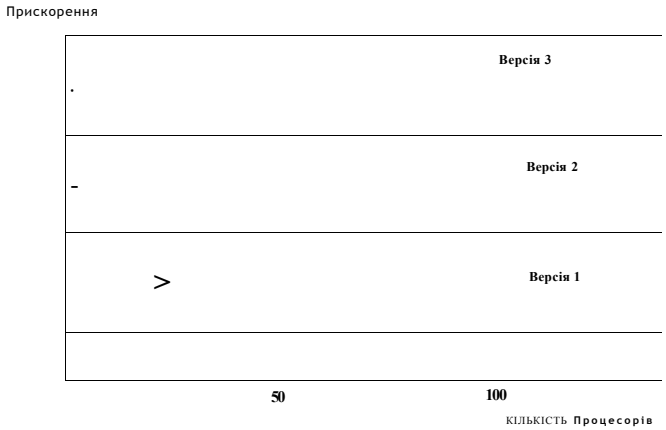


Рис. 12.3. Прискорення програми моделювання процесів молекулярної динаміки для системи Intel Paragon

Перша версія програми погано піддалася розпаралелюванню. З невеликими витратами, шляхом детальнішого налагоджування коду фрагментів програми, що опрацьовувалися окремими процесорами, отримано другу версію. Та лише радикальне покращення програмної частини реалізації інформаційних обмінів окремими процесорами надало остаточний варіант за версією 3, із наближенням до лінійного прискорення.

Зрозумілим є намагання програмістів забезпечити лінійне прискорення (теоретична межа), коли із збільшенням у п разів кількості задіяних процесорів також у п разів зменшується час виконання програми. Однак це не завжди вдається, і причину цього пояснює закон Амдала.

Закон Амдала і його наслідки. Припустимо, що у деякій програмі частка операцій, які потрібно виконувати послідовно, дорівнює {, де 0<i"<1 (при цьому частка розуміється не по статичному числу рядків коду, а по числу реально виконуваних операцій). Крайні випадки в значеннях *і* відповідають цілком паралельним *((=0)* і цілком послідовним (r=1) програмам. Отож, для того, щоб оцінити, яке прискорення Б може бути отримане на комп'ютері з р процесорами при даному значенні {, можна скористатися законом Амдала, згідно з яким Б = р/(1+(р-1)i"). При великих значеннях р прискорення Б прямує до величини 1/£ Тобто, відповідно до цього закону, якщо 9/10 програми здійснюється паралельно, а 1/10, як і раніше, послідовно, то прискорення більш ніж у 10 разів одержати в принципі неможливо поза залежністю від якості реалізації паралельної частини коду і кількості використовуваних процесорів (ясно, що прискорення в 10 разів досягається лише в тому випадку, коли час виконання паралельної частини дорівнює 0).

Подивимося на проблему з іншої сторони: а яку ж частину коду треба прискорити (а значить і попередньо досліджувати), щоб одержати задане прискорення? Відповідь можна знайти в наслідку із закону Амдала: для того, щоб прискорити виконання програми в *q* разів необхідно прискорити не менше, ніж у q разів (1-1/**c7**)-у частину програми. Отже, якщо є бажання прискорити програму в 100 разів у порівнянні з її послідовним варіантом, то необхідно в 100 разів прискорити 99% коду, що майже завжди складає значну частину програми. Таким чином, якщо після оцінки закладеного в програмі алгоритму стало зрозуміло, що частка послідовних операцій велика, то на значне прискорення

розраховувати безумовно не доводиться і потрібно думати про заміну окремих компонент алгоритму.

У зв'язку з наведеними проблемами число процесорів в реальних багатопроцесорних системах, які орієнтовані в основному на вирішення складних одиничних задач, є невеликим. Інформацію про кількість процесорів таких систем подає рис. 12.4.

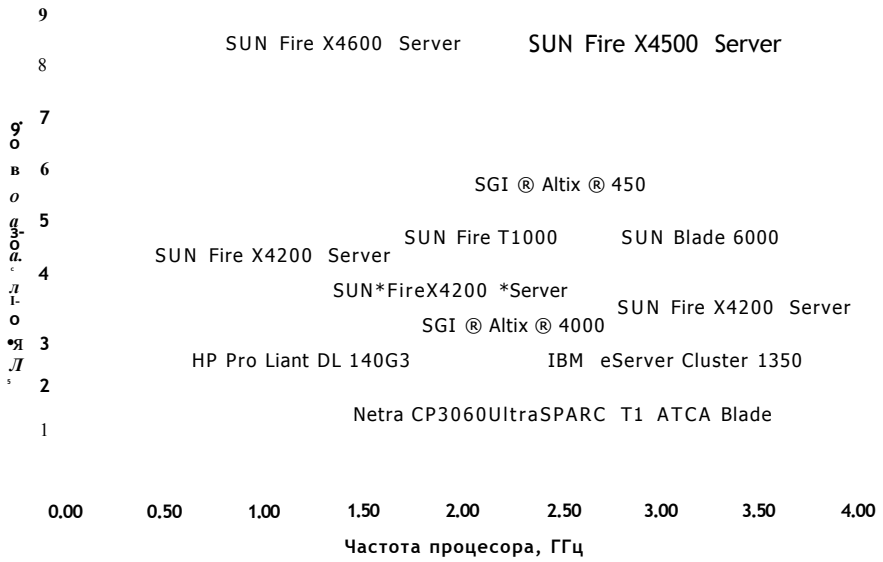


Рис. 12.4. Дані про реальні багатопроцесорні системи

Потрібно відзначити, що при створенні систем із масовою паралельною обробкою інформації, про які згадувалось вище, розробники орієнтуються не стільки на скорочення часу виконання алгоритмів, скільки на те, що відповідно до росту потужності системи будуть збільшуватись розміри вирішуваних задач.

12.3. Багатопотокова обробка інформації

Звичайно, перераховані вище підходи до підвищення продуктивності комп'ютерних систем, використовуючи паралельну обробку інформації, не виключають необхідність підвищення продуктивності однопроцесорних систем шляхом нарощування ресурсів процесора, наприклад, шляхом нарощування ємності кеш пам'яті та добавлення нових операційних пристроїв. Хоча зрозуміло, що це тягне за собою зростання кількості транзисторів та площі кристалу, ускладнення процесора, і відповідно, вартості.

Іншим підходом є підвищення ефективності використання наявних ресурсів комп'ютерних систем з одним та декількома процесорами. Існує декілька дешевших, ніж масове розпаралелювання варіантів, які також дозволяють прискорити обчислення, а саме:

- Багатопроцесорність на одному кристалі (Chip Multiprocessing). Два процесорні ядра фізично розташовані на одному кристалі із використанням спільної або розподіленої кеш пам'яті. Природно, що розмір кристала отримують досить великим, що є причиною великої вартості цього кристала. В багатопроцесорній системі можуть функціонувати декілька таких кристалів.

• Багатопотокова обробка з квантуванням часу (Time-Slice Multithreading). Процесор перемикають поміж програмними потоками через фіксовані проміжки часу. Накладні витрати часом виходять досить значними, особливо якщо який-небудь процес знаходиться в стані очікування

• Багатопотокова обробка з перемиканням за подіями (Switch-on-Event Multithreading). Перемикання задач при наявності тривалих пауз, наприклад "невлучень" до кеш пам'яті, велике число яких притаманне серверним аплікаціям. Тут процес, що чекає на завантаження даних з порівняно повільної основної пам'яті до кеш пам'яті, пригальмовують, вивільняючи тим самим ресурси процесора на користь інших процесів. Проте багатопотокова обробка з перемиканням за подіями, як і багатопотокова обробка з квантуванням часу, не завжди дозволяє досягти оптимального використання ресурсів процесора, зокрема через помилки в передбаченні розгалужень, існуючі залежності поміж командами тощо

• Одночасна багатопотокова обробка (Simultaneous Multithreading). Тут програмні потоки виконуються на одному процесорі "одночасно", тобто без переключення між ними. Ресурси процесора розподіляються динамічно, за принципом "не використовуєш - віддай іншому"

Одночасна багатопотокова обробка покладена в основу технології гіперпотокової обробки (Hyper-Threading) фірми Intel, яку розглянемо детальніше

Багатопотокові обчислення використовуються не лише в серверах, де багатопотоковість існує первинно, але і в робочих станціях і настільних персональних комп'ютерах. Потоки можуть відноситися як до однієї, так і до різних прикладних програм, але майже завжди активних потоків більше від одного (аби переконатися в цьому, достатньо у Windows 20007XP відчинити Task Manager та увімкнути відбиття ним числа виконуваних потоків). Разом з тим відомо, що стандартний процесор може одночасно опрацьовувати лише один з декількох існуючих потоків, тому і змушений постійно перемикатися поміж цими потоками

Технологію гіперпотокової обробки реалізовано в процесорі Intel Xeon MP (Foster MP), на якому і відбувалося дослідження ефективності цієї технології. Процесор Xeon MP використовує притаманне Pentium 4 ядро Willamette, містить 256 KB кеш пам'яті другого рівня L2, 512 KB кеш пам'яті третього рівня L3, та підтримує функціонування в чотирипроцесорних конфігураціях. Підтримка технології гіперпотокової обробки також присутня у процесорі для робочих станцій Intel Xeon (ядро Prestonia, 512 KB кеш пам'яті другого рівня L2), що поступив на ринок дещо раніше від процесора Xeon MP. Далі розглянемо можливості технології гіперпотокової обробки на прикладі саме процесора Intel Xeon.

На рис. 12.5 a показано завантаженість процесорів комп'ютерної системи, яка складається з двох суперскалярних процесорів. Кожен процесор може виконувати по 3 команди в одному циклі. Незадіяні (позначено білим) прямокутники свідчать про неоптимальну утилізацію системних ресурсів В той же ж час технологія гіперпотокової обробки не лише дозволяє опрацьовувати паралельно декілька потоків, але і зменшує при цьому наявне число незадіяних виконавчих часових інтервалів. На рис. 12.5 b показано завантаженість процесорів комп'ютерної системи, яка також складається з двох фізичних процесорів, та в якій використовується технологія гіперпотокової обробки. Кожен з процесорів тут опрацьовує по два потоки інформації, що дозволило наблизити систему до пікової продуктивності.

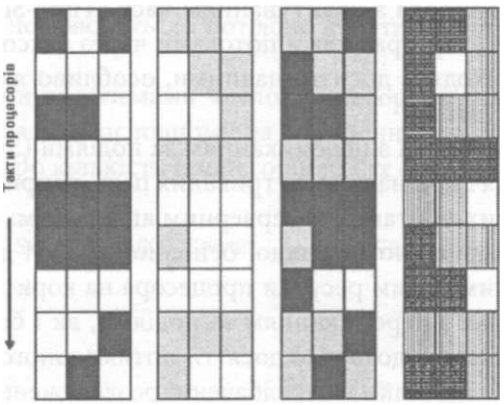


Рис. 12.5. Завантаженість процесорів комп'ютерної системи, яка складається з двох суперскалярних процесорів без використання технологіїгіперпотоковоїобробки (а) та з використанням технології гіперпотокової обробки (Ь)

Технологія гіперпотокової обробки ґрунтується на тому, що одночасно тільки частина ресурсів процесора використовується на опрацювання програмного коду. Не використовувані ресурси можна завантажити іншою роботою, наприклад, задіяти на паралельне виконання ще одного додатку (або іншого потоку з того ж додатку). Заради цього в одному фізичному процесорі Intel Xeon формують два логічні процесори (LP - Logical Processor), що поділяють поміж собою обчислювальні ресурси єдиного фізичного процесора. Операційна система та прикладна програма "відчувають" саме два логічні процесори та спроможні розподіляти роботу між ними так само, як і у випадку повноцінної двопроцесорної системи. Поділ ресурсів (зокрема, виконавчих вузлів) поміж двома потоками подано на рис. 12.5b.

Потрібно відзначити, що технологією гіперпотокової обробки передбачено, аби при наявності лише одного активного потоку дозволити йому виконуватися з швидкодією, як і на звичайному процесорі, тобто щоб ефективність використання процесора не зменшувалася. Заради цього у процесорі передбачено два режими роботи: однозадачний (ОЗ) та багатозадачний (БЗ). У режимі ОЗ активним є лише один логічний процесор, який без обмежень користується наявними ресурсами. Інший логічний процесор призупинено командою HALT. Із появою другого програмного потоку логічний процесор, що не був задіяний, активується (за допомогою переривання поточного стану HALT). При цьому фізичний процесор перемикається до стану БЗ. Пригальмовування незадіяних логічних процесорів командою HALT покладено на операційну систему, яка, до речі, і відповідає за таке ж швидке виконання одного потоку, як і у випадку без використання технології Hyper-Threading.

Для кожного з двох логічних процесорів зберігають так званий архітектурний стан (Architecture State), який складено із станів регістрів різного типу - загального призначення, керуючих, регістрів контролера переривань і службових (рис. 12.6). У кожного логічного процесора є свій контролер переривань і множина регістрів, для коректної роботи з якими введена таблиця альтернативних назв регістрів, яка відслідковує від-

повідність поміж регістрами загального призначення логічних процесорів та фізичного процесора (використовують одну таку таблицю на кожен логічний процесор).

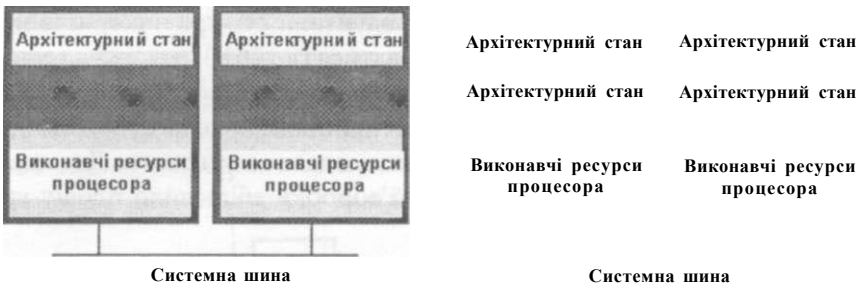


Рис. 12.6. Традиційна двопроцесорна система (а) та альтернативна система з підтримкою технології гіперпотокової обробки (Ь)

При паралельному опрацюванні двох потоків підтримується вміст двох лічильників команд. Переважна частка команд отримується з таблиці Trace Cache (TC), де ці команди зберігають в декодованому (трансльованому до рівня мікрооперацій) виді. Доступ до TC обидва активні логічні процесори отримують за чергою, через такт. У той же час, коли активним є лише один логічний процесор, він отримує монопольний доступ до TC, без чергування за тактами. Так само відбувається і доступ до пам'яті команд, коли треба опрацювати складну команду, що не має динамічно компільованого варіанту для подання на безпосереднє виконання. Буфери перетворення з передісторією ITLB (Instruction Translation Look-aside Buffer), задіяні за умови відсутності необхідних команд у кеш пам'яті команд, дублюються і постачають команди за правилом "кожен для свого потоку". Блок декодування команд є поділюваним і у випадку, коли потрібне декодування команд для обох потоків, обслуговує їх за чергою (знову-таки через такт). Блоки черги мікрооперацій та їх розміщення поділено навпіл, аби розподілити елементи для кожного логічного процесора. Блоки диспетчерів у кількості 5 штук опрацьовують черги декодованих команд, незважаючи на приналежність до LP0 або LP1, і спрямовують команди на виконання цільовим виконавчим блокам залежно від готовності до виконання операндів команд і наявності вільного стану цільових виконавчих блоків (динамічне виконання). Кеш пам'яті усіх рівнів (L1/L2 для Xeon, а також L3 для Xeon MP) є цілком поділюваними поміж двома логічними процесорами, проте для забезпечення когерентності (цілісності) даних записи до буферів перетворення з передісторією DTLB (Data Translation Look-aside Buffer) забезпечуються дескрипторами у вигляді ідентифікаторів логічних процесорів.

Таким чином, команди обох логічних процесорів можна виконувати одночасно на ресурсах одного фізичного процесора, причому ці ресурси поділено на чотири наступні класи: дубльовані, цілком поділювані, із дескрипторами елементів, динамічно поділювані залежно від режиму роботи: однозадачний режим першого або другого логічного процесора чи багатозадачний режим.

