МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

МИКОЛАЇВСЬКИЙ НАЦІОНАЛЬНИЙ АГРАРНИЙ УНІВЕРСИТЕТ

Факультет менеджменту

Кафедра економічної кібернетики, комп'ютерних наук та інформаційних технологій

КОМП'ЮТЕРНА СХЕМОТЕХНІКА ТА АРХІТЕКТУРА КОМП'ЮТЕРІВ

Конспект лекцій

для здобувачів першого (бакалаврського) рівня вищої освіти ОПП «Комп'ютерні науки» спеціальності 122 «Комп'ютерні науки» денної форми здобуття вищої освіти

Миколаїв

2025

УДК 004.2+621.38+681.3
К63

Друкується за рішенням науково-методичної комісії факультету менеджменту Миколаївського національного аграрного університету від 27 березня 2025 року, протокол № 7.

Укладач:

О. О. Жебко – асистент кафедри економічної кібернетики, комп'ютерних наук та інформаційних технологій, Миколаївський національний аграрний університет

Рецензенти:

І.О. Калініна – д.т.н., професор кафедри інтелектуальних інформаційних систем Чорноморського національного університету імені Петра Могили

В.В. Поживатенко – канд. фіз.-мат. наук, старший викладач кафедри вищої та прикладної математики

Комп'ютерна схемотехніка та архітектура комп'ютерів : конспект лекцій для здобувачів першого (бакалаврського) рівня вищої освіти ОПП «Комп'ютерні науки» спец. F3 «Комп'ютерні науки» денної форми здобуття вищої освіти / уклад. О. О. Жебко. Миколаїв : МНАУ, 2025. 142 с.

К63

Конспект лекцій призначений для вивчення теоретичних та прикладних основ комп'ютерної схемотехніки й архітектури комп'ютерів, ознайомлення з принципами побудови, функціонування та взаємодії апаратних компонентів сучасних обчислювальних систем. Містить навчальні матеріали з основних тем курсу «Комп'ютерна схемотехніка та архітектура комп'ютерів», що передбачені освітньо-професійною програмою підготовки здобувачів першого (бакалаврського) рівня вищої освіти спеціальності 122 «Комп'ютерні науки», галузі знань 12 «Інформаційні технології».

ЗМІСТ

Передмова

Курс дисципліни «Комп'ютерна схемотехніка та архітектура комп'ютерів» призначений для формування у здобувачів вищої освіти спеціальності F3 «Комп'ютерні науки» системного уявлення про принципи побудови, функціонування та організації обчислювальної техніки, зокрема цифрових схем, процесорів, пам'яті та взаємодії компонентів апаратного забезпечення.

Предметом вивчення дисципліни є закономірності функціонування логічних елементів, архітектура комп'ютерів різних типів, структурні рівні обчислювальних систем, засоби схемотехнічного моделювання та принципи побудови сучасних комп'ютерних систем.

Об'єктом вивчення є апаратна частина комп'ютерів — елементи цифрових схем, процесорна логіка, пам'ять, шини та засоби керування, а також методи їх проектування, оптимізації та моделювання.

Метою викладання дисципліни є підготовка висококваліфікованих фахівців, здатних аналізувати архітектуру комп'ютерів, проектувати цифрові схеми, розуміти принципи побудови апаратного забезпечення й ефективно використовувати ці знання у професійній діяльності в галузі ІТ.

Основними завданнями, що мають бути вирішені в процесі викладання дисципліни, є:

– ознайомлення студентів із базовими компонентами комп'ютерної системи та їх функціональним призначенням;

– навчання методам логічного синтезу та оптимізації цифрових схем;

– формування практичних навичок аналізу архітектури комп'ютера;

– розвиток вмінь застосовувати отримані знання у моделюванні та побудові комп'ютерних систем.

Конспект лекцій складено з урахуванням чинної освітньо-професійної програми, на основі провідних підручників, посібників та нормативних документів.

ЗМІСТОВИЙ МОДУЛЬ 1
ОСНОВИ КОМП'ЮТЕРНОЇ ТЕХНІКИ ТА СХЕМОТЕХНІКИ

ТЕМА 1.1. ІСТОРІЯ РОЗВИТКУ ТА ПОКОЛІННЯ ЕОМ

План

1. Арифмометри докомп'ютерної доби.
2. Перші ЕОМ - ABC, Z3, Colossus, Mark I.
3. Чотири покоління ЕОМ.
4. Огляд вітчизняних ЕОМ.

Обчислювальні пристрої докомп'ютерної доби

Рахункові пристрої, здатні автоматично виконувати складні математичні обчислення, конструювалися математиками та інженерами протягом багатьох століть. Перші механічні обчислювачі, що виконували арифметичні дії, були створені в середині XVII століття математиками Блезом Паскалем та Готфрідом Лейбніцем. У середині XIX століття Чарльз Бебідж в Кембриджському університеті створив різницеву машину, яка дозволяла виконувати фіксований набір обчислювальних алгоритмів. Їм же було розпочато, але не завершено, будівництво першої аналітичної машини, на якій передбачалося програмувати за допомогою перфокарт.

Відомий санкт-петербурзький математик Пафнутій Чебишев у XIX столітті також конструював ряд механічних обчислювальних пристроїв, які дозволяли складати та множити числа. Наприкінці XIX століття подібні пристрої почали масово випускатися і називалися арифмометрами. Одним із найвдаліших приладів такого роду виявився арифмометр російського механіка Вільгодта Однера, який пізніше випускався в СРСР під маркою «Фелікс».

Ранні ЕОМ

У 30-х - першій половині 40-х років XX століття в низці країн почали активно розроблятися програмовані електронно-механічні обчислювальні пристрої. У 1939 році американськими вченими Джоном Атанасовим і

Кліффордом Беррі була спроектована перша ЕОМ ABC (Atanasoff-Berry Computer) — цифровий пристрій, який не мав механічних складових частин, що рухаються. Пальму першості цієї ЕОМ присудив Федеральний районний суд США 1973 року. В ABC вперше з'явилася двійкова арифметика та елемент електронних схем під назвою тригери. ABC призначалася для вирішення систем лінійних рівнянь і не була програмованою, оскільки не дозволяла зберігати програму в пам'яті і таким чином вимагала для своєї роботи активної участі людини.

У 1941 році німецький інженер Конрад Цузе створив ЕОМ під назвою Z3. Вона, як і ABC, працювала з двійковими даними та використовувала двійкові електронні схеми на основі телефонних електромеханічних реле. Але, на відміну ABC, дана ЕОМ була вже програмованою, тобто могла виконувати складені заздалегідь програми, а чи не вимагала введення кожної окремої команди людиною. Для зберігання програм Z3 використовувала перфоровану стрічку, тобто зовнішній носій. Однак умовних переходів і циклів у машинній мові Z3 не було — там, де вони були потрібні в програмі, було необхідне втручання оператора. Ця ЕОМ використовувалася для розрахунків у галузі літакобудування та управління ракетами. Єдиний екземпляр Z3 був знищений під час бомбардування Берліна в 1945 році.