Більшість прикладних програм, що отримують прискорення у багатопроцесорних системах, можуть також прискорюватися і на процесорі із увімкнутим режимом гіпер-

потокової обробки без будь-яких змін власних машинних кодів. Але існують і проблеми. Наприклад, якщо один процес знаходиться в стані очікування, то він спроможний захопити усі ресурси фізичного процесора, перешкоджаючи тим самим функціонуванню другого логічного процесора. Таким чином, продуктивність при використанні технології гіперпотокової обробки може іноді й знижуватися (до 20%).

Ще однією метою реалізації технології гіперпотокової обробки було зведення до мінімуму зростання числа транзисторів, площі кристала та енергоспоживання при помітному зростанні продуктивності. І це завдання вдалося виконати. Додання до Хеоп/ Xeon-MP підтримки технології гіперпотокової обробки збільшило площу кристала та енергоспоживання лише на 5%.

Загалом ефективність багатопотокової технології підтверджується зростанням продуктивності. Наступним рисунком (рис. 12.7) подано порівняльну продуктивність, що забезпечує двопроцесорна система без використання багатопотокової технології та із використанням цієї технології

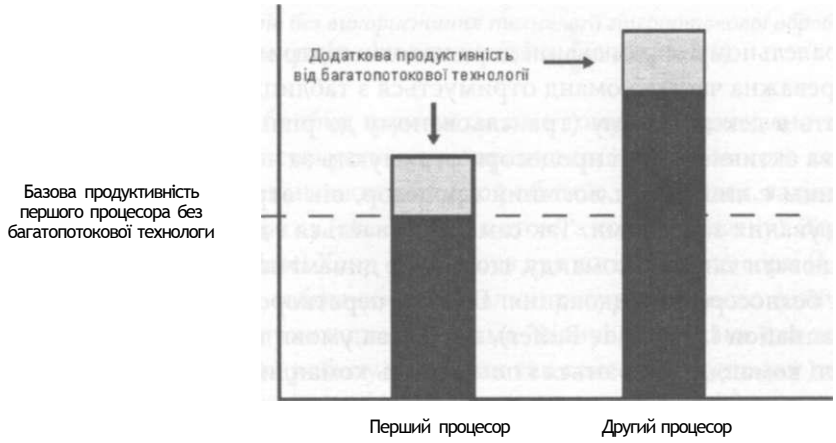


Рис.12.7. Продуктивність двопроцесорної системи залежно від кількості процесорів та залежно від використання (невикористання) технології гіперпотокової обробки (процесор Intel' Xeon")

Бачимо зростання продуктивності (темний колір) при використанні другого процесора в порівнянні із базовим її рівнем для одного задіяного процесора, та додатковий приріст продуктивності за рахунок увімкнення багатопотокової обробки (світлий колір).

12.4. Класифікація паралельних комп'ютерних систем

12.4.1. Класифікація Шора

Одна з перших класифікацій комп'ютерних систем була запропонована Д. Шором на початку 70-х років. Вона цікава тим, що є спробою виділення типових способів компонування комп'ютерних систем на основі фіксованого числа базових блоків: пристрою керування, арифметико-логічного пристрою, пам'яті команд і пам'яті даних. Додатково передбачається, що вибірка з пам'яті даних може здійснюватися словами, тобто вибираються всі розряди одного слова, і/або бітовим шаром - по одному розряду з однієї і тієї ж позиції кожного слова (іноді ці два способи називають горизонтальною і вертикальною

вибірками відповідно). Звичайно ж, при аналізі даної класифікації треба робити знижку на час її появи, оскільки передбачити велику різноманітність паралельних систем теперішнього часу тоді було у принципі неможливо. Отже, згідно з класифікацією Д. Шора, всі комп'ютери розбиваються на шість класів, перший з яких дістав назву машини I, другий - машини II, і т. д.

На поданих нижче рисунках позначено: ПІ - пам'ять команд (інструкцій), ПК - пристрій керування, АЛП - арифметико-логічний пристрій, ПД - пам'ять даних.

Машина I - це комп'ютерна система, яка містить пристрій керування, арифметико-логічний пристрій, пам'ять команд і пам'ять даних із послівною вибіркою (рис. 12.8).

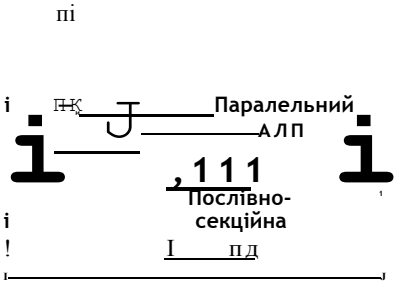


Рис. 12.8. Машина I

Зчитування даних здійснюється вибіркою всіх розрядів деякого слова для їх паралельної обробки в арифметико-логічному пристрої. Склад АЛП спеціально не вказується, що допускає наявність декількох функціональних пристроїв, в тому числі конвеєрного типу. За цими міркуваннями до даного класу потрапляють як класичні послідовні машини (IBM 701, PDP-11, VAX 11/780), так і конвеєрні скалярні (CDC 7600) і векторно-конвеєрні (CRAY-1).

Якщо в машині I здійснювати вибірку не по словах, а по одному розряду з усіх слів, то одержимо машину II (рис. 12.9). Слова в пам'яті даних, як і раніше, розташовуються горизонтально, але доступ до них здійснюється інакше. Якщо в машині I відбувається послідовна обробка слів при паралельній обробці розрядів, то в машині II - послідовна обробка бітових шарів при паралельній обробці множини слів.

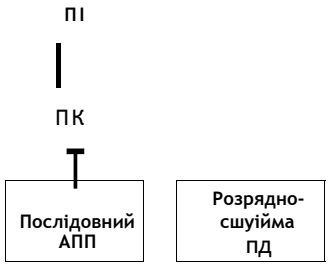


Рис. 12.9. Машина II

Структура машини II лежить в основі асоціативних комп'ютерів (наприклад, так побудовано центральний процесор машини БТАРчАМ), причому фактично такі комп'ютери мають в складі арифметико-логічного пристрою множину порівняно простих операцій-

них пристроїв порозрядної обробки даних. Іншим прикладом служить матрична система ICL DAP, яка може одночасно обробляти по одному розряду з 4096 слів.

Якщо об'єднати принципи побудови машин I і II, то одержимо машину III (рис. 12.10). Ця машина має два арифметико-логічні пристрої - горизонтальний і вертикальний, а також модифіковану пам'ять даних, яка забезпечує доступ як до слів, так і до бітових шарів. Вперше ідею побудови таких систем у 1960 році висунув У Шуман, що називав їх ортогональними (якщо пам'ять представляти як матрицю слів, то доступ до даних здійснюється в напрямі, "ортогональному" традиційному - не по словах (рядках), а по бітових шарах (стовпцях)). У принципі, як машину STARAN, так і ICL DAP можна запрограмувати на виконання функцій машини III, але оскільки вони не мають окремих АЛП для обробки слів і бітових шарів, віднести їх до даного класу не можна. Повноправними представниками машин класу III є обчислювальні системи сімейства OMEN-60 фірми Sanders Associates, побудовані в прямій відповідності до концепції ортогональної машини.

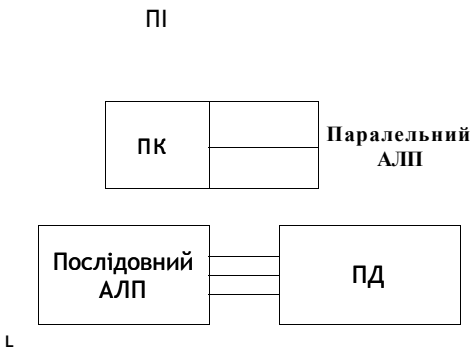


Рис. 12.10. Машина III

Якщо в машині I збільшити кількість пар арифметико-логічний пристрій ⟺ пам'ять даних (іноді цю пару називають процесорним елементом), то одержимо машину IV (рис. 12.11). Єдиний пристрій керування видає команду за командою відразу всім процесорним елементам. З одного боку, відсутність з'єднань між процесорними елементами робить подальше нарощування їх числа відносно простим, але з іншого боку, сильно обмежує застосовність машин цього класу. Таку структуру має обчислювальна система PEPE, яка об'єднує 288 процесорних елементів

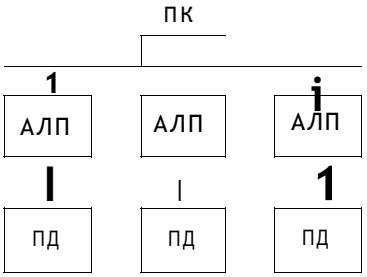


Рис. 12.11. Машина IV

Якщо ввести безпосередні лінійні зв'язки між сусідніми процесорними елементами машини IV, то одержимо структуру машини V (рис. 12.12). Будь-який процесорний елемент тепер може звертатися до даних як у своїй пам'яті, так і в пам'яті безпосередніх сусідів. Подібна структура характерна, наприклад, для класичної матричної багатопроцесорної системи ІЬІЛАС IV.

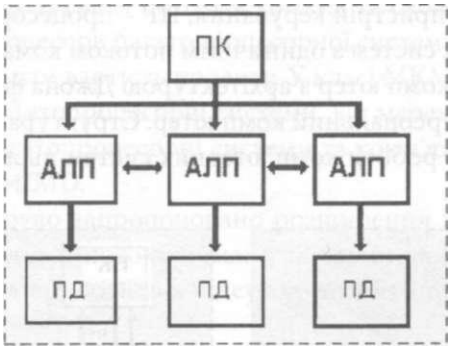


Рис. 12.12. Машина V

Відмітимо, що у всіх машинах з 1-ї по *У-у* дотримана концепція розділення пам'яті даних і арифметико-логічних пристроїв, припускаючи наявність шини даних або якого-небудь комутуючого елемента між ними. Машина VI, названа матрицею з функціональною пам'яттю (або пам'яттю з вбудованою логікою), є іншим підходом, що передбачає розподіл логіки процесора по всьому запам'ятовуючому пристрою (рис. 12.13). Прикладом можуть служити як прості асоціативні запам'ятовуючі пристрої, так і складні асоціативні процесори.

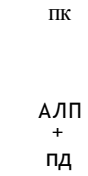


Рис. 12.13. Машина VI

12.4.2. Класифікація Фліна

Одну з перших практично значимих класифікацій паралельних комп'ютерних систем подав у 1966 році співробітник фірми ІВМ Майкл Флін, який зараз є професором Стенфордського університету (СІЛА). Його класифікація базується на оцінці потоку інформації, яка поділена на потік даних між основною пам'яттю та процесором, та потік команд, які виконує процесор. При цьому потік даних та команд може бути як одиничним, так і множинним. Згідно з М. Фліном, усі комп'ютерні системи поділяють наступним чином:

- ОКОД - комп'ютерні системи з одиничним потоком команд та одиничним потоком даних (SISD - Single Instruction stream Single Data stream).
- МКОД - комп'ютерні системи з множинним потоком команд та одиничним потоком даних (MISD - Multiply Instruction stream Single Data stream).

• ОКМД - комп'ютерні системи з одиничним потоком команд та множинним потоком даних (SIMD - Single Instruction stream Multiply Data stream).

• МКМД - комп'ютерні системи з множинним потоком команд та множинним потоком даних (MIMD - Multiply Instruction stream Multiply Data stream).

Розглянемо запропоновану М. Фліном класифікацію детальніше. На поданих нижче рисунках позначено: ПК - пристрій керування, ПР - процесор, ПД - пам'ять даних.

До класу комп'ютерних систем з одиничним потоком команд та одиничним потоком даних належить, зокрема, комп'ютер з архітектурою Джона фон Неймана, яким, наприклад, є розповсюджений персональний комп'ютер. Структура цієї системи представлена на рис. 12.14a. Організація роботи комп'ютерних систем цього класу була розглянута в попередніх розділах книги.

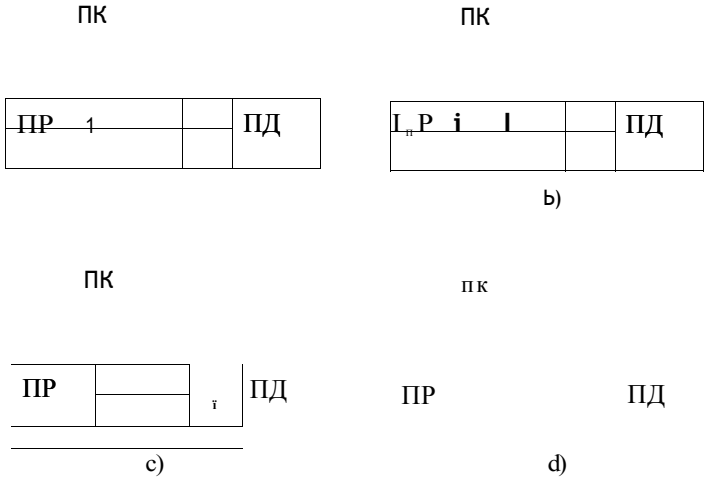


Рис. 12.14. Структура комп'ютерної системи типу ОКОД a), МКОДЬ), ОКМД c), МКМД d)

Структура комп'ютерної системи з множинним потоком команд та одиничним потоком даних показана на рис. 12.14b. Комерційні універсальні комп'ютерні системи цього типу на даний час невідомі, проте вони можуть з'явитися у майбутньому. До цього типу систем з деякими умовностями можна віднести спеціалізовані потокові процесори, зокрема систолічні, які використовують, наприклад, при обробці зображень.

В комп'ютерній системі з одиничним потоком команд та множинним потоком даних (рис. 12.14 c) одночасно обробляється велика кількість даних. До цього класу, зокрема, належать раніше розглянуті векторні процесори. До комп'ютерних систем з одиничним потоком команд та множинним потоком даних можна віднести також апаратну підсистему процесорів Pentium, яка реалізовує технологію MMX опрацювання даних для графічної операційної системи Windows.

Характерним прикладом комп'ютерної системи з одиничним потоком команд та множинним потоком даних може служити система, яка складається з двох частин: зовнішнього комп'ютера з архітектурою Джона фон Неймана, який виконує роль пристрою керування, і масиву ідентичних синхронізованих елементарних процесорів, здатних одночасно виконувати ту ж саму дію над різними даними. Кожен процесор у масиві має місцеву пам'ять невеликої ємності, де зберігаються дані, які обробляються паралельно.

З масивом процесорів з'єднано шину пам'яті зовнішнього комп'ютера таким чином, що він може довільно звернутися до кожного процесора масиву. Програма може виконуватися традиційно послідовно на зовнішньому комп'ютері, а її частина може паралельно виконуватися на масиві процесорів.

У комп'ютерній системі з множинним потоком команд та множинним потоком даних кожен процесор оперує із своїм потоком команд та своїм потоком даних (рис. 12. - 4c1). Як правило, окремі процесори багатопроцесорної системи є серійними пристроями, що дозволяє значно зменшити вартість проекту. У класі МКМД треба відрізняти сильно зв'язані системи, власне багатопроцесорні системи, від мереж комп'ютерів, тобто слабо зв'язаних систем; тобто багатопроцесорні системи та комп'ютерні мережі потрапляють до різних підкласів класу МІУГО.

В 1978 році Д. Куком було запропоновано розширення класифікації Фліна. У своїй класифікації Д. Кук розділив потоки команд та даних на скалярні та векторні потоки. Комбінація цих потоків приводить в підсумку до 16 типів архітектури паралельних комп'ютерних систем.

12.5. Типи архітектур систем ОКМД

До комп'ютерних систем класу ОКМД належать різні типи систем, які під керуванням одиничного потоку команд обробляють потоки даних. В першу чергу до цих систем потрібно віднести векторні та матричні комп'ютерні системи. Перші з них були детально розглянуті раніше. Призначення матричних комп'ютерних систем багато в чому схоже з призначенням векторних комп'ютерних систем - обробка великих масивів даних. В основі матричних комп'ютерних систем лежить матричний процесор, що складається з регулярного масиву процесорних елементів (ПЕ). Організація систем подібного типу на перший погляд достатньо проста. Вони мають пристрій керування, що генерує потік команд і велике число ПЕ, що працюють паралельно і обробляють кожен свій потік даних. Проте на практиці, щоб забезпечити достатню ефективність системи при вирішенні широкого кола завдань, необхідно організувати зв'язки між процесорними елементами так, щоб якнайповніше завантажити їх роботою. Саме характер зв'язків між ПЕ і визначає різницю у властивостях системи.