У 1943 році у Великій Британії була створена ЕОМ під назвою Colossus. Основним елементом реалізації двійкових електронних схем вперше використовувалися електронні лампи. У 1944 року було випущено вдосконалена версія ЕОМ, названа Colossus Mark II. Ця ЕОМ призначалася для розшифровування перехоплених німецьких зашифрованих радіоповідомлень, одержуваних з радіоефіру, — за допомогою цих радіограм німецьке командування керувало діями різних військових з'єднань та координувало їх. Colossus дозволила скоротити час розшифровування повідомлень із тижнів до години. Після Другої Першої світової багато екземпляри цієї ЕОМ, і навіть її креслення було знищено, а секретність із проекту було знято лише 2000 року. Приблизно в цей же час Colossus було відновлено, і виявилося, що швидкість

його роботи співпадає зі швидкістю роботи комп'ютера з процесором Pentium 2 — але, зрозуміло, лише при розв'язанні задач дешифрування даних.

У 1944 році, на замовлення Військово-морського флоту США, компанія IBM створила ЕОМ під назвою Mark I для використання у військових розрахунках. Крім того, Mark I застосовувалась у Манхеттенському проекті зі створення в США атомної бомби. Першу версію комп'ютера було встановлено в Гарвардському університеті. Розробкою керував капітан другого рангу Говард Ейкен. При виготовленні Mark I були використані двійкові електронні схеми на основі електромеханічного реле, ЕОМ важила 4,5 тонни та мала корпус із нержавіючої сталі. Ця ЕОМ була програмованою, вважала програму з паперової перфорованої стрічки, але не підтримувала циклів та умовних пропозицій. Цикли реалізовувалися за допомогою замикання початку та кінця перфорованої стрічки з командами програми. Mark I вперше реалізувала концепцію роздільного зберігання програми та даних, і ця ідея згодом отримала назву «гарвардської архітектури». Також слід зазначити, що ЕОМ Mark I була першою повністю програмованою ЕОМ, яка працювала без участі людини.

Слід зазначити, що потужним стимулом для появи і перших кроків обчислювальної техніки була Друга світова війна, а також гонка озброєнь, що послідувала за нею, розвиток ядерних і космічних програм. Таким чином, виникали нові обчислювальні завдання — для розрахунку термоядерних моделей, різних завдань балістики, навігації, криптографії, радіолокації, а також логістики, телекомунікацій тощо. Ці завдання потребували нових методів розв'язання. При цьому ресурси та бюджети на створення нових рішень для цих завдань були фактично необмежені.

Покоління ЕОМ

Історію ЕОМ прийнято розбивати на так звані покоління, які змінювалися з появою нових технологій виробництва ЕОМ.

До першого покоління відносять ЕОМ, створені у другій половині 40-х XX століття. Ці ЕОМ ґрунтувалися на електромеханічних реле, лампових

діодах та тріодах. Дані пристрої використовувалися для створення логічних елементів та схем керування перших комп'ютерів, дозволяючи виконувати обчислювальні процеси. Прикладами таких комп'ютерів є ENIAC (США), EDVAC (США), Z4 (ФРН, спадкоємець Z3), МЕСМ (СРСР), "Урал" (СРСР), "Стріла" (СРСР). Проте загальноприйняті принципи конструювання ЕОМ ще вироблено. Деякі з перерахованих ЕОМ використовували десяткову арифметику (наприклад, ENIAC), деякі — двійкову, але з незвичайним розміром машинного слова (43 біта, «Стріла») тощо. Ці ЕОМ були дуже ненадійними і часто не могли безперебійно відпрацювати протягом однієї доби. Великі фізичні розміри, значна кількість деталей, високе енергоспоживання, складність обслуговування (насамперед, громіздкі та дорогі системи охолодження) — все це було причиною того, що ЕОМ у цей час були доступні лише великим науково-дослідним інститутам, військовим та державним установам. Але швидкодія ЕОМ вимірювалася тисячами операцій на секунду, що з людини з арифмометром було вже недосяжно. Ставало все очевиднішим, що подальший прогрес науки і техніки вже однозначно пов'язувався з використанням та розвитком ЕОМ.

Прикладом ЕОМ першого покоління є ENIAC (Electronic Numerical Integrator and Computer). Ця ЕОМ була практично першою, яку можна було використати на вирішення широкого спектра завдань. Вона була випущена в 1945 році в США, мала масу близько 30 тонн і споживала близько 200 кВт (сучасна міська квартира споживає в середньому менше 1 кВт). ENIAC застосовувалася для артилерійських та аеродинамічних розрахунків, а пізніше була використана при розрахунках у галузі ядерної фізики, зокрема при розробці атомної зброї. Серед відомих вчених тієї епохи, які вплинули на розвиток обчислювальної техніки, слід назвати фізика та математика Джона фон Неймана. Беручи участь у розрахунках у рамках Манхеттенського проекту та ґрунтуючись на виявлених недоліках ENIAC, він запропонував низку принципів для конструювання наступної, більш досконалої ЕОМ під назвою EDVAC. Пізніше сукупність цих принципів одержала назву архітектури фон

Неймана. Принципи архітектури фон Неймана, хай і в дещо зміненому вигляді, застосовуються при конструюванні ЕОМ досі.

До другого покоління відносять ЕОМ 1950-1960-х років. У цей час відбувся перехід при побудові електронних схем від електронних ламп та електромеханічних реле до напівпровідникових елементів. Цей перехід дозволив значно підвищити обчислювальну потужність ЕОМ, а також багаторазово зменшити їх розміри, знизити споживану електричну потужність, а також ціну та витрати на обслуговування. У результаті розширився спектр користувачів ЕОМ — їх почали набувати університети та великі промислові підприємства (банки, заводи тощо), а також дедалі більше державних установ. Серед відомих ЕОМ другого покоління можна назвати IBM 7090, створену в 1959 році та призначену для наукових та інженерних розрахунків. Слід також згадати DEC PDP-7, створену 1964 року; дана ЕОМ була універсальною і на той час недорогою. Нарешті, слід назвати високопродуктивні комп'ютери CDC-6600 (1964) і БЭСМ-6 (1968, СРСР).

До третього покоління відносяться ЕОМ 1960-х-70-х років. У цей час відбувся наступний еволюційний крок — з'явилися інтегральні схеми, які містили до кількох тисяч напівпровідникових елементів. Тепер усі електронні схеми ЕОМ розміщувалися на наборі інтегральних схем. Таким чином, принцип модульності зробив ще один крок вперед, що позитивно вплинуло на технології виробництва, дозволивши, зокрема, реалізувати взаємозамінність окремих складових ЕОМ. У результаті вдалося значно збільшити продуктивність і надійність ЕОМ, і навіть зменшити розміри й витрати з їхньої експлуатацію. Прикладами ЕОМ цього покоління є IBM System/360 (IBM S/360), комп'ютери сімейства ЄС ЕОМ у СРСР.

До четвертого покоління відносяться ЕОМ, що випускаються з середини 1970-х років до нашого часу. Головним нововведенням у цих ЕОМ стала наявність мікропроцесора (далі просто процесора), який:

- є центральним блоком ЕОМ і здійснює виконання програм;
- реалізований у вигляді однієї компактної інтегральної схеми.

Розмір цієї інтегральної схеми становив лише кілька сантиметрів, що помітно контрастувало з ЕОМ третього покоління, процесори яких розміщувалися на кількох друкованих платах. Ціна ЕОМ значно знизилася і досягла рівня ціни автомобіля, а пізніше – звичайної побутової техніки. В результаті ЕОМ стали доступні середньому та малому бізнесу, а в 1980-х роках з'явилися і набули широкого поширення персональних комп'ютерів. Першими персональними комп'ютерами стали ЕОМ компаній Apple та IBM (останні отримали назву IBM PC). В даний час персональні комп'ютери виготовляються багатьма компаніями – Dell EMC, Lenovo, Huawei та ін.

Закон Мура

В 1965 інженер Гордон Мур (пізніше він заснував компанію Intel) виявив і сформулював емпіричну закономірність, пізніше названу законом Мура.

Закон Мура стверджує, що кількість напівпровідникових елементів в інтегральних схемах кожні 24 місяці подвоюється, тобто зростає експоненційно.

Це означає, що приблизно так само зростає і продуктивність інтегральних схем, а отже, і продуктивність комп'ютерів загалом.

Наслідком закону Мура є твердження, що продуктивність комп'ютерів кожні 24 місяці подвоюється, тобто зростає експоненційно.

З моменту своєї появи і дотепер закон Мура виконується. Однак у 1960-1970-х роках і зараз він реалізується різними шляхами. Наразі фізичні закони вже не дозволяють нарощувати продуктивність електронних схем за рахунок більш «щільного» розміщення транзисторів на одній кремнієвій пластині. Так відбувається тому, що розміри транзисторів стають порівнянними з розмірами окремих атомів, а швидкість їхньої взаємодії наближається до швидкості світла. Робити транзистори менше атомів, а швидкість їх взаємодії — швидше за швидкість світла неможливо. Значно збільшувати розмір кремнієвої пластини також не виходить. Але таке збільшення і не є виходом, оскільки важливою є вимога збереження невеликих розмірів, а також подальшої

мініатюризації — як самих комп'ютерів, так і обчислювальних пристроїв, що вбудовуються в різні механічні прилади.

Тому в даний час конструктори нових процесорів йдуть шляхом розпаралелювання. Зокрема, створюються багатоядерні процесори: на одній кремнієвій пластині розміщується до кількох десятків мікропроцесорів (ядер), що працюють паралельно, що значно прискорює час роботи підсумкового процесора. З іншого боку, такий підхід ускладнює завдання і конструкторам процесорів, і програмістам — організація паралельних обчислень потребує істотних інтелектуальних витрат. Однак на даний момент цей підхід є основним способом продовжувати нарощувати обчислювальну потужність ЕОМ.

Огляд вітчизняних ЕОМ

Говорячи про вітчизняні ЕОМ першого покоління, вкажемо насамперед на МЕСМ (Мала Електронна Рахункова Машина), яка була створена в 1951 році С. А. Лебедєвим і була першою ЕОМ загального призначення в СРСР та в континентальній Європі. Практично одночасно з МЕСМ була створена ЕОМ М-1, яка мала схожі з МЭСМ характеристиками, але в ній поряд з лампами широко застосовувалися напівпровідникові діоди. Пізніше було створено БЭСМ-1 (1955 р.), яка підтримувала арифметику з плаваючою комою, і навіть БЭСМ-2 (1958 р.), у якій широко застосовувалися напівпровідникові діоди.

ЕОМ другого покоління БЕСМ-6 (1966–1983 рр.) підтримувала локальний паралелізм на базі конвеєра, віртуальну пам'ять, багатозадачність та захист програм. Лампові ЕОМ оборонного призначення М-40 та М-50 (кінець 1950-х рр.) були сконструйовані для управління комплексу протиракетної оборони (ПРО) СРСР. Пізніше для використання в системах ПРО були сконструйовані повністю напівпровідникові ЕОМ Е92б (1961) і 5Е51 (1965). ЕОМ «Сетунь» (1959 р.) та «Сетунь-70» (1970 р.) були засновані на трійковій логіці. «Сетунь» є єдиною у світі серійною троїчною ЕОМ. Мінськ-222 (1966 р.) була першою у світі розподіленою обчислювальною

системою, вона була створена на базі ЕОМ другого покоління «Мінськ-2»/«Мінськ-22» і мала високу гнучкість і параметризованість.

Більшість радянських ЕОМ третього та четвертого поколінь (1970–1980-ті роки) копіювали іноземні, що дозволило знизити витрати на їх створення та підвищити комп'ютеризацію промисловості в СРСР. Проте це негативно позначилося розвитку вітчизняної обчислювальної техніки. Тому оригінальні вітчизняні розробки цієї епохи заслуговують на особливу увагу.

В академії наук СРСР у 1970–1990-х роках розроблялося сімейство високопродуктивних комп'ютерів «Ельбрус» (керівники проекту — В. С. Бурцев та Б. А. Бабаян). ЕОМ сімейства «Ельбрус» призначалися для обробки великих обсягів даних. Ці ЕОМ, звичайно, не були призначені для невеликих організацій або приватних осіб: техніка такого класу була доступна лише великим науковим та оборонним центрам. Маючи високу продуктивність, вони використовувалися в ядерних дослідницьких центрах, а також, з кінця 1980-х років, використовувалися в системі протиракетної оборони.

Перший в континентальній Європі комп'ютер був створений в Україні понад 60 років тому і введений в експлуатація 21 грудня 1951 року. Перша ЕОМ називалася Малою електронною лічильною машиною – «МЕОМ». Незважаючи на скромне слово «Мала», вона налічувала 6000 електронних ламп і ледь вмістилася в лівому крилі будівлі гуртожитку колишнього монастирського селища Феофанія в 10 км від Києва. Машина була створена в лабораторії обчислювальної техніки Інституту електротехніки АН УРСР під керівництвом академіка Сергія Олексійовича Лебедєва.

Інституту електротехніки було виділено напівзруйновану будівлю колишнього монастирського готелю в передмісті Києва - Феофанії. На першому її поверсі і почалися роботи з проектування тоді секретної електронної лічильної машини. Найбільша кімната відводилася для майбутнього дітища С.О. Лебедєва.

Машина займала найбільшу кімнату площею 60 м² в лівому крилі лабораторії у Феофанії і працювала з небувалою на ті часи швидкістю - 3

тисячі операцій за хвилину (для порівняння, сучасні комп'ютери виконують мільйони операцій за секунду) і могла виконувати операції віднімання, додавання, множення, ділення, зсуву, порівняння з урахуванням знака, порівняння за абсолютною величиною, передачі керування, передачі чисел з магнітного барабану, складання команд. Перший запуск «МЕОМ» запам'ятався саме «ламповою проблемою»: при включенні машини 6000 ламп, що запрацювали одночасно, перетворили приміщення в тропіки. Технікам довелося терміново розбирати стелю, щоб відвести з кімнати хоча б частину тепла.

В розробці і створенні «МЕОМ» брали участь 12 інженерів (разом з С.О.Лебедєвим), яким допомагали 15 техніків та монтажниць. «МЕОМ» використовували в багатьох наукових дослідженнях аж до 1957 року, потім машину розібрали на частини, які передали в Політехнічний інститут у Києві для проведення лабораторних робіт. У 1952 році «МЕОМ» була практично єдиною в країні ЕОМ, на якій вирішувалися різноманітні науково-технічні завдання в галузі термоядерних процесів, космічних польотів і ракетної техніки, ліній електропередач, механіки, статистичного контролю якості. Однією з найважливіших задач, яка була розв'язана на «МЕОМ» в цей період, були розрахунки стійкості паралельної роботи агрегатів Куйбишевської гідроелектростанції, які визначаються системою нелінійних диференціальних рівнянь другого порядку.