Між матричними і векторними комп'ютерними системами є істотна різниця.

В складі системи команд векторного процесора є команди обробки векторів даних, що дозволяє ефективно завантажити конвеєри його операційних блоків. Векторні процесори простіше використовувати, тому що команди для обробки векторів - це зручніша модель програмування, ніж команди для паралельно включених ПЕ.

Матричний процесор інтегрує безліч ідентичних процесорних елементів, логічно об'єднаних у матрицю і працюючих в стилі ОКМД. Не так істотно, як конструктивно реалізована матриця процесорних елементів - на одному кристалі чи на декількох. Важливий сам принцип - ПЕ логічно скомпоновані в матрицю і працюють синхронно, тобто присутній тільки один потік команд для всіх.

Є дві головні конфігурації, які були використані в матричних комп'ютерних системах класу ОКМД. В першій схемі кожен процесор Р має власну локальну пам'ять М (рис. 12.15).

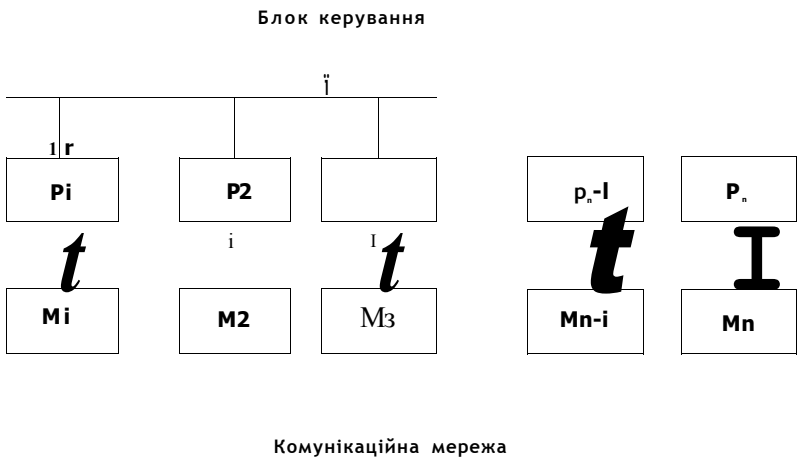


Рис. 12.15. Багатопроцесорна комп'ютерна система з одиничним потоком команд та множинним потоком даних із власною пам'яттю в кожного процесора

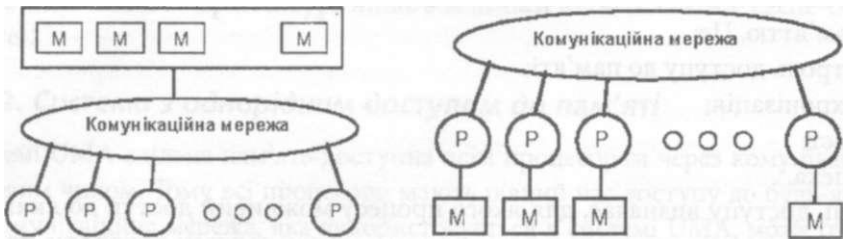
Процесори можуть зв'язатися один з одним через комунікаційну мережу. Якщо комунікаційна мережа не забезпечує прямого зв'язку між заданою парою процесорів, то згодом ця пара може обмінятися даними через проміжний процесор. Таку схему зв'язку було використано в комп'ютері ILLIAC IV. Комунікаційна мережа в ILLIAC IV дозволяла кожному процесору зв'язуватися безпосередньо з чотирма сусідніми процесорами. В матриці із 8x8 процесорів кожний і-й процесор міг контактувати безпосередньо з (і-1)-м, (і+1)-м, (і-8)-м, і (і+8)-м процесорами.

У другій схемі процесори і модулі пам'яті зв'язуються між собою через комунікаційну мережу (рис. 12.16). Два процесори можуть передати дані один одному через проміжний модуль пам'яті або, можливо, через проміжний процесор. За такою схемою, наприклад, побудовано процесор BSP фірми Burroughs.

Рис. 12.16. Багатопроцесорна комп'ютерна система з одиничним потоком команд та множинним потоком даних із спільною пам'яттю

12.6. Типи архітектур систем МКМД

Комп'ютерні системи класу МКМД складаються з багатьох процесорів та багатьох модулів пам'яті, з'єднаних за допомогою комунікаційної мережі. Вони можуть бути поділені на дві великі групи: зі спільною пам'яттю таз передачею повідомлень. На рис. 12.17 а та Ь показано загальну структуру цих двох груп комп'ютерних систем.



а Ь

Рис. 12.17. Загальна структура комп'ютерних систем зі спільною пам'яттю а та з передачею повідомлень Ь

В першій групі процесори обмінюються інформацією через їх спільну пам'ять, причому кожний процесор має рівні можливості читати та записувати дані до пам'яті, а також однакову швидкість доступу до пам'яті, тому їх часто називають симетричними багатопроцесорними системами. Комерційними прикладами комп'ютерних систем першої групи є багатопроцесорні сервери фірм Sequent Computer's Balance and Symmetry, Sun Microsystems та Silicon Graphics Inc.

У другій групі процесори обмінюються інформацією через комунікаційну мережу. В комп'ютерній системі з передачею повідомлень (також їх називають системами з розподіленою пам'яттю) зазвичай наявна локальна пам'ять і процесор у кожному вузлі комунікаційної мережі. Тут відсутня спільна пам'ять, тому необхідно переміщувати дані з однієї локальної пам'яті до іншої за допохмогою механізму передачі повідомлень. Це, зазвичай, робиться шляхом посилання-отримання кількох команд, які повинні бути вписані в прикладне програмне забезпечення. Комерційними прикладами систем передачі повідомлень є системи nCUBE, iPSC/2 і різні системи, базовані на трансп'ютерах. Ці системи кінець кінцем поступилися системам глобальної мережі Internet, в якій вузли є або серверами, або персональними комп'ютерами.

Архітектуру з розподіленою пам'яттю довелося використовувати із-за переходу до все більших систем. Потрібно відзначити, що програмування в системі зі спільною пам'яттю є простішим, а в системах передачі повідомлень забезпечується масштабованість. Тому з'явились комбіновані системи з розподіленою та з спільною пам'яттю, такі як SGI Origin2000, та інші.

12.7. Організація комп'ютерних систем із спільною пам'яттю

12.7.1. Типи комп'ютерних систем із спільною пам'яттю

При проектуванні багатопроцесорних систем ключовим є питання поділу даних між процесорами. Ідеальною відповіддю на це питання є застосування єдиного адресного

простору. Тут процесори взаємодіють через спільні змінні, що зберігаються в єдиній пам'яті з єдиним адресним простором для усіх процесорів. У системі із спільною пам'яттю процесори взаємодіють між собою шляхом зчитування-запису інформації з комірок спільної пам'яті, яка є однаково доступною для всіх процесорів. Кожен процесор може мати регістри, буфери, кеш та локальну пам'ять, як додаткові ресурси пам'яті. Потрібно брати до уваги кілька основних проблем, які з'являються при проектуванні систем із спільною пам'яттю. Це:

- контроль доступу до пам'яті;
- синхронізація;
- захист;
- безпека.

Контроль доступу визначає, для якого процесу можливий доступ до яких ресурсів. Моделі контролю доступу вимагають перевірки вмісту таблиці контролю доступу для кожного запиту доступу від процесорів до спільної пам'яті. Ця таблиця вміщує прапорці, які визначають законність кожної спроби доступу. Якщо є спроба доступу до ресурсів, то поки бажаний доступ не завершений, всі спроби доступу нехтуються, і нелегальні процеси блокуються. Запити від сумісних процесів можуть змінити вміст таблиці контролю доступу протягом їх виконання. Прапорці контролю доступу з правилами синхронізації визначають функціональність системи.

Механізми синхронізації обмежують час доступу від сумісних процесів до спільних ресурсів. Досконала синхронізація гарантує, що інформація передається належним чином та забезпечується належна системна функціональність.

Захист - це системна особливість, що перешкоджає наданню процесам довільного доступу до ресурсів, що належать іншим процесам. Сумісне використання і захист є протилежними за функціями, оскільки сумісне використання дозволяє доступ, тоді як захист обмежує його.

Найпростіша система із спільною пам'яттю має один модуль пам'яті, який може бути доступний від двох процесорів (рис. 12.18). Запити надходять в модуль пам'яті через його два порти. Арбітражний блок у межах модуля пам'яті передає запити до диспетчера пам'яті. Якщо модуль пам'яті не зайнятий і надходить один запит, то арбітражний блок передає цей запит до диспетчера пам'яті і запит обслуговується. Модуль знаходиться в зайнятому стані під час обслуговування запиту. Якщо надходить новий запит, коли пам'ять зайнята, обслуговуючи попередній запит, процесор Р, що послав запит, може утримувати свій запит на лінії, поки пам'ять не стає вільною, або може повторювати свій запит пізніше.

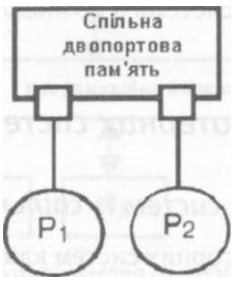


Рис. 12.18. Система із спільною пам'яттю з двома портами

Залежно від типу комунікаційної мережі, системи із спільною пам'яттю можна класифікувати наступним чином:

- системи UMA з однорідним доступом до пам'яті (UMA - uniform memory access);
- системи NUMA з неоднорідним доступом до пам'яті (NUMA - nonuniform memory access);
- системи COMA - архітектура лише з кеш-пам'яттю (COMA - cache-only memory architecture).

12.7.2. Системи з однорідним доступом до пам'яті

У системі UMA спільна пам'ять доступна всім процесорам через комунікаційну мережу однаковим чином. Тому всі процесори мають рівний час доступу до будь-якої комірки пам'яті. Комунікаційна мережа, яка використовується в системі UMA, може бути одиничною або множинною шиною, координатною мережею чи багатопортовою пам'яттю.

Типова структура системи з однорідним доступом до пам'яті на основі одношинної комунікаційної мережі приведена на рис. 12.19 а.

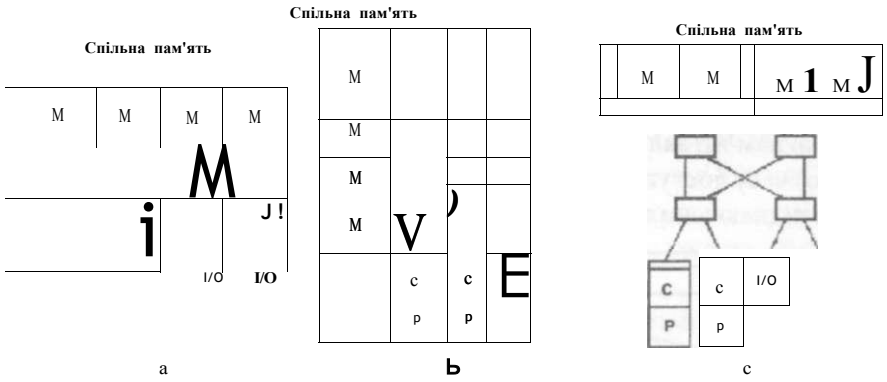


Рис. 12.19. Багатопроцесорні системи ЦМА із спільною пам'яттю на основі шинної топології а, координатного комутатора Ь та багатоярусної мережі с

В граничному випадку час пересилання через шину може бути зменшений до нуля після того, як вміст кеш пам'ятей С завантажено від спільної пам'яті. Ця організація пам'яті є найпопулярнішою серед систем із спільною пам'яттю. Вона дозволяє досить просто розширити систему шляхом підключення більшої кількості процесорів. Приклади цієї архітектури - сервери Sun Starfire servers, HP V series, і Compaq AlphaServer GS. Зрозуміло, що продуктивність цієї системи обмежена часом циклу шини. Тому кожен процесор має свою кеш пам'ять, що суттєво зменшує кількість звернень до шини. Наявність багатьох кеш пам'ятей породжує проблему їх когерентності, тобто несуперечності вмісту кеш пам'яті кожного процесора із вмістом спільної основної пам'яті багатопроцесорної системи. Зазначену проблему вирішують шляхом спостереження за шиною, що з'єднує процесори з пам'яттю, за допомогою контролера кожної кеш пам'яті разом із реалізацію в кожній кеш пам'яті наскрізного запису. Можна також в частині процесорів не використовувати кеш пам'ять.

Проблема когерентності пам'яті є однією з причин того, що багатопроцесорні системи із спільною пам'яттю на основі спільної шини мають невелику кількість процесорів.

Так, максимальна кількість процесорів типу Alpha 21264 в системі Compaq Alfa Server GS 140 рівна 14, процесорів PA-8500 в системі HP N9000 - 8, процесорів Power PC 604 у системі RS 6000 - 4.

Як вже вказувалось, використовувана в системі UMA комунікаційна мережа може бути одиничною чи множинною шиною, координатною мережею чи багатопортовою пам'яттю. На рис.12.19Ь та рис.12.19с подано типові логічні організації багатопроцесорних систем із спільною пам'яттю на основі топології координатної комутації та багатоярусної мережі. Тут буквою С позначено локальну кеш пам'ять, I/O - пристрої введення/виведення, Р - процесори, М - модулі спільної пам'яті.

Структури на основі топології координатної комутації (рис. 12.19а) та багатоярусної мережі (рис. 12.19b) дозволяють вирішити проблему обмеженої пропускної здатності системи із спільною шиною. Координатний комутатор та багатоярусна мережа забезпечують множинність шляхів з'єднання процесорів та блоків пам'яті. Тому тут кількість процесорів є зазвичай більшою, ніж в системах із спільною шиною.

12.7.3. Системи з неоднорідним доступом до пам'яті

У системі NUMA кожен процесор має частину спільної пам'яті (рис. 12.20). Ця пам'ять має єдиний адресний простір. Тому, будь-який процесор може звернутися до будь-якої комірки спільної пам'яті безпосередньо, використовуючи її адресу. Проте час доступу до модулів спільної пам'яті залежить від їх відстані від процесора. Це приводить до різного (неоднорідного) часу доступу до пам'яті від різних процесорів. Використовується кілька архітектур для підключення процесорів до модулів пам'яті в системі NUMA.

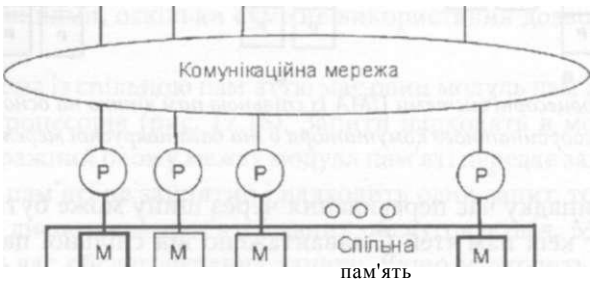


Рис. 12.20. Система NUMA із спільною пам'яттю

Як правило, система NUMA розрахована на застосування великої кількості процесорів. UMA є більш вибагливою щодо швидкодії підсистеми пам'яті. Тобто реалізація UMA є порівняно простішою та розповсюдженішою (багатопроцесорні сервери фірми Compaq, як приклад). Архітектуру NUMA реалізовано, наприклад, в суперкомп'ютері ORIGIN 2000 фірми SGI (більш точно - тут є архітектура CC-NUMA, або NUMA із множиною когерентних кеш пам'ятей).

У симетричних багатопроцесорних комп'ютерних системах із спільною пам'яттю має місце практична межа числа їх процесорів. Ефективна схема з кеш пам'яттю зменшує

навантаження на шину між процесором і основною пам'яттю, але зі збільшенням числа процесорів навантаження на шину також зростає. Оскільки шина використовується також для передачі сигналів, що забезпечують когерентність, ситуація з навантаженням на шину ще більш ускладнюється. З якогось моменту в плані продуктивності шина перетворюється на вузьке місце. Для систем із спільною пам'яттю такою межею стає число процесорів в межах від 16 до 64. Наприклад, кількість процесорів системи Silicon Graphics Power Challenge обмежена 64 процесорами типу R10000, оскільки при подальшому збільшенні числа процесорів продуктивність падає.

Обмеження на число процесорів в архітектурі з спільною пам'яттю служить спонукальним мотивом для розвитку кластерних систем. В останніх кожен вузол має локальну основну пам'ять, тобто додатки "не бачать" спільної основної пам'яті. По суті, когерентність підтримується не стільки апаратурою, скільки програмним забезпеченням, що не кращим чином позначається на продуктивності. Одним із шляхів створення великомасштабних комп'ютерних систем є технологія CC-NUMA. Наприклад, система NUMA Silicon Graphics Origin підтримує до 1024 процесорів R10000. Технологія CC-NUMA передбачає включення множини незалежних вузлів, кожний з яких може бути, наприклад, системою із спільною пам'яттю. Таким чином, вузол містить множину процесорів, у кожного з яких присутні локальні кеш пам'яті першого і другого рівнів. У вузлі є й основна пам'ять, спільна для всіх процесорів цього вузла, але така, що розглядається як частина спільної основної пам'яті системи. В архітектурі CC-NUMA вузол виступає основним будівельним блоком. Наприклад, кожен вузол у системі Silicon Graphics Origin містить два мікропроцесори MIPS R10000. Вузли об'єднуються за допомогою якої-небудь комунікаційної мережі, яка представлена координатним комутатором, кільцем або має іншу топологію.

Відповідно до технології CC-NUMA, кожен вузол у системі володіє власною основною пам'яттю, але з погляду процесорів має місце спільна пам'ять, де кожен елемент будь-якої локальної основної пам'яті має унікальну системну адресу. Коли процесор ініціює доступ до пам'яті і потрібна комірка відсутня в його локальній кеш пам'яті, кеш-пам'ять другого рівня процесора організує операцію вибірки. Якщо потрібна комірка знаходиться в локальній основній пам'яті, вибірка проводиться з використанням локальної шини. Якщо ж необхідна комірка зберігається у віддаленій секції спільної пам'яті, то автоматично формується запит, що посилається по комунікаційній мережі на потрібну локальну шину, і вже по ній до підключеної до даної локальної шини кеш пам'яті. Всі ці дії виконуються автоматично, прозорі для процесора і його кеш пам'яті.

У даній конфігурації головна турбота - когерентність кеш-пам'ятей. Хоча окремі реалізації і відрізняються в деталях, загальним є те, що кожен вузол має довідник, в якому зберігається інформація про місцезнаходження в системі кожної складової спільної пам'яті, а також про стан кеш пам'яті.

12.7.4. Системи лише з кеш пам'яттю

Подібно до NUMA в системі COMA кожен процесор має частину спільної пам'яті (рис. 12.21). Проте в даному випадку спільна пам'ять є кеш пам'яттю. Система COMA вимагає, щоб дані мігрували до процесора, що запросив їх.

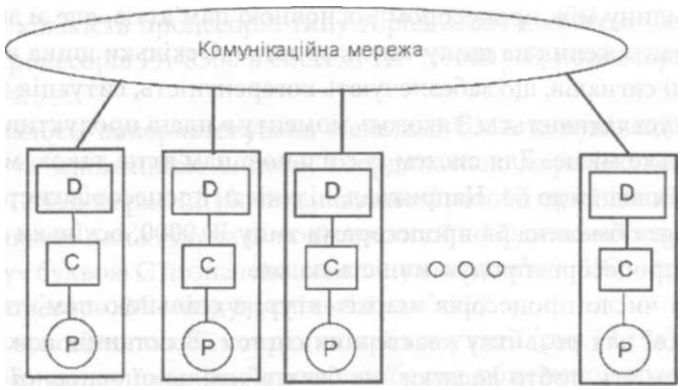


Рис. 12.21. Система COMA зі спільною пам'яттю

Тут відсутня ієрархія пам'яті, а адресний простір визначається ємністю кеш пам'яті. Крім того, тут наявна директорія D кеш пам'яті, яка допомагає у віддаленому доступі до кеш пам'яті. Прикладом цієї архітектури є система KSR-1 фірми Kendall Square Recearch.

12.8. Організація комп'ютерних систем із розподіленою пам'яттю

Системи передачі повідомлень - клас багатопроцесорних систем, в яких кожен процесор має доступ до своєї локальної пам'яті. На відміну від систем із спільною пам'яттю, комунікації в системі передачі повідомлень виконуються шляхом посилання та отримання повідомлень. Вузол в такій системі складається з процесора і його локальної пам'яті (рис. 12.22).

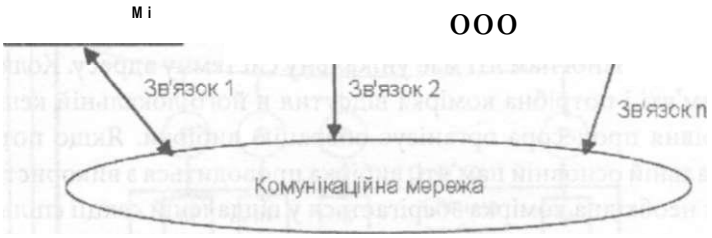


Рис. 12.22. Система з розподіленою пам'яттю

Вузли зазвичай запам'ятовують повідомлення в буферах (тимчасові комірки пам'яті, де повідомлення чекають до того часу, коли вони можуть бути послані або одержані), і виконують посилання-отримання повідомлення під час обробки даних. Паралельна обробка повідомлення і поточного обчислення здійснюється під керуванням операційної системи. Процесори не розділяють спільну пам'ять і кожен процесор має доступ до його власного адресного простору. Вузли в системі передачі повідомлень можуть бути з'єднані різноманітними способами - від спеціалізованих комутаторів до географічно розподілених мереж.

Підхід передачі повідомлень є масштабованим до великих пропорцій, тобто число процесорів може бути збільшено без істотного зменшення ефективності їх взаємодії.

Для організації зв'язку системи передачі повідомлень використовують статичні мережі, зокрема гіперкубічні мережі, які були досить популярні протягом багатьох років. Двовимірні і тривимірні решітчасті мережі так само широко використовувались в системах передачі повідомлень. Два важливі чинники повинні розглядатися в проектуванні комунікаційної мережі для систем передачі повідомлень. Це - пропускна здатність і мережний час очікування. Пропускна здатність визначена як кількість бітів інформації, переданих за одиницю часу (біт/сек). Мережний час очікування визначений як час, потрібний для передачі повідомлення. В 1987 році була введена блокова маршрутизація як альтернатива традиційній маршрутизації з проміжним зберіганням для того, щоб скоротити розмір необхідних буферів і щоб зменшити час очікування повідомлення. У блоковій маршрутизації пакет ділиться на менші блоки, які називають блоками керування потоком даних, так, що ці блоки рухаються конвеєрним методом разом з блоком заголовка до вузла призначення. Коли блок заголовка блокується через мережний затор, наступні за ним блоки блокуються також.

12.9. Комунікаційні мережі багатопроцесорних систем

12.9.1. Типи комунікаційних мереж

В основі архітектури будь-якої багатопроцесорної комп'ютерної системи лежить здатність до обміну даними між її компонентами. Це забезпечується комунікаційною мережею (КММ), яка з'єднує між собою вузли комп'ютерної системи за допомогою каналів передачі даних (каналів зв'язку). В ролі вузлів можуть виступати процесори, модулі пам'яті, пристрої введення-виведення, комутатори або декілька перерахованих елементів, об'єднаних у функціональний пристрій. Організація внутрішніх комунікацій комп'ютерної системи називається топологією.

Топологію комунікаційної мережі визначає множина вузлів, які об'єднані множиною каналів. Зв'язок між вузлами зазвичай реалізується по двоточковій схемі (point-to-point). Будь-які два вузли, зв'язані каналом зв'язку, називають суміжними вузлами або сусідами. Кожен канал з'єднує один вузол-джерело з одним вузлом-приймачем. Канал характеризується кількістю сигнальних ліній, частотою або швидкістю передачі бітів по кожній сигнальній лінії, затримкою - часом пересилання біта з одного вузла до іншого. Для більшості каналів затримка знаходиться в прямій залежності від фізичної довжини лінії зв'язку та швидкості розповсюдження сигналу.

Вузол у мережі може бути термінальним, тобто джерелом або приймачем даних, комутатором, що пересилає інформацію з вхідного порту на вихідний, або суміщати обидві ролі. У мережах із прямими зв'язками кожен вузол одночасно є як термінальним вузлом, так і комутатором, і повідомлення пересилаються між термінальними вузлами безпосередньо. У мережах з непрямими зв'язками вузол може бути або термінальним, або комутатором, але не одночасно, тому повідомлення передаються опосередковано, за допомогою виділених комутуючих вузлів. Існують також такі топології, які не можна однозначно зарахувати ні до прямих, ні до непрямих. Будь-яку пряму КММ можна зобразити у вигляді непрямої, розділивши кожен вузол на двотермінальний вузол і вузол комутації. Сучасні прямі мережі реалізуються саме таким чином - комутатор від-

діляється від термінального вузла і поміщається у виділений маршрутизатор. Основна перевага прямих КММ полягає в тому, що комутатор може використовувати ресурси термінальної частини свого вузла. Це стає істотним, якщо врахувати, що, як правило, останній включає комп'ютер або процесор.

Комунікаційні мережі багатопроцесорних систем можуть бути поділені на наступні групи:

- залежно від того, чи залишається конфігурація взаємозв'язків незмінною, принаймні поки виконується певне завдання, розрізняють мережі із статичною і динамічною топологіями;
- залежно від способу функціонування: синхронні та асинхронні;
- залежно від стратегії керування: централізовані та децентралізовані;
- залежно від способу перемикання: схемні та пакетні.

З'єднання в статичній мережі є фіксованими, тоді як в динамічній мережі вони можуть змінюватися в процесі роботи мережі за допомогою програмних засобів. При цьому статичні комунікаційні мережі в свою чергу ділять на одно-, двовимірні та гіперкубічні.

Динамічні комунікаційні мережі ділять на шинні (одно та багатошинні) та комутуючі (одноярусні, багатоярусні та координатні), як це показано на рис. 12.23.

Рис. 12.23. Топології комунікаційних мереж

Дві можливі стратегії операцій взаємодії в мережі - це синхронна й асинхронна. У синхронних КММ всі дії жорстко узгоджені в часі, що забезпечується за рахунок єдиного генератора тактових імпульсів (ГТІ), сигнали якого одночасно транслюються у всі вузли. В асинхронних мережах єдиного генератора немає, а функції синхронізації розподілені по всій системі, причому в різних частинах мережі часто використовуються локальні ГТІ.

Залежно від вибраної стратегії комутації розрізняють мережі з комутацією з'єднань і мережі з комутацією пакетів. Як у першому, так і в другому варіанті інформація пересилається у вигляді пакету. Пакет є групою бітів, для позначення якої застосовують також термін повідомлення.

У мережах з комутацією з'єднань шляхом відповідного встановлення комутуючих елементів мережі формується канал від вузла-джерела до вузла-приймача, що зберіга-

ється, поки весь пакет не досягне пункту призначення. Пересилання повідомлень між певною парою вузлів проводиться завжди поодинці і за тим же маршрутом.

В мережі з комутацією пакетів приймається, що повідомлення самостійно знаходить свій шлях до місця призначення. На відміну від мереж із комутацією з'єднань, маршрут від початкового пункту до пункту призначення кожного разу може бути іншим. Пакет послідовно проходить через вузли мережі. Черговий вузол запам'ятовує прийнятий пакет у своєму буфері тимчасового зберігання, аналізує його і робить висновки, що з ним робити далі. Залежно від завантаженості мережі ухвалюється рішення про можливість негайного пересилання пакету до наступного вузла і про подальший маршрут проходження пакету на шляху до пункту призначення. Якщо всі можливі канали для переміщення пакету до чергового вузла зайняті, в буфері вузла формується черга пакетів, яка "розсмоктується" у міру звільнення ліній зв'язку між вузлами (якщо черга також насичується, то відповідно до однієї із стратегій маршрутизації може відбутися так зване "скидання хвоста", тобто відмова від пакетів, що знову поступають).

КММ можна також класифікувати по тому, як в них організоване керування. У деяких мережах, особливо з комутацією з'єднань, прийняте централізоване керування (рис.12.24).

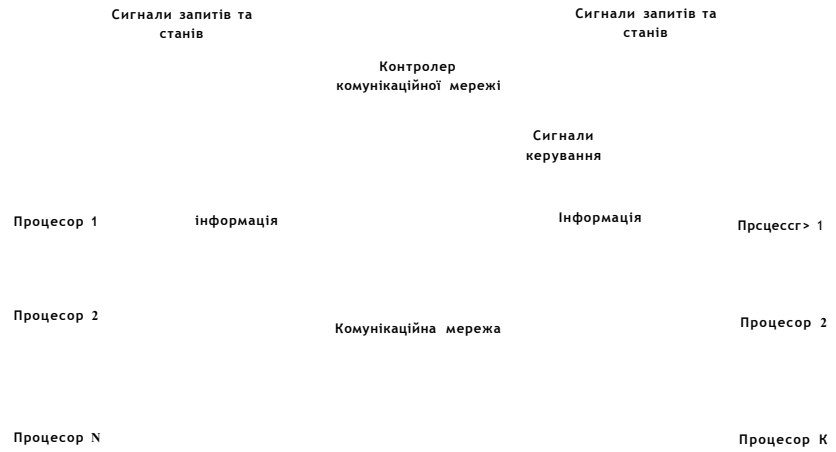


Рис.12.24. Багатопроцесорна система на основі комунікаційної мережі з централізованим керуванням

Процесори посилають запит на обслуговування в контролер комунікаційної мережі, який проводить арбітраж запитів із врахуванням заданих пріоритетів, і встановлює потрібний маршрут. До даного типу слід віднести мережі з шинною топологією. Процесорні матриці також будуються як мережі з централізованим керуванням, яке здійснюється сигналами від центрального процесора. Приведена схема застосовна і до мереж з комутацією пакетів. Тут тег маршрутизації, що зберігається в заголовку пакету, визначає адресу вузла призначення. Більшість серійних комп'ютерних систем має саме цей тип керування.

У схемах із децентралізованим керуванням функції керування розподілені по вузлах мережі. Варіант із централізованим керуванням простіше реалізується, але розширення мережі в цьому випадку пов'язане із значними труднощами. Децентралізовані мережі

в плані підключення додаткових вузлів є значно гнучкішими, проте взаємодія вузлів у таких мережах є істотно складнішою.

У ряді мереж зв'язок між вузлами забезпечується за допомогою деякої множини комутаторів, але існують також мережі з одним комутатором. Наявність великого числа комутаторів веде до збільшення часу передачі повідомлення, але дозволяє використовувати прості комутуючі елементи. Подібні мережі зазвичай будують як багатоярусні.

12.9.2. Основні характеристики комунікаційних мереж багатопроцесорних систем

До основних характеристики комунікаційних мереж багатопроцесорних систем належать наступні:

- розмір;
- складність;
- кількість зв'язків;
- діаметр;
- затримка;
- частота роботи;
- пропускна здатність;
- здатність до розширення;
- надійність.

Розмір мережі визначається кількістю вузлів, які об'єднані мережею.

Складність мережі вимірюється кількістю обладнання, потрібного для її реалізації. При цьому одиницею вимірювання може бути транзистор, вентиль або вузол деякої приведеної складності, наприклад, двовходовий мультиплексор.

Важливою характеристикою мережі є кількість зв'язків (каналів) між вузлами мережі, яка суттєво впливає на складність мережі та її надійність.

Діаметр мережі - число вузлів, через які повинне пройти повідомлення, щоб досягти його місця призначення.

З діаметром тісно пов'язана інша характеристика мережі - затримка, яка рівна часу проходження повідомлення через вузли мережі до місця призначення.

Частота роботи мережі визначає, як швидко дані можуть передаватися через мережу одне за одним.

Із частотою та кількістю зв'язків мережі тісно пов'язана пропускна здатність мережі, тобто кількість інформації, яка може бути передана через мережу за одиницю часу.