Потрібно було визначити умови, за яких максимально можлива потужність може передаватися без порушення стійкості системи. Крім того, у зв'язку з швидким розвитком реактивної та ракетної техніки на МЕОМ доводилось розв’язувати і задачі зовнішньої балістики. Це були задачі різної складності, починаючи з відносно простих багатоваріантних розрахунків траєкторій, що проходять в межах земної атмосфери при незначному перепаді висот, до дуже складних, пов'язаних з польотом об'єктів за межами земної атмосфери. «МЕОМ» використовували в багатьох наукових дослідженнях аж до 1957 року, потім машину розібрали на частини, які передали в

Політехнічний інститут у Києві для проведення лабораторних робіт. Результати, отримані в ході роботи над МЕОМ, були використані Лебедєвим для створення Великої Електронної Обчислювальної Машини ВЕОМ-1. Вона була найшвидшою машиною в Європі і однією з найшвидших електронно обчислювальних машин (ЕОМ) в світі. ВЕОМ-1 була машиною, здатною вирішувати складні математичні завдання, замінюючи тисячі обчислень. Машина безперечно внесла величезний внесок у розвиток атомної енергетики і дослідження космосу.

У довготривалому запам'ятовуючому пристрою (ДЗП) постійно зберігалися деякі константи і підпрограми які часто зустрічаються. Вміст ДЗП нічого не змінювало під час роботи машини. Крім того, машина мала зовнішній накопичувач на магнітних стрічках (НМС) - чотири блоки по 30 тисяч чисел в кожному, а також проміжний накопичувач на магнітному барабані (НМБ) ємністю 5120 чисел зі швидкістю вибірки до 800 чисел в секунду.

Технічні характеристики:

Структура команд триадресна. Система числення двійкова. Спосіб представлення чисел - з плаваючою комою. Розрядність - 39 двійкових розрядів: мантиса - 32 розряду, знак числа - 1 розряд, порядок - 5 розрядів, знак порядку - 1 розряд. Діапазон чисел: 10-9 ÷ 10 + 10. Швидкодія - 8000 операцій в секунду.

Характеристики запам'ятовуючого пристрою:

- ємність феритового ОЗП - 1024 числа;
- ємність ДЗП - 1024 числа;
- ємність зовнішнього НМС - чотири блоки по 30 000 чисел в кожному;
- швидкість обміну з НМС - 350 чисел в секунду;
- ємність зовнішнього НМБ - 5120 чисел;
- швидкість обміну з НМБ - 800 чисел в секунду.

Введення інформації в машину з фотозчитувального пристрою на перфокарті зі швидкістю введення 20 чисел на секунду. Друкування результатів на електромеханічний принтер зі швидкістю 1,5 числа на секунду і на фотодрукувальному пристрої зі швидкістю 200 чисел на секунду.

Машина побудована на стандартних дво- і чотирьохлампових комірках. Кількість електронних ламп близько 5000. Живлення машини від мережі трифазного змінного струму напругою 220 В ± 10%, частотою 50 Гц. Споживана потужність близько 30 кВт (без системи охолодження).

Займана площа - до 100 м². Для машини ВЕОМ розроблена система контрольних завдань-тестів, що дозволяють швидко знаходити несправності в машині, а також система профілактичних випробувань, що дозволяють заздалегідь виявляти місця можливих несправностей.

ВЕОМ-1 була машиною паралельної дії, мала розвинену структуру і організацію зв'язків пристроїв і збалансованість їх характеристик. Принципи її організації та конструкції втілилися і удосконалювалися в наступних ЕОМ, розроблених в СРСР.

ТЕМА 1.2. АРХІТЕКТУРА ФОН НЕЙМАНА

План

1. Принципи архітектури фон Неймана.
2. Системна шина та загальна схема типової ЕОМ, заснованої на архітектурі фон Неймана.
3. Гарвардська архітектура.
4. Комбінування фон Неймановської та гарвардської архітектур у сучасних обчислювальних системах.

Принципи архітектури фон Неймана. Традиційно, під архітектурою ЕОМ розуміють повну і детальну специфікацію інтерфейсу користувача ЕОМ. При цьому користувачами вважалися як люди — переважно програмісти, які працюють з ЕОМ, — так і технічні засоби, які взаємодіють з ЕОМ. Це визначення було дано Фредеріком Бруксом в 1975 році в його знаменитій книзі «Міфічний людино-місяць» і стало загальноприйнятим в індустрії. Однак воно, на наш погляд, виглядає дещо архаїчним і не цілком вписується в контекст нашого курсу.

Архітектурою ЕОМ називатимемо основні принципи організації обчислювальних пристроїв, їх основні вузли та інтерфейси, а також технології та методи, за допомогою яких вони реалізовані.

Більшість сучасних ЕОМ створено на основі архітектури фон Неймана, запропонованої Джоном фон Нейманом та його колегами з Університету Прінстона. Ця архітектура була створена для ЕОМ під назвою EDVAC, що розробляється в рамках Манхеттенського проекту (проект створення атомної бомби в США, здійснювався наприкінці Другої світової війни). Звіт, який описує основні принципи організації цього комп'ютера, був написаний фон Нейманом у 1945 році і, з дозволу американських військових, розісланий до основних провідних світових університетів. Цей звіт зробив вирішальний вплив на розвиток індустрії та досліджень у комп'ютерній сфері. Нижче наведено основні принципи архітектури фон Неймана.

1. Програмне управління роботою ЕОМ.

2. Принцип програми, що зберігається.

3. Принцип адресації пам'яті.

4. Використання двійкової системи обчислення.

5. Ієрархічність пристроїв, що запам'ятовують.

6. Використання умовних переходів. Детально розглянемо ці принципи.

Програмне керування роботою ЕОМ. Будь-яке складне обладнання має інтерфейс для взаємодії з оператором, і чим складніше завдання, для вирішення яких обладнання призначене, тим складніший інтерфейс взаємодії з оператором воно має. За допомогою цього інтерфейсу людина керує обладнанням — приймає рішення, які останнє прийняти не в змозі, проводить діагностику, перезапуск і т. д. Наприклад, багато сучасних літаків мають інтелектуальні системи автопілотування, але навіть вміючи самостійно вирішувати складні навігаційні завдання, ці системи вимагають від екіпажу завдання початкових умов маршруту, окремих параметрів режиму польоту і т. д. ЕОМ також є складним обладнанням, яке допомагає кінцевому користувачеві вирішувати його завдання, які, як правило, вже не лежать в області програмування та обчислювальної техніки, як, наприклад, що розглядається вище завдання керування літаком. Іншим прикладом є численні завдання управління бізнес-організацією — облік різних ресурсів, моніторинг індустріальних процесів та ін. Можна також згадати про мобільні пристрої, які вирішують завдання спілкування людей, допомагають орієнтуватися на місцевості, зручно робити покупки та виконувати платежі тощо.

Принцип програмного управління ЕОМ архітектури фон Неймана стверджує, що управління обчислювальною технікою - мобільними пристроями, ноутбуками, робочими станціями і серверами та ін. - Може здійснюватися оператором не безпосередньо, а за допомогою програм. В ідеалі програми взагалі звільняють оператора (кінцевого користувача) від участі в управлінні роботою ЕОМ, вимагаючи лише завдання початкових даних. Але

навіть останні часто зчитуються програмою автоматично, будучи, як кажуть, параметрами контексту.

Програма, що виконується на ЕОМ, складається з окремих команд (дій), і кожна команда дозволяє здійснити одиничний акт перетворення інформації над деякими даними. Команди, які можуть виконуватись конкретною ЕОМ, складають її машинну мову.