Важливою характеристикою мережі є її здатність до розширення шляхом додавання додаткових вузлів. Вартість додавання вузла залежить в значній мірі від кількості компонентів і зв'язків, потрібних для додавання вузла. Для деяких мереж вартість додавання вузла фіксована. Причому, для одних мереж вартість додавання вузлів зростає лінійно або квадратично як функція розміру мережі, для інших - логарифмічно. Вартість часто є обмежуючим чинником для розширення даної мережної топології.

Надійність в значній мірі залежить від надлишковості мережі, яка визначається кількістю різних шляхів, через які повідомлення може пройти від його джерела до місця призначення. Якщо немає ніякої надлишковості, дефектний вузол може зашкодити проходженню всіх або частини повідомлень через нього, та досягненню їх місць призна-

чення. Надлишкова мережа не тільки більш пристосована до дефектів компонентів, але також забезпечує зміну напрямів пересилання повідомлень, і таким чином полегшує вирішення питань блокування в мережі.

12.9.3. Статичні топологи комунікаційних мереж багатопроцесорних систем

Статичні (фіксовані) комунікаційні мережі мають однонаправлені та двонаправлені фіксовані канали зв'язку між процесорами. Може бути виділено два види статичних мереж. Це мережі з повним з'єднанням (CCN - completely connected networks) та мережі з обмеженим з'єднанням (LCN - limited connection networks).

У мережі з повним з'єднанням (повнозв'язних мережах) кожен вузол з'єднаний зі всіма іншими вузлами у мережі. Мережа з повним з'єднанням гарантує швидку доставку повідомлення від будь-якого початкового вузла до будь-якого вузла призначення, оскільки йому доведеться перетнути лише один канал зв'язку. Так як кожен вузол з'єднаний з кожним іншим вузлом в мережі, організація обміну повідомленнями між вузлами стає простим завданням. Мережі з повним з'єднанням є, проте, дорогими, оскільки вимагають великої кількості каналів зв'язку для їх побудови. Ця незручність стає очевидною при великих кількостях вузлів N. Слід зазначити, що число каналів зв'язку в мережі з повним з'єднанням рівне $N(N-1)/2$, тобто асимптотична складність цієї мережі, виражена кількістю каналів зв'язку, є $0(N^2)$. Часова затримка мережі з повним з'єднанням, виражена кількістю пересічених зв'язків, є постійною і рівною 1. Мережа з повним з'єднанням, приведена в якості прикладу на рис. 12.25, має N = 6 вузлів. Як результат, для їх з'єднання потрібно 15 каналів зв'язку.

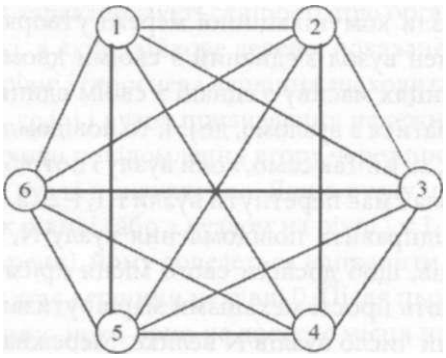


Рис. 12.25. Приклад мережі з повним з'єднанням

Мережі з обмеженим з'єднанням не забезпечують безпосереднього зв'язку кожного вузла з кожним іншим вузлом мережі. Натомість, комунікації між деякими вузлами мають бути здійснені через інші вузли мережі. Довжина шляху між вузлами, виміряна кількістю каналів зв'язку, які доведеться перетнути, є довшою, порівняно з випадком мереж з повним з'єднанням. Крім того, при застосуванні мереж з обмеженим з'єднанням виникає ще дві інших проблеми. Це потреба пошуку ефективної топології комунікаційної мережі і потреба в механізмі маршрутизації повідомлень по мережі, поки вони не досягають своїх місць призначення.

Нижче ці дві проблеми обговорюється детальніше.

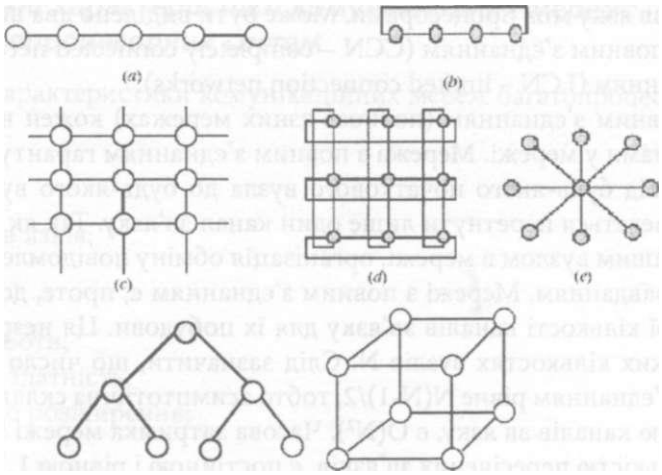
З розвитком архітектури комп'ютерів створено цілий ряд топологій комунікаційних мереж з обмеженим з'єднанням. Ці топології включають:

• одновимірні (лінійні) топології;

• двовимірні топології (кільце, зірка, дерево, решітка);

• тривимірні топології (повнозв'язна топологія, кубічна топологія, гіперкубічні топології).

Прості приклади цих топологій комунікаційних мереж показані на рис. 12.26.



« (a)

Рис. 12.26. Приклади мереж з обмеженим з'єднанням

У лінійній топології вузли комунікаційної мережі утворюють одновимірний масив (рис. 12.26 а), в якому кожен вузол з'єднаний з своїми двома безпосередніми сусідніми вузлами. Два вузли в кінцях масиву з'єднані з своїм єдиним безпосереднім сусідом. Якщо вузлу і потрібно зв'язатися з вузлом j, де j>i, то повідомленню від вузла і доводиться перетнути вузли i+1, i + 2 , j - i . Так само, коли вузлу і потрібно зв'язатися з вузлом), де i>j, то повідомлення від вузла і має перетнути вузли i-1, i - 2 , i - j . У найгіршому випадку, коли вузлу 1 доводиться відправити повідомлення вузлу N, повідомленню доведеться перетнути в сумі N-1 вузлів, щоб досягти свого місця призначення. Тому, хоча лінійні масиви є простими і мають прості механізми маршрутизації, вони є повільними. Це особливо відчувається, коли число вузлів N велике. Мережна асимптотична складність лінійного масиву є O(N) і його часова асимптотична складність є O(N).

Якщо обидва вузли в кінцях лінійної мережі з'єднати, отримається мережа з структурою кільця (рис. 12.26 Ь). Вона може бути однонаправленою та двонаправленою, залежно від кількості каналів зв'язку та напрямку передачі в них даних. Прикладом систем, в яких використано топологію кільця, є мережа Token Ring та система SCI.

Решітчаста (сотова) топологія комунікаційної мережі (рис. 12.26 с) в загальному є n-вимірною матрицею, яка має $K_0 x K, x \ldots x K_n$, вузлів, де п - число вимірів мережі, і Кі є основа виміру і. Існує велика кількість сотових мереж, які відрізняються з'єднаннями вузлів. Наприклад, на рис. 12.26 d показана сотово-кільцева мережа, в якій крайні вузли замкнуті в кільце. Рис. 12.27 показує приклад сотової мережі розміром 3x3x2. Вузол, позиція якого є (i, j, k), з'єднаний із своїми сусідами з позицією i+1, j+I, і к+1. Потрібно від-

значити, що для сотової мережі з N вузлами найдовша відстань між двома довільними вузлами пропорційна №'², тобто її часова асимптотична складність є 0(№/²).

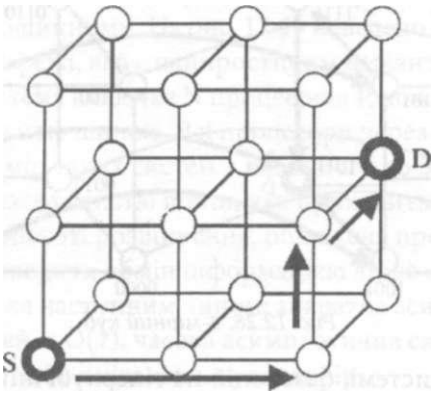


Рис. 12.27. Приклад решітчастої топології

Багатопроцесорні комп'ютерні системи з сотовою топологією комунікаційної мережі є ефективними для виконання наукових обчислень. Іншою перевагою сотових мереж є те, що вони масштабуються. Велика матриця може бути отримана з малої без зміни порядку вузла (кількості входів). Тому багато комп'ютерних систем з розподіленою пам'яттю базуються на таких мережах, наприклад система MPP фірми Goodyear Aerospace, система Paragon фірми Intel, система J-Machine Масачусетського технологічного інституту.

Зіркоподібна топологія (рис. 12.26 e) передбачає об'єднання багатьох вузлів через один центральний вузол і характеризується простотою організації керування.

В деревовидній мережі, в якій двійкове дерево, показане на рис. 12.26 f, є частковим випадком, якщо вузлу на рівні i (коренева вершина знаходиться на рівні 0) потрібно зв'язатися з вузлом на рівні j, де i>j і вузол призначення належить піддереву того ж кореня, то йому доведеться направити повідомлення вгору через прохідні вузли рівнів i-1, i-2, j+I, поки воно не досягне вузла призначення. Якщо вузлу на рівні i потрібно зв'язатися з іншим вузлом на тому ж рівні i (або з вузлом на рівні j ***Ф*** I, де вузол призначення належить піддереву іншого кореня), йому доведеться направити повідомлення вгору дерева, поки повідомлення не досягає вершини на рівні 0. Після цього повідомлення доведеться направити вниз від вершини, поки воно не досягне місця призначення. Слід зазначити, що кількість вузлів в мережі двійкового дерева, що має k рівнів, рівна $N(k) = 2^k - 1$.

Максимальна глибина мережі двійкового дерева рівна $\log_2 N$, де N - число вузлів в мережі. Тому, апаратна асимптотична складність цієї .мережі є $0(2^n)$ і її часова асимптотична складність є 0(log N).

Досить популярними є кубічні (рис. 12.26 g) та гіперкубічні топології, які дозволяють зменшити часову складність статичних топологій мереж.

Гіперкубічні мережі мають структуру n-мірного куба, n-мірний куб (гіперкуб порядку п) визначають як неорієнтований граф, що має 2" мічених вершин від 0 до 2^{r-1}, в якому є дуга між заданою парою вершин лише в тому випадку, якщо двійкове представлення їх адрес відрізняється лише одним бітом. Як приклад, на рис. 12.28 показаний 4-мірний куб.

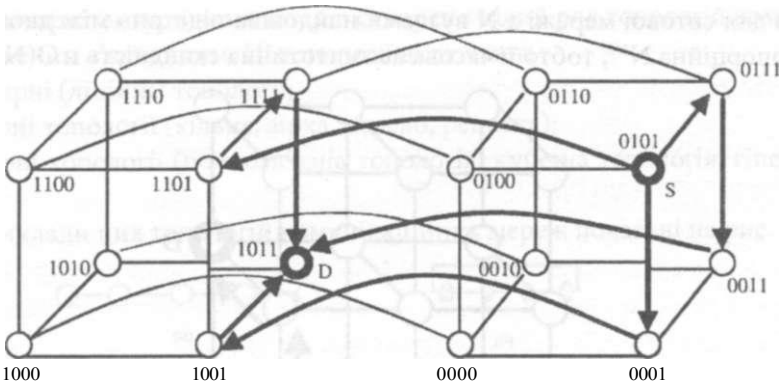


Рис. 12.28. 4-мірний куб

У багатопроцесорній системі, базованій на гіперкубічній мережі, процесори розміщуються у вершинах графа. Дуги графа представляють канали зв'язку між процесорами. Як видно з рисунку, кожен процесор в 4-мірному кубі з'єднаний з чотирма іншими процесорами. У n-мірному кубі кожний процесор має канали зв'язку з п іншими процесорами. Оскільки в гіперкубі є дуга між заданою парою вузлів лише якщо двійкове представлення їх адрес відрізняється одним бітом, ця властивість забезпечує простий механізм маршрутизації повідомлень. Маршрут повідомлення, що відправляється з вузла і та призначене для вузла j, може бути знайдений шляхом виконання над двійковим представленням адрес і **Taj** операції виключного АБО (XOR). Якщо результат виконання операції XOR приводить до 1 в заданій бітовій позиції, то повідомлення має бути посланим по каналу зв'язку, який охоплює відповідний вимір. Наприклад, якщо повідомлення послане від початкового (S) вузла 0101 до вузла призначення (D) 1011, то результатом операції XOR буде 1110. Це означатиме, що повідомлення буде послано лише уздовж вимірів 2, 3, і 4 (рахуючи справа наліво) для того, щоб прибути до місця призначення. Порядок, в якому повідомлення перетинає три виміри, не має важливого значення. Як тільки повідомлення перетинає три виміри в будь-якому порядку, воно досягне місця призначення. Три можливих непересічних маршрути, які може пройти повідомлення у наведеному прикладі, показані виділеними лініями на рис. 12.28. Непересічні маршрути не мають будь-яких спільних каналів зв'язку.

У n-мірному кубі кожен вузол має порядок п. Порядок вузла визначений як число каналів зв'язку, що входять в вузол. Верхня межа кількості непересічних каналів в п-мірному кубі рівна п. Гіперкуб представляється як логарифмічна архітектура. Це тому, що максимальне число каналів зв'язку, які повідомленню доведеться перетнути, щоб досягти місця призначення, в n-мірному кубі, що містить N=2" вузлів, рівне $\log_2 N$.

Одна з бажаних особливостей гіперкубічних мереж - рекурсивна природа їх побудови. n-мірний куб може бути побудований на основі двох підкубів, кожен з яких має порядок п-1, шляхом з'єднання вузлів подібних адрес в обох підкубах. Так 4-мірний куб, показаний на рис. 12.28, побудований на основі двох підкубів, порядок кожного з яких рівний трьом. Потрібно відзначити, що побудова 4-мірного куба на основі двох 3-мірних кубів вимагає збільшення порядку кожного вузла.

Система iPSC фірми Intel є прикладом базованої на гіперкубічній комунікаційній мережі комерційної багатопроцесорної комп'ютерної системи.

12.9.4. Шинні динамічні комунікаційні мережі багатопроцесорних систем

Як ми вже вказували, динамічні комунікаційні мережі багатопроцесорних систем можуть бути одно- та багатошинними. На рис. 12.29 наведено приклад використання одношинної комунікаційної мережі, яка є найпростішим варіантом з можливих з'єднань. В загальному вигляді така система включає N процесорів Р, кожен з яких має свою локальну пам'ять, які з'єднані спільною шиною. Всі процесори через шину взаємодіють з спільною пам'яттю. Типовий розмір таких систем - від одного до 50 процесорів. Він визначається потрібною пропускною здатністю шини, яка приходиться на один процесор. Тому описана архітектура, при простоті розширення, обмежена пропускною здатністю шини, по якій одночасно може проводити обмін інформацією лише один процесор. Складність цієї мережі може бути оцінена наступним чином: апаратна асимптотична складність, виражена кількістю магістралей, є O(I), часова асимптотична складність, яка вимірюється кількістю затримок, є 0(г\'), де N - кількість процесорів.

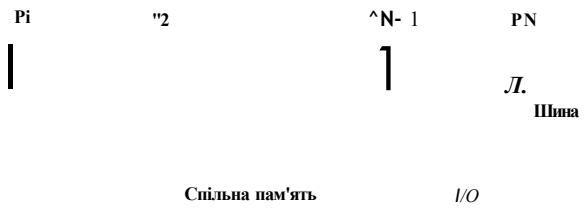


Рис. 12.29. Одношинна багатопроцесорна система

В табл. 12.1 наведено характеристики деяких комерційних одношинних багатопроцесорних комп'ютерних систем.

Таблиця 12.1

Назва системи	Максимальна кількість процесорів	Тип процесора	Частота	Ємність пам'яті	Пропускна здатність шини
HP 9000 K.640	4	PA-8000	180 MHz	-UW6 MB	460 MB/s
IBM RS/6000 R40	8	PowerPC 604	112 MHz	2.048 MB	1.800 MB/s
Sim Enterprise 6000	30	UltraSPARC 1	167 МП/.	30.720 MB	2.600 MB/s

Використання багатьох шин для з'єднання процесорів та блоків пам'яті є природнім розширенням одношинних систем. В цьому випадку можливі кілька варіантів схем з'єднань. Серед них можна відзначити наступні:

• багатошинна комунікаційна мережа з повним з'єднанням блоків пам'яті (multiple bus with full bus-memory connection),

• багатошинна комунікаційна мережа з одним з'єднанням блоків пам'яті (multiple bus with single bus memory connection),

• багатошинна комунікаційна мережа з частковим з'єднанням блоків пам'яті (multiple bus with partial bus-memory connectiez),

• багатошинна комунікаційна мережа з базованим на класах з'єднанням блоків пам'яті (multiple bus with class-based memory connection).

В першому варіанті з'єднання, наведеному на рис. 12.30, всі блоки пам'яті під'єднані до всіх шин. Тут, як і на наступних рисунках, прийнято, що кількість процесорів N = 6, кількість модулів пам'яті M = 4, кількість шин B = 4.

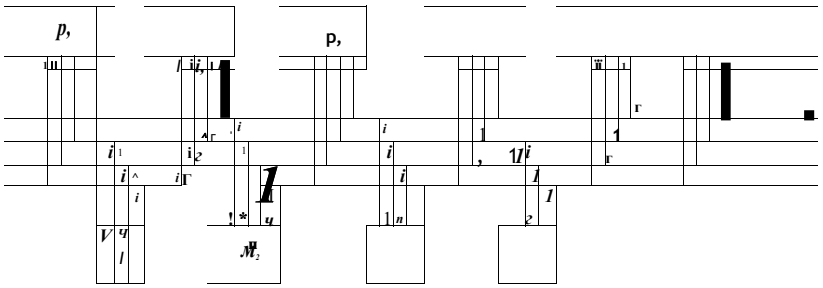


Рис. 12.30. Багатошинна комунікаційна мережа з повним з'єднанням блоків пам'яті

Цей варіант з'єднання є досить складним та дорогим, оскільки вимагає великої кількості шин, реалізація яких вимагає значних апаратних ресурсів.

В другому варіанті (рис. 12.31) кожний модуль пам'яті під'єднаний до деякої однієї шини.

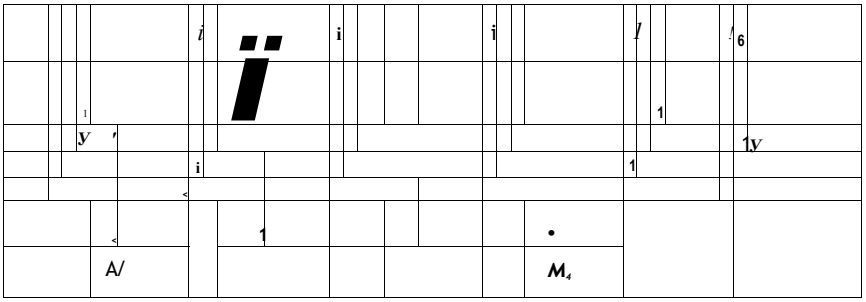


Рис. 12.31. Багатошинна комунікаційна мережа з одним з'єднанням блоків пам'яті

Тут структура комунікаційної мережі спрощується, однак можливі конфлікти доступу до блоків пам'яті.

В третьому варіанті (рис. 12.32) кожен модуль пам'яті під'єднаний не до всіх, як в першому варіанті, а до деякої групи шин (в даному випадку до двох шин).

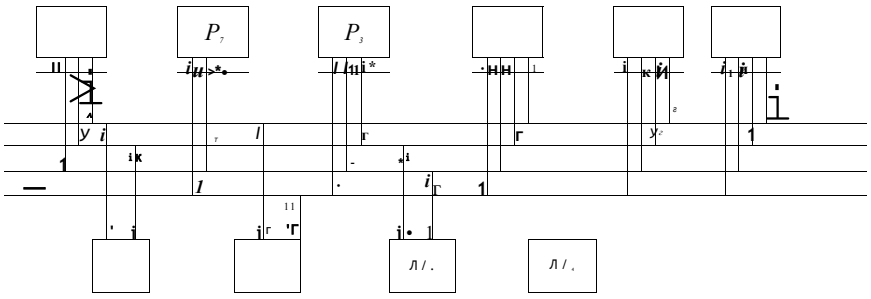


Рис. *12.32. Багатошинна комунікаційна мережа з частковим з'єднанням блоків пам'яті*

Це дозволяє зменшити кількість можливих конфліктів доступу до блоків пам'яті.

В четвертому варіанті (рис. 12.33) модулі пам'яті згруповані в класи, які під'єднані до свого набору шин.

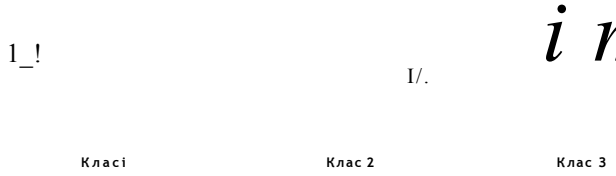


Рис. 12.33. Багатошинна комунікаційна мережа з базованим на класах з'єднанням блоків пам'яті

Тут клас - це довільно вибрана множина модулів пам'яті. Такий підхід також дозволяє зменшити кількість конфліктів шляхом врахування особливостей виконуваних задач (інтенсивності обміну процесорів з пам'яттю). Можна характеризувати цей тип мережі використовуючи кількість необхідних зв'язків і навантаження на кожну шину, як показано в табл. 12.2, де наведені характеристики розглянутих багатошинних з'єднань. У цій таблиці к представляє число класів; g представляє число шин на групу, і M] представляє число модулів пам'яті в класі}.

Таблиця 12.2

Тип з'єднання	Кількість з'єднань	Навантаження на шину і
З повним з'єднанням		
З одним з'єднанням	*BN + M*	
З частковим з'єднанням		***Ы + M/к***
З базованим на класах з'єднанням	***Вїї +*** ***M£1 + B*** *к)*	

В цілому багатошинна багатопроцесорна організація має кілька бажаних особливостей, таких як висока надійність і легкість нарощування. Коли кількість шин є меншою за кількість модулів пам'яті (або кількості процесорів), змагання за володіння шиною посилюється.

Важливим питанням в багатошинній багатопроцесорній організації є шинна синхронізація. Інформація може бути передана по шині синхронно або асинхронно. Час для будь-якої передачі по синхронній шині відомий наперед. При прийманні та відправленні інформації через шину потрібно брати до уваги час передачі. Організація асинхронної шини, з другого боку, залежить від наявності даних і готовності пристрою ініціювати шинні операції.

В одношинній багатопроцесорній системі потрібен шинний арбітраж для вирішення шинного конфлікту, який має місце, коли більш ніж один процесор конкурує за оволодіння шиною. В даному випадку процесори, які хочуть використовувати шину, подають запити до шинної арбітражної логіки. Арбітр вирішує, використовуючи схему пріоритету, якому процесору буде надано доступ до шини протягом певного часового інтервалу.

Процес асинхронної взаємодії одного процесора з іншим називають квітуванням. Він вимагає використання трьох керуючих сигналів: запит шини, надання шини та інформування про її зайнятість. Перший вказує, що процесор подав запит про надання шини, тоді як другий вказує, що шина надана. Третій сигнал зазвичай використовується, щоб вказати, чи зайнятою є шина в даний час. Рис. 12.34 ілюструє механізм квітування шини.

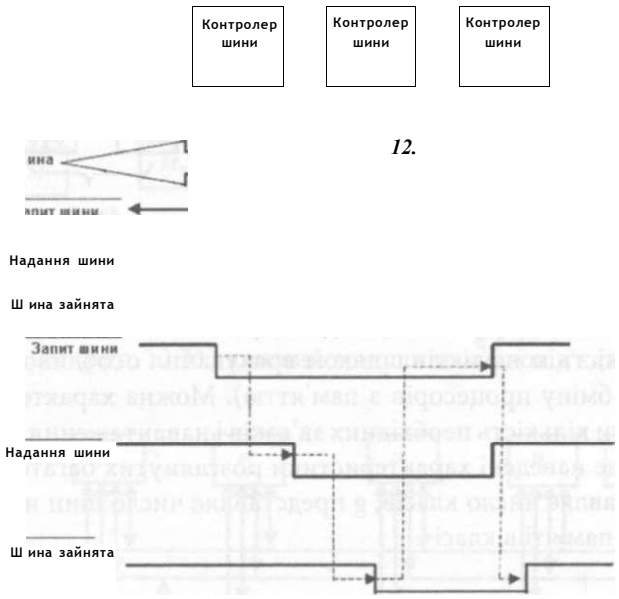


Рис. 12.34. Механізм квітування шини: a - схема, b - часова діаграма

Для прийняття рішення якому процесору надати контроль над шиною, шинна арбітражна логіка використовує вбудовану схему пріоритету. Використовують різні типи схем пріоритету, наприклад випадковий пріоритет, однаковий пріоритет, пріоритет найдовше не використовуваного LRU.

У однаковому пріоритеті, коли два або більше запитів зроблено, є рівний шанс у будь-якого з них. При використанні правила LRU найвищий пріоритет надано процесору, який не використав шину протягом найдовшого часу.

12.9.5. Комутуючі динамічні комунікаційні мережі багатопроцесорних систем

12.9.5.1. Типи комутуючих динамічних комунікаційних мереж

В мережах цього типу з'єднання між процесорами та модулями пам'яті здійснюється використовуючи прості комутуючі елементи.

Всі топології комутуючих комунікаційних мереж багатопроцесорних систем розділяють на три типи: неблокуючі, неблокуючі з реконфігурацією і блокуючі.

У неблокуючих мережах забезпечується з'єднання між будь-якими парами вхідних і вихідних терміналів без зміни режиму роботи комутуючих елементів мережі. В рамках цієї групи розрізняють мережі строго неблокуючі та неблокуючі в широкому сенсі.

У строго неблокуючих мережах виникнення блокувань принципово неможливе через застосовану топологію. До таких належать матрична мережа та мережа Клоса. Неблокуючими в широкому сенсі називають топології, в яких конфлікти при будь-яких з'єднаннях не виникають тільки при дотриманні певного алгоритму маршрутизації.

У неблокуючих мережах з реконфігурацією також можлива реалізація з'єднання між довільними вхідними і вихідними терміналами, але для цього необхідно змінити настройку комутаторів мережі та маршрут зв'язку між сполученими терміналами. Прикладами таких мереж служать мережі Бенеша, Бетчера, "Мемфіс" й інші.

У блокуючих мережах, якщо яке-небудь з'єднання вже встановлене, це може стати причиною неможливості встановлення інших з'єднань. До блокуючих належать мережі "Баньян", "Омега", n-куб і інші.

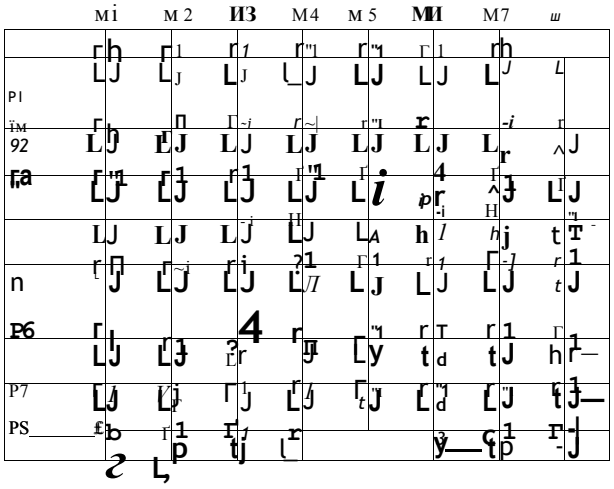
Існує три базові топології комутуючих динамічних комунікаційних мереж багатопроцесорних систем: координатні, одноярусні та багатоярусні.

12.9.5.2. Координатна мережа

Координатна мережа забезпечує одночасне з'єднання всіх входів з усіма виходами. Вона містить комутуючий елемент (КЕ) на перетині будь-яких двох ліній. На рис. 12.35 наведено приклад координатної мережі розміром 8x8. В цьому випадку КЕ знаходяться на кожному з 64 перетинів. На рисунку показано випадок, коли забезпечується одночасне з'єднання між входами Р. та виходами М для і від 1 до 8. Два можливих стани КЕ показано внизу рисунка: прямо та навхрест.

В загальному для координатної мережі розміром NxN кількість КЕ рівна N^2, тобто апаратна асимптотична складність, виражена кількістю КЕ, є $0(N^2)$, тоді як часова асимптотична складність є 0(1).

Потрібно зауважити, що координатна мережа є неблокуючою.



(а) **(Ь)**

Рис. *12.35. Координатна мережа розміром 8x8 та два можливих стани КЕ: а - прямо та Ь - навхрест*

Топологія комутуючої комунікаційної мережі на основі матричного координатного комутатора є класичним прикладом одноярусної динамічної мережі. Головна перевага даної топології полягає в тому, що мережа є неблокуючою і забезпечує меншу затримку передачі повідомлень в порівнянні з іншими топологіями, оскільки будь-який шлях містить тільки один ключ. Проте через те, що число ключів в мережі рівне ИхМ, використання координатного комутатора у великих мережах стає непрактичним, хоча це достатньо хороший вибір для малих мереж. Нижче буде показано, що для великих неблокуючих мереж можна запропонувати інші топології, що вимагають істотно меншої кількості ключів.

12.9.5.3. Матрична одноярусна комутуюча мережа

Матрична одноярусна комутуюча мережа є найшвидшою. Вона забезпечує реалізацію всіх типів з'єднань і є неблокуючою. Структура матричної одноярусної комутуючої мережі наведена нарис. 12.36.

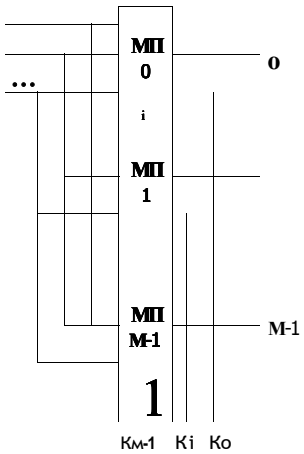


Рис. 12.36. Структура матричної одноярусної комутуючої мережі

Дана мережа складається з М 1М-входових мультиплексорів, які керуються М кодами $К_0$, ..., $К_м$, розрядність кожного з яких рівна к^И. Кожний з мультиплексорів реалізується на основі N двовходових схем І, об'єднаних Ы-входовою схемою АБО. Тому для комутації N входів на М виходів дана комунікаційна мережа повинна містити М груп по N двовходових схем І, об'єднаних Ы-входовою схемою АБО. Витрати обладнання на двовходову схему І, як і на двовходову схему АБО, рівні одному вентилю. Позначимо затримку одного вентиля через 1. Тоді витрати обладнання на дану комунікаційну мережу, без врахування витрат на керування, складуть М^,^ = М(2>1-1) вентилів, а затримка $Т_{кмм}$ = (Ь$_в$Н+1)і.

12.9.5.4. Багатоярусні блокуючі комутуючі мережі

Економнішими порівняно з координатною та матричною КММ є багатоярусні комутуючі мережі, які будуються на базі комутуючого елементу, що зазвичай має два входи і два виходи. Можливі комбінації з'єднання входів з виходами такого КЕ показані на рис. 12.36.

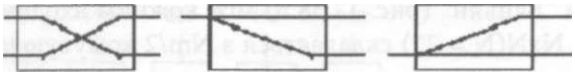


Рис. 12.36. Можливі комбінації з'єднання входів з виходами двовходового комутаційного елемента: а - прямо, Ь - навхрест, с - розширення зверху, сі - розширення знизу

Багатоярусна комунікаційна мережа складається з деякої множини ярусів, побудованих на двоходових КЕ, та об'єднаних між'ярусними зв'язками (МЗ), як це показано на рис. 12.37. Ці зв'язки можуть відображати одну з можливих функцій маршрутизації, таку як батерфляй, куб і т. д.