Машинна мова ЕОМ мала володіти виразною силою: потрібно, щоб за її допомогою було можливо і зручно задавати відповідні алгоритми. З іншого боку, він повинен дозволяти ефективне виконання відповідної ЕОМ. Компроміс між наочністю та виразною силою, з одного боку, та обчислювальною ефективністю, з іншого боку, є однією з головних проблем сучасних мов програмування (вважатимемо, що читачеві очевидно, що сучасні мови програмування є надбудовою над машинними мовами ЕОМ, а програми , створені- з їх допомогою, щодо нескладно перетворюються на команди ЕОМ).

Один із піонерів обчислювальної техніки Чарльз Беббідж намагався реалізувати цей принцип ще в XIX столітті, але зазнав невдачі. Технології XX століття дозволили успішно створити складні програмовані пристрої. Відповідно, для них почали розроблятись спеціальні «зрозумілі» їм мови (машинні мови), за допомогою яких стало можливо виразити обчислювальні алгоритми.

Принцип програми, що зберігається. Цей принцип стверджує, що в процесі виконання програма разом зі своїми даними зберігається в ЕОМ, а не передається через операторський інтерфейс по одній команді. При цьому важливим є те, що і код, і дані програми знаходяться в тому самому місці — в оперативній пам'яті. У процесі виконання програми процесор вибирає з оперативної пам'яті команди та його операнди (тобто дані). При цьому можна зробити команду операндом, і тоді над нею можна виконувати різні операції — таким чином, з'являється можливість перетворювати програми при їх виконанні. Також слід зазначити, що принцип програми, що зберігається,

забезпечує однаковий час вибірки команд і операндів з пам'яті, дозволяє використовувати непрямі системи адресації для роботи з пам'яттю довільних розмірів та багато іншого.

Наявність оперативної пам'яті, у якій зберігається як програма, як її дані, була новаторської ідеєю. Наприклад, комп'ютер Mark I був фактично потужним електромеханічним калькулятором, забезпеченим керуючим пристроєм. Виконувана програма безпосередньо зчитувалася з перфострічки, а не завантажувалася в оперативну пам'ять повністю перед запуском, тобто принцип програми архітектури, що зберігається, фон Неймана порушувався. Витрати на покомандне зчитування сильно впливали швидкість виконання програми. Однак на той момент (40-і роки XX століття) це всіх влаштовувало: той факт, що обчислювальний пристрій можна було програмувати, а не подавати йому на вхід кожну команду вручну, вже був значним кроком уперед. Менш знаменитий сучасник комп'ютера Mark I, німецький комп'ютер Z3, створений і загублений під час Другої світової війни, був влаштований аналогічним чином.

Принцип програми, що зберігається, згодом дозволив створити компілятори, операційні системи, різні системні програми, які маніпулюють машинним кодом програм як даними, дозволяючи досягти значної автоматизації в програмному управлінні ЕОМ.

Принцип адресації пам'яті. Даний принцип постулює, що пам'ять ЕОМ (зокрема оперативна пам'ять) влаштована таким чином. Вона є набором пронумерованих осередків, кожна з яких має, таким чином, адресу доступу — номер. Ці комірки називаються словами, у різних ЕОМ розмір слова буває різним (докладно про це буде розказано в наступних лекціях). Команди програми, і навіть її дані перебувають у цих словах послідовно. Процесору в будь-який момент часу є будь-яке слово в пам'яті за номером (адресою). Отже, пам'ять є масивом слів.

Вже пізніше цей принцип дозволив організовувати різні схеми адресації, використовуючи адресні сегменти, непряму адресацію тощо. буд.

Такі схеми, у свою чергу, дозволили реалізувати гнучкі стратегії розміщення програм та даних у пам'яті, а також здійснювати ефективний доступ до них під час роботи процесора.

Використання двійкової системи обчислення. Цей принцип означає, що для представлення команд програм та даних використовується двійкова система числення. Перші комп'ютери використовували десяткову систему обчислення, були також варіанти використання троїчної системи обчислення.

Саме використання двійкової системи дозволило налагодити індустріальне виробництво ЕОМ. Справа в тому, що оскільки в двійковій системі є лише дві цифри (0 і 1), для їх подання може бути використана будь-яка фізична система з двома стабільними станами. Наприклад, тріод і транзистор (відкритий або закритий стан), тригер з двома стійкими станами, імпульсна схема (наявність або відсутність електричного імпульсу) тощо. Робилися спроби конструювання ЕОМ на основі інших систем обчислення (наприклад, трійкової та десяткової) , і було досягнуто цікавих результатів. Але лише для двійкової системи вдалося створити електронні схеми, надійні та недорогі при масовому виробництві.

Цей принцип також був очевидним. Альтернативою двійкової системи обчислення була традиційна десяткова система. Для неї існувало багато засобів, наприклад, алгоритми для арифмометрів, які широко використовувалися в 40–50 роки XX століття. Арифмометри — це невеликі механічні обчислювачі, які дозволяли виконувати 4 головні арифметичні дії та містилися на письмовому столі. Були також створені, опубліковані та широко використовувані таблиці значень функцій та фізичних величин з досить високою точністю у формі десяткових дробів тощо. Люди старшого покоління пам'ятають таблиці Володимира Брадіса, у яких зібрано значення тригонометричних функцій, логарифмів та інші величини для використання в математичних розрахунках з точністю чотири знаки після коми.Таким чином, при переході на двійкову систему потрібно було перетворити налагоджені

десятиліттями кошти та процедури розрахунків, які застосовують у багатьох галузях промисловості та науки.

Ієрархічність пристроїв, що запам'ятовують. Цей принцип стверджує, що ЕОМ повинна мати різні види пам'яті, що мають різний об'єм і швидкодію. Стисло розглянемо різні види пам'яті сучасних ЕОМ.

Спочатку сфокусуємось на довгостроковій та оперативній пам'яті.

Довгострокова пам'ять призначається для тривалого зберігання програм та даних комп'ютера. Фотографії, фільми, документи тощо зберігаються саме в довгостроковій пам'яті. Там же зберігаються всі програми, які встановлені та можуть виконуватися на вашому комп'ютері. Таким чином, зміст довгострокової пам'яті залишається незмінним після вимкнення та включення комп'ютера, чого не можна сказати про інші види пам'яті. Ця пам'ять має великі обсяги та відносно низьку швидкість доступу. Існують такі варіанти реалізації довгострокової пам'яті:

- механічні магнітні диски (жорсткі диски, які часто називають «вінчестерами»);
- твердотільні накопичувачі (SSD);
- зовнішні пристрої - гнучкі і оптичні диски, твердотільна флеш-пам'ять, магнітні стрічки.

Оперативна пам'ять призначена для зберігання програми та її даних у процесі виконання. Тобто програма завантажується з довгострокової пам'яті до оперативної, даючи можливість процесору виконати її. Таким чином, процесор не працює з довгостроковою пам'яттю, що значно зменшило б швидкість виконання програми. Відповідно, оперативна пам'ять має значно менший обсяг, ніж довгострокова, але швидкість доступу до неї значно вища — адже потрібно, щоб процесор міг отримувати наступні команди та значення відповідних змінних дуже швидко. У зв'язку з цим виготовлення оперативної пам'яті використовуються інші технології, ніж виготовлення довгострокової пам'яті.