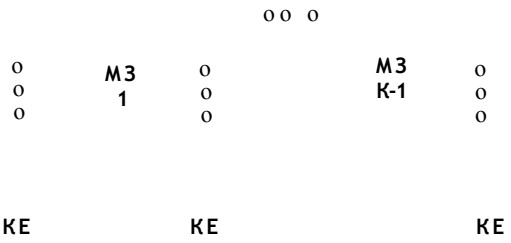
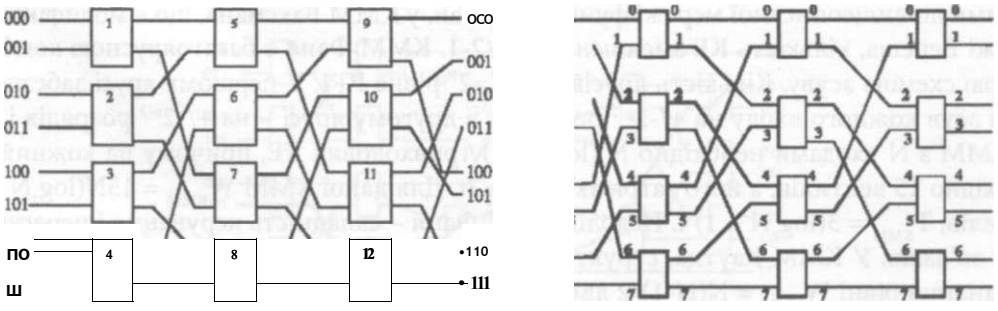


Рис. 12.37. Багатоярусна комутуюча комунікаційна мережа

Існує цілий ряд багатоярусних комутуючих комунікаційних мереж. Структура широко розповсюджених топологій мереж "Баньян" та "Омега" для N=8 показана відповідно на рис. 12.38 а та на рис. 12.38 Ь. Вона містить в кожному з ярусів по N/2 КЕ і N каналів зв'язку. Якщо кожен КЕ виконує перемикання прямо і навхрест, то така комунікаційна мережа може виконати $2^{N/2 \cdot \log_2 N}$ перестановок, що істотно менше за N1 перестановок, можливих в неблокуючій мережі. Проте ті перестановки, які вона виконує, є найбільш використовуваними в багатопроцесорних системах. Потрібно відзначити, що в даній КММ є можливість її розділення на комунікаційні мережі меншого розміру шляхом включення на передачу прямо КЕ в тих ярусах, що стоять перед цими комунікаційними мережами.



а **Ь**

Рис. 12.38. Багатоярусна комунікаційна мережа 8x8: а - "Баньян" та Ь - "Омега'

У топології "Баньян" (рис. 12.38 а) між кожним входом і виходом існує лише один шлях. Мережа MxM(M = 2^r) складається з ІЧт/2 комутуючих елементів. Для керування мережею пакет, що передається, містить в своєму заголовку трирозрядний двійковий номер вузла призначення. Дана мережа належить до мереж з самомаршрутизацією, оскільки адреса пункту призначення не тільки визначає маршрут повідомлення до потрібного вузла, але і використовується для керування проходженням повідомлення по цьому маршруту. Кожен КЕ, до якого потрапляє пакет, проглядає один біт адреси і, залежно від його значення, направляє повідомлення на вихід 1 або 2. Стан КЕ першого ярусу мережі (лівий стовпець КЕ) визначається старшим бітом адреси вузла призначення. Середнім ярусом (другий стовпець) управляє середній біт адреси, а третім ярусом (правий стовпець) - молодший біт. Якщо значення біта рівне 0, то повідомлення пропускається через верхній вихід КЕ, а при одиничному значенні - через нижній. На рисунку показаний маршрут повідомлення з вхідного вузла 2 (010) до вихідного вузла 5 (101).

Топологія "Омега" є підкласом топології "Баньян". Ці топології досить популярні через те, що комутація забезпечується простими КЕ, що працюють з однаковою швидкістю, а повідомлення передаються паралельно. Крім того, великі мережі можуть бути побудовані з мереж меншого розміру.

12.9.5.5. Багатоярусні неблокуючі комутуючі мережі з реконфігурацією

Розглянемо далі багатоярусні КММ, що забезпечують повний набір з N1 перестановок. Дані КММ забезпечують одночасне безконфліктне з'єднання довільного виходу з довільним входом, і тому представляють підвищений інтерес.

Однією з найбільш відомих і вивчених КММ даного класу є КММ Бенеша, структура якої для N=8 наведена на рис. 12.39.

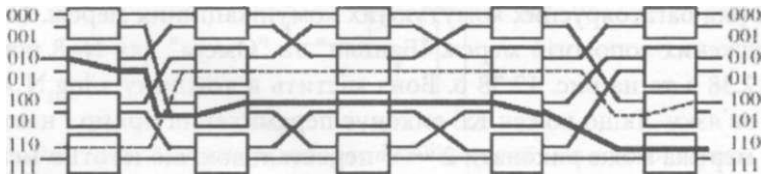


Рис. 12.39. Багатоярусна небпокуюча комутуюча мережа Бенеша з реконфігурацією

Для даної мережі $W_{кмм}$ = N(log_2N - 1/2) двовходових КЕ, або $W_{кмм}$ = 3N(log_2N-l) вентилів, кількість ярусів m=2log_2N-l, а затримка $T_{кмм}$=2(2log_2N-1)t.

Існує цілий спектр інших неблокуючих комутуючих мереж із реконфігурацією з подібними до вищеописаної мережі функціями. Так, у КММ Ваксмана, що є модифікацією мережі Бенеша, кількість КЕ зменшена на N/2-1. КММ Фаня є багатоярусною комбінаційною схемою зсуву. Кількість ярусів для N=2^p рівне P+1. У першому ярусі забезпечується зсув кожного входу на +/-2^{p-1} розрядів, в другому ярусі - на +/-2^{p-2} розрядів і т. д. На КММ з N входами необхідно N (log_2N+l) тривходових КЕ, причому на кожний КЕ необхідно 15 вентилів, а його затримка рівна 3t. Для даної КММ $W_{кмм}$ = 15N(log_2N + 1) вентилів, $T_{кмм}$ = 3(log_2N + 1) t. Недолік КММ Фаня - складність керування і нерегулярність зв'язків. У КММ Каутца, структура якої для N=6 наведена на рис. 12.40, витрати обладнання рівні $W_{кмм}$ = N(N-1)/2 двовходових КЕ, або W = 3N(N-1) вентилів, а затримка T = 2(2N-3) t, яка визначається сумою затримок КЕ по найдовшому шляху проходження даних. Перевага даної мережі - однорідність і регулярність.

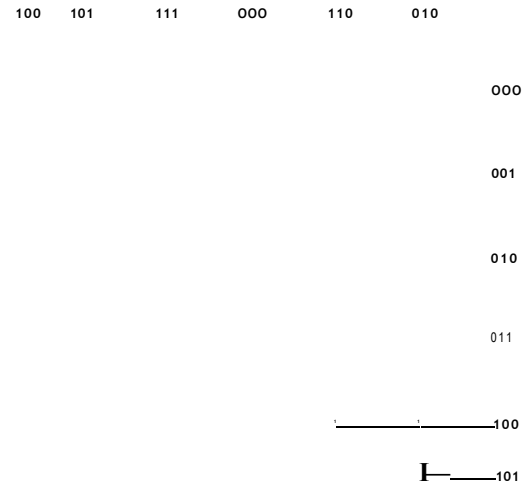
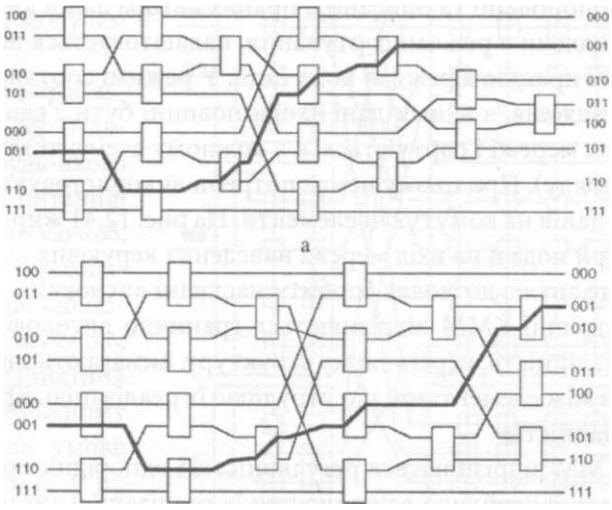


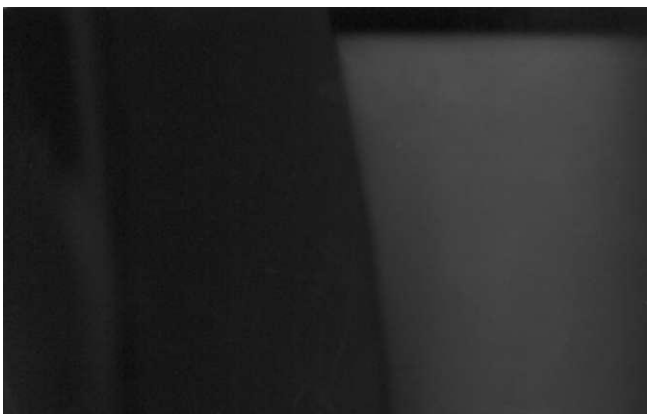
Рис. 12.40. Структура КММ Каутца для n=6

Одночасне довільне з'єднання всіх виходів КММ з всіма входами забезпечують КММ, створені на базі структур сортувальних мереж, що запропоновано та описано в працях автора даної книги. Потрібно відзначити, що можна синтезувати велику кількість структур багатоярусних КММ на базі сортувальних мереж з кількістю ярусів від logjN^ogjN+l)^ до N-1. Вибір конкретної структури залежить від вимог по швидкодії, обмежень за апаратною складністю, а також, можливо, і обмежень на топологію КММ. Розглянемо два крайні випадки даних структур. Структура КММ мінімальної швидкодії з даного класу, назвемо її крайньою зліва, аналогічна структурі відповідного графа сортування (рис. 12.41 а), має характеристики: $W_{кмм} = 6(3'°s^{N} - 2'°s^{N})$ вентилів, $T_{rмм} = 2(N - l)t$. Швидша КММ, назвемо її крайньою справа, відповідна графу сортування Бетчера (рис. 12.41b), має характеристики: $W_{мм} = 6\ 2'°s^{N}\text{-}^{2}(\log^{2}N - \log N + 4) - 1$ вентилів, $T_{rмм} = \log N\ (\log N + l)t$.



Ь

Рис. 12.41. КММ, створені на базі структур сортувальних мереж:
а - крайня зліва, Ь - крайня справа



Проведемо порівняння розглянутих багатоярусних неблокуючих комутуючих мереж з реконфігурацією за апаратною та часовою складностями. В табл. 12.3 наведено кількість КЕ та кількість ярусів залежно від числа входів N розглянутих КММ. Видно, що найвигіднішими за даними критеріями є КММ Бенеша і гранична справа КММ на основі сортувальної мережі Бетчера.

Таблиця 12.3

Тип мережі	Кількість КЕ	Кількість ярусів
Бенеша	N(logN- 1/2)	m=2log,N-l
Каутца	N(N-l)/2	2N-3
Фаня	5/3N(log,N + 1)	log N + 1
Ваксмана	N(logN - 1)	m=2log,N-l
Сортувальна крайня зліва		N - 1
Сортувальна крайня справа	2'°6N-2(i.,2. _ i . . . .)	logN(logN+ 1).2

Розглянемо принципи керування багатоярусними КММ. У матричній одноярусній КММ керуючий код для кожного з N мультиплексорів (приймемо, що кількість входів КММ рівна кількості виходів) займає N розрядів. Він дешифрується дешифратором мультиплексора і пропускає на вихід КММ інформацію з необхідного входу. У багатоярусних КММ кількість бітів керуючої інформації також рівна №о§ N. але в них $\log_2$N-розрядний код рухається разом з інформаційним кодом і настроює КЕ на необхідний вид перемикань. Раніше був описаний принцип керування багатоярусною мережею "Баньян" (рис. 12.38а). Подібно в КММ "узагальнений куб" тег T=T, T,,..T, формується як сума за модулем два двійкових номерів джерела 5=5,5, „.Б і приймача О = Г) Г5 ,—О,, тобто Т = 8 + Б. Кожний розряд тега поступає на відповідний його номеру КЕ і настроює його на режим роботи прямо, якщо Т =0, або навхрест, якщо Т.=1.

Ефективно розв'язується питання керування в КММ, побудованих на базі сортувальних мереж, що запропоновано та описано в працях автора даної книги. В цьому випадку КЕ, спочатку працюючи в режимі сортування, налаштовується на необхідний режим перемикання, а потім працює в режимі комутації. У режимі сортування на входи КММ поступають адреси виходів, з якими дані входи повинні бути з'єднані. Дані адреси рухаються по елементах мережі і сортуються, а в кожному елементі запам'ятовується стан керуючого коду (розряду). При цьому немає потреби використовувати окремі шини для подачі керуючих сигналів на комутуючі елементи. На рис. 12.41 жирною лінією показано один із маршрутів при подачі на вхід мережі наведених керуючих кодів.

Проведений вище аналіз дозволяє зробити наступні висновки:

• одноярусні матричні КММ відрізняються гранично високою швидкодією, проте мають недостатню надійність, нерегулярну структуру, вимагають здійснення дуже великої кількості зв'язків між елементами, що утруднює їх реалізацію. Крім того, вони мають велику апаратну складність;

• багатоярусні КММ відрізняються регулярністю і однорідністю структури, а також локальністю зв'язків, що особливо важливо при їх реалізації у вигляді НВІС. Крім того, в більшості структур багатоярусних КММ вихід з ладу одного або декількох КЕ практично не впливає на їх працездатність;

• в багатоярусних КММ на базі сортувальних мереж, які характеризуються високими параметрами апаратної та часової складності, ефективно розв'язується питання керування.

12.9.5.6. Багатоярусні неблокуючі комутуючі мережі

У 1953 році Клос показав, що багатоярусна мережа на основі координатних комутаторів, що містить не менше трьох ярусів, може мати характеристики неблокуючої мережі.

Мережа Клоса з трьома ярусами, показана на рис. 12.42, містить $г_1$ координатних комутаторів у вхідному ярусі, т координатних комутаторів в проміжному ярусі і $г_2$ координатних комутаторів у вихідному ярусі. В кожного комутатора вхідного ярусу є $п_1$ входів і т виходів - по одному виходу на кожний координатний комутатор проміжного ярусу. Комутатори проміжного ярусу мають г входів, за кількістю координатних комутаторів вхідного ярусу, і $г_2$ виходів, що відповідає кількості перемикачів у вихідному ярусі мережі. Вихідний ярус мережі будується з координатних комутаторів з т входами і $п_2$ виходами. Звідси зрозуміло, що числа гц.п^г,,^, і т повністю визначають мережу. Число входів мережі N = г п., а виходів - М = глі$_2$.

Зв'язки всередині складеного комутатора організовані за наступними правилами:

• к-й вихід і-го вхідного комутатора з'єднаний з і-м входом к-го проміжного комутатора;

• к-й вхід)-го вихідного комутатора з'єднаний з]-м виходом к-го проміжного комутатора.

Кожен модуль першого і третього ярусів мережі з'єднаний з кожним модулем другого її ярусу.