Взаємодія процесора та оперативної пам'яті здійснюється через *системну шину* – швидкий канал передачі інформації. Незважаючи на ефективну реалізацію, системна шина є принциповим вузьким місцем архітектури фон Неймана, що уповільнює швидкодію комп'ютера. На вирішення цієї проблеми націлена велика кількість різних рішень на різних рівнях організації процесу виконання програми.

Розподіл пам'яті комп'ютера на оперативну та довгострокову був однією з головних інновацій архітектури фон Неймана. Однак у сучасних комп'ютерах ієрархія пам'яті значно складніша. Коротко розглянемо такі види пам'яті:

- регістри процесора;
- кеш-пам'ять процесора;
- зовнішні пристрої пам'яті (дискети, лазерні диски, флеш-картки);
- пам'ять для збереження параметрів комп'ютера.

Процесор має набір регістрів, кожен з яких є невеликим осередком пам'яті (8, 16, 32 або 64 біти). Ці комірки використовуються процесором для арифметичних операцій, а також для виконання інших команд. Команди, оперуючи регістрами, виконуються значно швидше, ніж у ситуації, коли дані беруться із довгострокової пам'яті, оскільки реєстри знаходяться безпосередньо всередині процесора. Наприклад, у процесора Intel 8080 було 7 регістрів, а Intel Itanium — 256 регістрів. Одним із факторів, що впливає на кількість регістрів процесора, є його ціна та масовість виробництва.

Оскільки обсяг пам'яті, який міститься в регістрах, вкрай незначний, а проблема оптимізації повідомлення процесора та оперативної пам'яті в процесі виконання програм є надзвичайно актуальною, використовується ще один вид пам'яті — *кеш-пам'ять*. Це особливий вид швидкої пам'яті, яка знаходиться в процесорі і призначається для зберігання даних, що активно використовуються процесором при виконанні однієї або кількох програм, що дозволяє суттєво прискорити їх виконання. У багатоядерних процесорах цю пам'ять влаштовано ієрархічно — кожне ядро має свою кеш-пам'ять, і весь

процесор також має власну кеш-пам'ять. Докладніше про це буде розказано у наступних лекціях.

Швидкодія пам'яті — швидкість виконання операції читання даних і запису даних, яку забезпечують мікросхеми, що реалізують цю пам'ять.

Висока швидкодія пам'яті (регістри, кеш-пам'ять, оперативна пам'ять) вимагає інших технологій апаратної реалізації, ніж низька швидкодія (жорсткий диск). При цьому швидка пам'ять виявляється дорожчою (причому в різних сенсах), ніж повільна, тому вона має значно менший обсяг. Обсяг пам'яті типового офісного комп'ютера, який ми розглянемо в наступних лекціях, становить від 8 до 16 Гб оперативної пам'яті та від 500 до 2000 Гб — жорсткий диск.

Нарешті, скажімо кілька слів про *зовнішні пристрої пам'яті*, які широко використовувалися і використовуються для різних комп'ютерів. Йдеться про дискети, лазерні диски та флеш-карти. Вони є зовнішніми, переносними – їх легко можна вставити в комп'ютер, що кожен із нас робив багато разів. Поряд з Інтернетом вони використовуються для передачі даних між різними комп'ютерами. Існує кілька поколінь таких пристроїв. Мабуть, першими були гнучкі диски або дискети, що використовуються для персональних комп'ютерів IBM PC. Пізніше з'явилися лазерні диски, щоправда, вони не призначалися для активної передачі даних — записати що-небудь на них можна було, залежно від їх різновиду, одноразово чи обмежену кількість разів, вони використовувалися головним чином для дистрибутивів програм, а також для фільмів та аудіозаписів. Нарешті, флеш-пам'ять (так звані флешки) широко використовуються в даний час і дозволяє переносити фільми, фотографії та аудіозаписи, так і програми, а також різні документи та інші види даних. Флеш-пам'ять зручна, оскільки не скрізь є Інтернет, а іноді буває, що його швидкість недостатня. Також не всі користувачі використовують зовнішні сховища на кшталт Яндекс-диску або Google-Drive, наприклад, з міркувань безпеки. Флеш-пам'ять виготовляється на основі спеціальних транзисторів, які запам'ятовують свій стан (відкритий або закритий) при вимкненні живлення.

Пам'ять для налаштувань комп'ютера використовується для збереження основних параметрів, які використовуються під час його запуску. У персональних комп'ютерах така пам'ять має об'єм в межах кількох кілобайт і зберігає налаштування жорстких дисків, стан адаптера Wi-Fi, налаштування завантаження операційної системи з жорсткого диска або локальної мережі тощо. Окремий вид пам'яті для налаштувань потрібен, оскільки настільки базові налаштування не можуть бути прочитані з жорсткого диска: частина з них використовується при його ініціалізації. А в деяких комплектаціях комп'ютерів, що завантажуються локальною мережею, жорсткого диска може і зовсім не бути.

Підсумовуючи, зауважимо, що наявність різних видів пам'яті в ЕОМ дозволяє знаходити гнучкий компроміс між розмірами (ємністю), швидкодією, ціною та надійністю пристроїв, що запам'ятовують, оскільки для різних завдань потрібні різні значення цих характеристик.

Також слід зазначити, що технології виробництва різних видів пам'яті суттєво різняться.

• Оперативна пам'ять на нижньому рівні зберігає біти даних у вигляді електричних зарядів конденсаторів.

• Твердотільні диски реалізуються на основі флеш-пам'яті, що складається зі спеціального виду транзисторів, що запам'ятовують свій стан.

• Кеш-пам'ять та регістри зберігають дані за допомогою спеціальних електронних схем – тригерів.

• Механічні жорсткі диски використовують магнітну пам'ять.

• Пам'ять для налаштувань може виготовлятися із застосуванням технології CMOS (Complementary Metal-Oxide-Semiconductor), яка базується на тригерах і потребує постійного електроживлення; вона може працювати кілька років від годинної батареї. *Тригер* є найпростішою електронною схемою, яка має два стійкі стани: високий або низький електричний потенціал.

Про різні види пам'яті ми докладніше розповімо у наступних лекціях.

Нарешті, відзначимо, що принцип ієрархічності пам'ятних пристроїв, що розглядається в даному розділі, як і попередні принципи архітектури фон Неймана, на зорі комп'ютерної епохи був абсолютно неочевидним.

Використання умовних переходів. *Умовний перехід — це машинна команда або конструкція мови програмування, яка дозволяє направити виконання програми за різними гілками в залежності від значення певної змінної або результату виконання певної умови.*

У сучасних мовах програмування, а також у машинних мовах існує велика кількість різних умовних переходів. Наприклад, у мові С є конструкція if/then/else, яка дозволяє програмі «розгалужуватися» за значенням деякої логічної умови. Є також команда switch/case, яка дозволяє «розгалужувати» потік виконання програми за умовами, що мають не два значення — істину чи брехню як логічні умови, а деякий спектр значень, тим самим реалізуючи більш ніж два розгалуження.

Крім безпосереднього використання, умовні переходи дуже важливі в циклах. Адже в циклах постійно перевіряється виконання певної умови і, в залежності від його істинності, або запускається наступна ітерація тіла циклу, або відбувається вихід із циклу на наступну після циклу інструкцію. Також умовні переходи використовуються неявно у низці інших конструкцій мов програмування та машинних мовах.

Незважаючи на те, що сьогодні умовні переходи здаються чимось зрозумілим, вони з'явилися далеко не відразу. Наприклад, вже згадані комп'ютери Mark I та Z3 їх не підтримували. Там програми, де був потрібний умовний перехід, комп'ютер мав зупинитися і передати керування оператору.

Системна шина. На рис. 1 представлена схема типової ЕОМ, створеної на основі архітектури фон Неймана. Усі вузли ЕОМ розділені втричі групи — процесор, оперативна пам'ять, периферійні пристрої (жорсткий диск, графічна карта комп'ютера, мережна карта, різні порти — USB, HDMI тощо). Зв'язок процесора з рештою вузлів ЕОМ здійснюється за допомогою уніфікованого каналу доступу — системної шини.

Рис. 1. Загальна схема типової ЕОМ, заснованої на архітектурі фон Неймана

Слід зазначити, що рис. 1 не зовсім точно слідує оригінальній архітектурі фон Неймана — там не було єдиного процесора, а замість нього були керуючий і арифметично-логічний пристрої, а також як периферійні пристрої використовувалися лише пристрої вводу-виводу. Процесор як єдиний пристрій, що інтегрує різні блоки, що у виконанні програм, виник лише у ЕОМ четвертого покоління.

Дамо визначення системної шині.

Системна шина є швидким каналом передачі даних та призначається для забезпечення ефективної комунікації процесора з оперативною пам'яттю та периферійними пристроями ЕОМ.

У перших комп'ютерів системна шина складалася з трьох шин спеціального призначення: шини даних, адресної шини, шини, що управляє.

• Шина даних призначена для передачі даних між процесором, пам'яттю та зовнішніми пристроями.

• Адресна шина призначена для передачі адрес пам'яті та ідентифікаторів зовнішніх пристроїв.

• Керуюча шина призначена для передачі керуючих сигналів між процесором та оперативною пам'яттю. Подібна координація необхідна, оскільки виконання тих чи інших дій потребує часу та пристрої повинні «знати», коли їм необхідно розпочати ту чи іншу дію. Як приклад можна навести координацію процесором зчитування даних оперативною пам'яттю з

шини даних. Отримавши по керуючій шині відповідний сигнал, оперативна «вирішує», що дані на шині даних готові до зчитування, і починає їх читання.

Зазначимо, що в сучасних ЕОМ системну шину організовано інакше, але наведений вище поділ каналів передачі інформації активно застосовується на різних рівнях організації системної шини.

Гарвардська архітектура. Вузьким місцем архітектури фон Неймана є системна шина: вона інтенсивно використовується під час виконання програми, оскільки і код програми та її дані знаходяться в одному місці (оперативна пам'ять), а виконуються в іншому (процесор). Тому її пропускна спроможність багато в чому визначає швидкість виконання програми. Існують різні способи вирішення цієї проблеми, наприклад, кеш-пам'ять, який обговорюватиметься далі, проте вона є принциповою і її повністю вирішити не вдається.

Спільне зберігання програми та її даних в оперативній пам'яті ЕОМ дозволяє суттєво здешевити виробництво ЕОМ та спростити її архітектуру, але не є ефективним за швидкодією. Якщо реалізувати роздільне зберігання програми та її даних, можна пересилати команди програми та відповідні дані у процесор паралельно, що значно прискорює роботу ЕОМ. Такий підхід отримав назву гарвардської архітектури.

Гарвардська архітектура передбачає роздільне зберігання програми та даних в оперативній пам'яті різного виду, а також можливість паралельного зчитування того й іншого процесором за допомогою окремих шин.

Гарвардська архітектура була запропонована Говардом Ейкеном наприкінці 1930-х років під час роботи над комп'ютером Mark I. Крім того, згадуваний нами комп'ютер Z3 також мав гарвардську архітектуру.

Комбінування фон Неймановської та гарвардської архітектур. Суттєвим недоліком гарвардської архітектури є те, що потрібно реалізувати два пристрої оперативної пам'яті та дві незалежні шини: при масовому виробництві це значно підвищує вартість комп'ютера. Однак у ряді випадків це виявляється доцільним та ідеї гарвардської архітектури використовуються

на практиці. Перелічимо кілька типових випадків, у яких застосовується гарвардська архітектура.

1. Швидка кеш-пам'ять багатьох сучасних процесорів поділяється на пам'ять даних та пам'ять для машинного коду. При цьому завантажені дані неможливо виконувати (тобто розглядати їх як код), а завантажений код – міняти (тобто розглядати його як дані). Доступ до різних видів кеш-пам'яті здійснюється через різні внутрішні шини процесора, що дозволяє підвищити його продуктивність, щоправда, ціною ускладнення та подорожчання.

2. У простих спеціалізованих обчислювальних пристроях для керування ліфтами, вузлами сучасних автомобілів, кондиціонерами тощо машинний код записується безпосередньо при виготовленні, і таким чином пристрій може тільки зчитувати його при виконанні, але не може змінювати. Оперативна пам'ять цих пристроїв призначена виключно для даних і не використовується для зберігання машинного коду. Для зберігання машинного коду в таких пристроях зазвичай використовується спеціальна пам'ять об'ємом до кількох сотень кілобайт і її інтегрують у блок процесора, призначений для читання машинного коду. Отже, у разі гарвардська архітектура не ускладнює обчислювальний пристрій, а, навпаки, спрощує і здешевлює його.

3. Для реалізації допоміжних вузлів ЕОМ – контролерів жорстких дисків, графічних адаптерів тощо – також використовуються ідеї гарвардської архітектури. Наприклад, контролер жорсткого диска потрапляє до попереднього пункту, оскільки виконує фіксований набір нескладних функцій. Графічний адаптер виявляється більш складним пристроєм, який може виконувати різні операції, але при цьому машинний код, який він виконує, розміщується в окремій пам'яті, запис в яку здійснює центральний процесор ЕОМ, а сам графічний адаптер лише зчитує цей код для виконання.

ТЕМА 1.3. ДВІЙКОВА СИСТЕМА ЧИСЛЕННЯ ТА МАШИННА АРИФМЕТИКА

План

1. Двійкова та інші системи числення.
2. Машинне слово та машинна арифметика.
3. Одиниці виміру цифрових даних: кілобайти, мегабайти, гігабайти, терабайти.

Двійкова та інші системи числення. Одним із положень архітектури фон Неймана є використання двійкової системи обчислення. Тобто числа та вся інформація в ЕОМ надаються за допомогою нулів та одиниць. Усі операції над даними, зокрема арифметичні — додавання, віднімання, множення, поділ тощо — також виконуються в двійковому обчисленні. Вирішальним фактором при виборі двійкової системи стало те, що електронна промисловість освоїла ефективне масове виробництво двійкових цифрових схем.

Двійкове уявлення числа є послідовністю, що складається з одиниць і нулів, тобто двійкових цифр.

Кожна двійкова цифра зберігається в одному біті. Таким чином, біт є найпростішою інформаційною одиницею у сучасних ЕОМ.

Розглянемо, як видаються числа у різних системах обчислення. Спочатку розкладемо число 1929 за десятковою основою:

$$1929_{10} = 1 * 10^3 + 9 * 10^2 + 2 * 10^1 + 9 * 10^0$$

Тепер розкладемо це число за двійковою основою:

$$1929_{10} = 1 * 2^{10} + 1 * 2^9 + 1 * 2^8 + 1 * 2^7 + 0 * 2^6 + 0 * 2^5 + 0 * 2^4 + 1 * 2^3 + 0 * 2^2 + 0 * 2^1 + 1 * 2^0 = 11110001001_2$$

У ряді випадків використовується шістнадцяткове подання чисел, тобто розкладання з основою 16. Шістнадцяткова система обчислення виявляється затребуваною, наприклад, при візуалізації двійкових даних (зокрема, бінарного коду коду програм) для сприйняття людиною.