Хоча в даній топології забезпечується шлях від будь-якого входу до будь-якого виходу, відповідь на питання, чи буде мережа неблокуючою, залежить від числа проміжних ланок. Клос довів, що подібна мережа є неблокуючою, якщо кількість координатних комутаторів в проміжному ярусі т задовольняє умові: т = $п_1$ + $п_2$ - 1. Якщо $п_1$ = $п_2$, то матричні перемикачі в проміжному ярусі є повними координатними комутаторами і

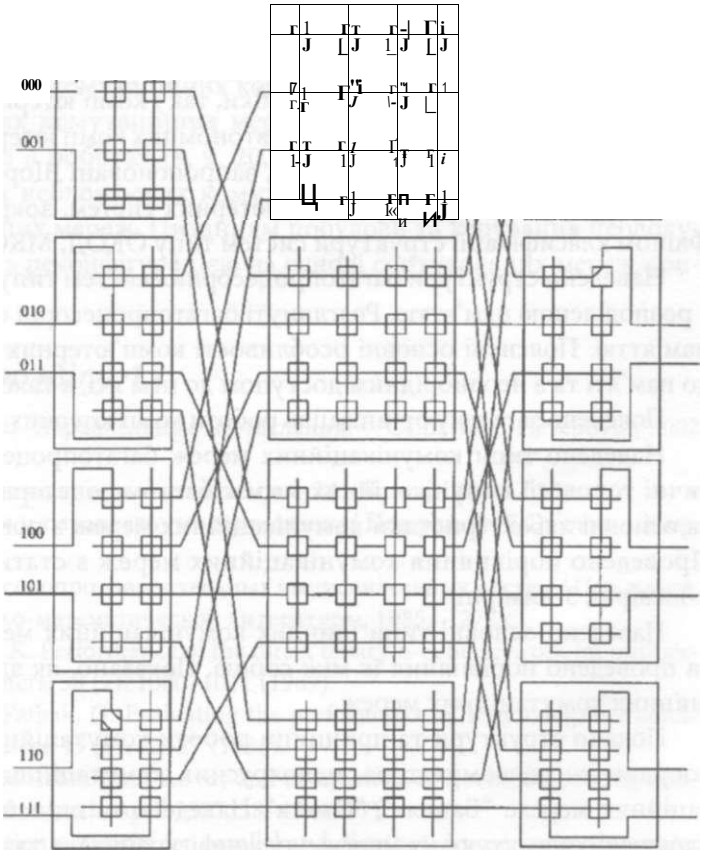


Рис. 12.42. Триярусна мережа Клоса

критерій неблокованості набуває вигляду: m = 2п -1. За умови m = п, мережу Клоса можна віднести до неблокуючих мереж з реконфігурацією. У всіх інших випадках дана топологія стає блокуючою.

Комп'ютерні системи, в яких з'єднання реалізовані відповідно до топології Клоса, випускають багато фірм, зокрема Fujutsu, Nippon, Hitachi.

На завершення в табл. 12.4 наведено порівняння швидкодії вище розглянутих динамічних комунікаційних мереж.

Таблиця 12.4

Тип мережі	Затримка мережі	Ціна (складність)	Наявність бпокуианмя
Одношинна	WO	0(1)	С
Багатошинна	*(XmN)*	*от*	С
Багатоярусна		*OtN* lag /V)	Є
Координатна	*Oil)*	*0{N²)*	Немає

12.10. Короткий зміст розділу

У розділі розглянуті питання подальшого підвищення продуктивності комп'ютерів шляхом створення паралельних комп'ютерних систем. До таких систем належать як багатопроцесорні комп'ютерні системи, в яких використовується паралелізм розподілу задач на велику кількість вузлів обробки, так і комп'ютерні мережі, які мають іншу форму паралелізму - мережу структурно автономних комп'ютерів. Наведені основні положення класифікації комп'ютерних систем, запропоновані Шором та Фліном. Подано відповідні класифікаціям структури комп'ютерних систем, зокрема відповідні запропонованій Фліном класифікації структури систем типу ОКОД, МКОД, ОКМД та МКМД.

Наведено структури багатопроцесорних систем типу ОКМД та МКМД з спільною та з розподіленою пам'яттю. Розглянуті багатопроцесорні системи типу МКМД з спільною пам'яттю. Пояснені основні особливості комп'ютерних систем з однорідним доступом до пам'яті та з неоднорідним доступом до пам'яті, а також лише з кеш пам'яттю.

Пояснено загальну організацію роботи комп'ютерних систем з розподіленою пам'яттю.

Наведено типи комунікаційних мереж багатопроцесорних систем. Розглянуті статичні топології комунікаційних мереж багатопроцесорних систем. Подано особливості та основні характеристики комунікаційних мереж з повним та з неповним з'єднанням. Проведено порівняння комунікаційних мереж з статичним з'єднанням - 1-вимірні, 2-вимірні, 3-вимірні.

Наведено основні типи шинних комунікаційних мереж багатопроцесорних систем та проведено порівняння їх між собою. Показано, як здійснюється синхронізація шин шинних комутаційних мереж.

Подано структури та принципи роботи комутаційних комунікаційних мереж типу координатного комутатора, одноярусних комутаційних мереж, багатоярусних комутаційних мереж: "Баньян", "Омега". Наведено та оцінено структури багатоярусних неблокуючих комутуючих мереж з реконфігурацією, а також неблокуючих комутаційних мереж.

12.11. Література для подальшого читання

Питання впровадження паралелізму в архітектурі комп'ютера та побудови паралельних комп'ютерних систем розкриті в багатьох підручниках та монографіях, зокрема [1-4, 11-16, 19-22, 24, 25, 27], і в інформаційних матеріалах фірм-виробників, розміщених на їх веб-сторінках, зокрема фірм IBM, Intel, Sun Microsystems, HP і т. д.

Закон Амдала та наслідки з нього, а також аналіз обмежень на кількість процесорів в багатопроцесорних системах наведено в роботі [19].

Багатопотокова обробка, в тому числі технологія гіперпотокової обробки, описана в роботі [19], а також в описах процесора Intel Xeon.

Питання класифікації комп'ютерних систем розглянуті в роботах [1-4, 16, 19, 24] та багатьох інших.

Велика кількість літератури присвячена розгляду принципів побудови систем типу ОКМД та МКМД. В першу чергу слід звернутися до літератури [2-4, 6, 13-15, 17, 18, 20-22, 25, 33-35, 38].

В працях [5, 7, 20-22, 25, 33-35, 38] наведено типи комунікаційних мереж багатопроцесорних систем. Тут же розглянуті статичні топології комунікаційних мереж багатопроцесорних систем, особливості та основні характеристики комунікаційних мереж з повним та з неповним з'єднанням, а також із статичним з'єднанням.

Основні типи шинних комунікаційних мереж багатопроцесорних систем та їх порівняння наведено в [10, 20, 37].

Структури та принципи роботи комутаційних комунікаційних мереж типу координатного комутатора, одноярусних комутаційних мереж, багатоярусних комутаційних мереж: "Баньян", "Омега" подано в роботах [8, 9, 20, 29, 30]. В цих же роботах можна знайти структури багатоярусних неблокуючих комутуючих мереж з реконфігурацією, а також неблокуючих комутаційних мереж. Питанням побудови та керування неблокуючими комутуючими мережами з реконфігурацією на основі сортувальних мереж присвячені роботи [39-41].

12.12. Література до розділу 12

1. Воеводин В. В., Воєводин В. В. Параллельные вычисления. - СПб.: БХВ-Петербург, 2002 - 608 с.

2. Головки Б. А. Параллельные вычислительные системы. - М.: Наука, 1980. - 520 с.

3. Мультипроцессорные системы и параллельные вычисления/ Под ред. Ф. Г. Энслоу. - Мир, 1976. - 384 с.

4. Тербер К. Дж. Архитектура высокопроизводительных вычислительных систем / Пер. с англ. - М.: Наука. Главная редакция физико-математической литературы, 1985. - 272 с.

5. Abraham, S. and Padmanabhan, K. Performance of the direct binary n-cube network for multiprocessors. IEEE Transactions on Computers, 38 (7), 1000-1011 (1989).

6. Agrawal, P., Janakiram, V. and Pathak, G. Evaluating the performance of multicomputer configurations. IEEE Transaction on Computers, 19 (5), 23-27 (1986).

7. Al-Tawil, K., Abd-El-Barr, M. and Ashraf, F. A survey and comparison of wormhole routing techniques in mesh networks. IEEE Network, March/April 1997, 38-45 (1997).

8. Bhuyan, L. N. (ed.) Interconnection networks for parallel and distributed processing. Computer (Special issue), 20 (6), 9-75 (1987).

9. Bhuyan, L. N., Yang, Q. and Agrawal, D. P. Performance of multiprocessor interconnection networks. Computer, 22 (2), 25-37 (1989).

10. Chen, W.-T. and Sheu, J.-P. Performance analysis of multiple bus interconnection networks with hierarchical requesting model. IEEE Transactions on Computers, 40 (7), 834-842 (1991).

11. Dasgupta, S. Computer Architecture: A Modern Synthesis, vol. 2; Advanced Topics, John Wiley, 1989.

12. Decegama, A. The Technology of Parallel Processing: Parallel Processing Architectures and VLSI Hardware, Vol. 1, Prentice-Hall, 1989.

13. Dongarra, J. Experimental Parallel Computing Architectures, North-Holland, 1987.

14. Duncan, R. A survey of parallel computer architectures. Computer, 23 (2), 5-16 (1990).

15. El-Rewini, H. and Lewis, T. G. Distributed and Parallel Computing, Manning & Prentice Hall, 1998.

16. Flynn. Computer Architecture: Pipelined and Parallel Processor Design, Jones and Bartlett, 1995.

17. Goodman, J. R. Using cache memory to reduce processor-memory traffic. Proceedings 10th Annual Symposium on Computer Architecture, June 1983, pp. 124-131.

18. Goyal, A. and Agerwala, T. Performance analysis of future shared storage systems. IBM Journal of Research and Development, 28 (1), 95-107 (1984).

19. Hennessy, J. and Patterson, D. Computer Architecture: A Quantitative Approach, Morgan Kaufmann, 1990.

20. Hesham El-Rewini Mostafa Abd-El-Barr. ADVANCED COMPUTER ARCHITECTURE AND PARALLEL PROCESSING. John Wiley, 2005

21. Hwang, K. and Briggs, F. A. Computer Architecture and Parallel Processing, McGraw-Hill, 1984.

22. Ibbett, R. N. and Topham, N. P. Architecture of High Performance Computers II, Springer-Verlag, 1989.

23. Juang, J.-Y. and Wah, B. A contention-based bus-control scheme for multiprocessor systems. IEEE Transactions on Computers, 40 (9), 1046-1053 (1991).

24. Flynn M.F. Some computer organizations and their effectiveness. - IEEETC, 1972, September. P. 848-960.

25. Lewis, T. G. and El-Rewini, H. Introduction to Parallel Computing, Prentice-Hall, 1992.

26. Linder, D. and Harden, J. An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes. IEEE Transactions on Computers, 40 (1), 2-12 (1991).

27. Moldovan, D. Parallel Processing, from Applications to Systems, Morgan Kaufmann Publishers, 1993.

28. Ni, L. and McKinely, P. A survey of wormhole routing techniques in direct networks. IEEE Computer, February 1993, 62-76 (1993).

29. Patel, J. Performance of processor-memory interconnections for multiprocessor computer systems. IEEE Transactions, 28 (9), 296-304 (1981).

30. Reed, D. and Fujimoto, R. Multicomputer Networks: Message-Based Parallel Processing, MIT Press, 1987.

31. Serlin, O. The Serlin Report On Parallel Processing, No. 54, pp. 8-13, November 1991.

32. Sima, E., Fountain, T. and Kacsuk, P. Advanced Computer Architectures: A Design Space Approach, Addison-Wesley, 1996.

33. Stone, H. High-Performance Computer Architecture, 3rd ed., Addison-Wesley, 1993.

34. The Accelerated Strategic Computing Initiative Report, Lawrence Livermore National Laboratory, 1996.

35. Wilkinson, B. Computer Architecture: Design and Performance, 2nd ed., Prentice-Hall, 1996.

36. Yang, Q. and Zaky, S. Communication performance in multiple-bus systems. IEEE Transactions on Computers, 37 (7), 848-853 (1988).

37. Youn, H. and Chen, C. A comprehensive performance evaluation of crossbar networks. IEEE Transactions on Parallel and Distribute Systems, 4 (5), 481-489 (1993).

38. Zargham, M. Computer Architecture: Single and Parallel Systems, Prentice-Hall, 1996.

39. Мельник А. А., Илькив В. С. Реализация алгоритмов сортировки. Систолические вычислительные структуры. Препринт N3-87. ИППММ АН УССР. - Львов, 1988. - с. 25-26.

40. Мельник А. А. О подходе к реализации многоступенчатых коммутирующих сетей. Высокопроизводительные вычислительные системы. Препринт N6-89. - Львов, 1989. - с. 46-47.

41. Мельник А. О. Принципи організації управління для одного класу багатоступінчастих комутуючих мереж. Матеріали НТК "Досвід розробки та застосування приладо-технологічних САПР мікроелектроніки". - Львів, 1995. - Ч. 1. - с. 28-29.

12.13. Питання до розділу 12

1. Наведіть хронологію нововведень в архітектурі комп'ютера з точки зору паралельної обробки інформації.

2. В якому комп'ютері вперше була використана паралельна пам'ять та паралельний АЛП?

3. В якому комп'ютері вперше використані процесори введення-виведення?

4. Які принципово нові архітектурні рішення були реалізовані в комп'ютері Stretch?

5. В якому комп'ютері вперше був використаний конвеєрний принцип виконання команд, віртуальна пам'ять та система переривань?

6. В комп'ютерах якої серії вперше були використані незалежні конвеєрні операційні пристрої?

7. Наведіть основні архітектурні принципи матричних процесорів та продемонструйте їх застосування на прикладі процесора ILLIAC IV.

8. Дайте оцінку впливу векторно-конвеєрних комп'ютерів фірми CRAY на подальший розвиток комп'ютерів.

9. Виділіть особливості паралельних комп'ютерів з спільною пам'яттю.

10. Наведіть особливості та приклади систем з масовою паралельною обробкою інформації.

11. Наведіть закон Амдала та наслідки, що витікають з цього закону.

12. Яка основна ідея багатопотокової обробки інформації?

13. Які існують основні підходи до реалізації багатопотокової обробки інформації?

14. Опишіть технологію Hyper-Threading, використану в процесорі Хеоп фірми Intel.

15. Наведіть основні положення класифікації комп'ютерних систем, запропоновані Шором.

16. Наведіть структуру машини 1 за класифікацією Шора.

17. Наведіть структуру машини 2 за класифікацією Шора.

18. Наведіть структуру машини 3 за класифікацією Шора.

19. Наведіть структуру машини 4 за класифікацією Шора.

20. Наведіть структуру машини 5 за класифікацією Шора.

21. Наведіть структуру машини 6 за класифікацією Шора.

22. Наведіть принципи класифікації комп'ютерних систем, запропоновані Фліном.

23. Наведіть структуру комп'ютерної системи ОКОД.

24. Наведіть структуру комп'ютерної системи МКОД.

25. Наведіть структуру комп'ютерної системи ОКМД.

26. Наведіть структуру комп'ютерної системи МКМД.

27. Наведіть структуру багатопроцесорної системи типу ОКМД з розподіленою пам'яттю.

28. Наведіть структуру багатопроцесорної системи типу ОКМД з спільною пам'яттю.

29. Поясніть структуру та організацію роботи багатопроцесорної системи типу МКМД з спільною пам'яттю.

30. Поясніть структуру та організацію роботи багатопроцесорної системи типу МКМД з розподіленою пам'яттю.

31. Наведіть класифікацію багатопроцесорних систем типу МКМД з спільною пам'яттю.

32. Поясніть основні особливості комп'ютерних систем з однорідним доступом до пам'яті.

33. Поясніть основні особливості комп'ютерних систем з неоднорідним доступом до пам'яті.

34. Поясніть основні особливості комп'ютерних систем лише з кеш пам'яттю.

35. Поясніть загальну організацію роботи комп'ютерних систем з розподіленою пам'яттю.

36. Наведіть типи комутаційних мереж багатопроцесорних систем.

37. Які існують статичні топології комутаційних мереж багатопроцесорних систем.

38. Наведіть особливості та основні характеристики комутаційних мереж з повним з'єднанням.

39. Наведіть особливості та основні характеристики комутаційних мереж з неповним з'єднанням.

40. Порівняйте комунікаційні мережі з статичним з'єднанням - 1-вимірні, 2-вимірні, 3-вимірні.

41. Наведіть основні типи шинних комутаційних мереж багатопроцесорних систем та порівняйте їх між собою.

42. Як здійснюється синхронізація шин шинних комутаційних мереж?

43. Як працює комунікаційна мережа типу Crossbar?

44. Наведіть структури одноярусних комутаційних мереж.

45. Наведіть структури багатоярусних комутаційних мереж: "Баньян", "Омега".

46. Наведіть структури комутаційних мереж з реконфігурацією.

47. Наведіть структури неблокуючих комутаційних мереж.