Хоча, звісно, код програми — не найзрозуміліше, що можна собі уявити, його шістнадцяткове уявлення більш читабельне, ніж двійкове. У шістнадцятковій системі числення використовуються десяткові цифри від 0 до 9 і кілька літер для позначення чисел в діапазоні від 10 до 15: A, B, C, D, E, F. Так, число 1929 у шістнадцятковому поданні виглядає так:

$$1929_{10} = 7 * 16^2 + 8 * 16^1 + 9 * 16^0 = 789_{16}$$

У загальному випадку подання натурального числа D в k-їчній системі обчислення виглядає так: де p - це кількість розрядів (цифр) числа, а $d_0, \dots, d_{p-1}$ - його k-ічні цифри, $\forall i\ d_i \in \{0, \dots, k-1\}$. Таке число зображується за допомогою десяткових цифр, а також, можливо, із застосуванням додаткових цифр, якщо основа системи обчислення більша за 10.

Таким чином, у шістнадцятковій системі обчислення ми маємо 16 цифр, у десятковій — 10, а в двійковій — 2. За допомогою цих цифр у кожній системі обчислення складаються числа.

Неважко скласти алгоритм, який розкладає довільне натуральне число в k-їчному обчисленні, а також довести єдиність такого уявлення.

Машинне слово. Двійкове уявлення числа 1929 року складається з 11 цифр, отже, для його зберігання потрібно 11 біт. Можна сказати, що біт є атомарним осередком у пам'яті ЕОМ. Але адресувати кожен біт недоцільно: ЕОМ працює з рядками, числами різного виду, а також масивами, змінюючи відразу багато біт. Тому доцільно адресувати не біти, а групи бітів. У сучасних комп'ютерах атомарним безліччю біт є байт - набір із 8 біт. Багато команд процесора вміють працювати з байтами. Відповідно, дані вирівнюються до межі байта. Так, число 1929 року в двійковому вигляді займає 2 байти і виглядає так: 0000011110001001. П'ять зайвих біт заповнюється нулями.

У сучасних ЕОМ одиницею адресації є байти, оскільки останні дозволяють оперувати занадто малими числами, а машинні слова.

Машинне слово — це атомарна кількість інформації, з якою може оперувати ця ЕОМ. Кожна ЕОМ має фіксований розмір машинного слова, наприклад, у сучасних Intel-архітектурах розмір машинного слова становить 32 або 64 біти, тобто чотири або вісім байт відповідно.

Розмір машинного слова задає такі важливі характеристики ЕОМ:

• кількість біт, які процесор може обробити за один такт, що в числі інших причин також визначає і розмір регістрів процесора;

• кількість біт у шині даних;

• кількість біт, яку процесор може за одну операцію прочитати з оперативної пам'яті;

• максимальний обсяг оперативної пам'яті, яка може бути неадресована безпосередньо процесором.

У першого процесора Intel 4004 розмір машинного слова становив 4 біти. У ЕОМ наступних поколінь розмір машинного слова становив 6, 18, 20, 36 чи 48 біт. Більшість сучасних комп'ютерів (Intel x86 та інших.), які ми згадуємо у цьому курсі, розмір машинного слова, як ми згадували вище, становить 32 чи 64 біта. Згодом розмір машинного слова неухильно збільшується, що закономірно, оскільки це дозволяє процесору за одну елементарну команду обробляти більший обсяг даних. У той самий час збільшення машинного слова потребує більш розвинених технологій апаратної реалізації.

Машинна арифметика. Мова йде про подання двійкових цілих чисел для ефективної реалізації арифметичних операцій. Висловлюючись точніше, ми розповімо, як видаються негативні цілі числа та як реалізується робота з ними.

Насамперед зауважимо: математики вважають, що низка цілих чисел нескінченна в обидві сторони, тобто не існує найменшого і найбільшого цілого числа. У машинній арифметиці нескінченність неприпустима і все звичайно, тому існує мінімальне та максимальне ціле число. Причина цього полягає в тому, що ціле число розміщується в машинному слові і, відповідно,

максимальне та мінімальне значення цілого числа обмежено розміром машинного слова.

Припустимо, що розмір машинного слова дорівнює восьми. Тоді ми маємо наступне уявлення восьмибітових двійкових чисел:

$$1_{10} = 0000\ 0001_2$$
$$10_{10} = 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0 = 0000\ 1010_2$$
$$21_{10} = 1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 0001\ 0101_2$$

Слід зазначити, що старші розряди, що залишилися незадіяними, заповнюються нулями і виходить, що будь-яке ціле число має розмір вісім біт. Отже, ми можемо використовувати один байт для зберігання цілих чисел. Але очевидно, що таким чином ми можемо оперувати цілими числами, що не перевищують найбільшого числа, яке можна уявити у восьми бітах. Ось це число:

$$1111\ 1111_2 = 1 * 2^8 + 1 * 2^7 + 1 * 2^6 + 1 * 2^5 + 1 * 2^4 + 1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0 = 511_{10}$$

Однак ми поки що говорили про невід'ємні цілі числа. Для представлення негативних цілих чисел резервується старший біт — він робиться знаковим: якщо цей біт дорівнює 0, то маємо невід'ємне число, і якщо він дорівнює 1, то маємо негативне ціле число. Нижче представлені восьмибітові від’ємні числа.

$$-1_{10} = 1000\ 0001_2$$
$$-10_{10} = 1000\ 1010_2$$
$$-21_{10} = 1001\ 0101_2$$

Зрозуміло, що, займаючи старший розряд для позначення знака числа,

$$0111\ 1111_2 = 1 * 2^7 + 1 * 2^6 + 1 * 2^5 + 1 * 2^4 + 1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0 = 255_{10}$$

ми зменшуємо значення максимально допустимого 8-бітового цілого числа — тепер воно становить

Відповідно мінімально допустиме восьмибітове ціле число дорівнює –255. Зазначимо, що представлений спосіб подання цілих чисел називається *прямим уявленням*. Його недоліком є необхідність обробляти знаковий розряд спеціальним чином, відмінним від того, як обробляються інші розряди восьмибітового цілого числа. Щоб вирішити цю проблему, введемо поняття

додаткового уявлення цілого числа. Додаткове уявлення невід'ємного числа (тобто числа, старший розряд якого дорівнює 0) тотожно прямому уявленню. Якщо ми маємо двійкове від'ємне число, то спочатку інвертуємо всі двійкові розряди цього числа (тобто замість 1 записуємо 0, замість 0 записуємо 1). Знаковий розряд ми залишаємо незмінним. Після цього ми додаємо до модуля даного числа одиницю. Якщо отримуємо переповнення у передостанньому старшому розряді, то одиницю перенесення ми відкидаємо.

$$-1_{10} = 1000\ 0001_{2\ пр} = 1111\ 1111_{2\ доп}$$
$$-10_{10} = 1000\ 1010_{2\ пр} = 1111\ 0110_{2\ доп}$$
$$-21_{10} = 1001\ 0101_{2\ пр} = 1110\ 1011_{2\ доп}$$

Тепер ми отримуємо таку перевагу при виконанні операції віднімання порівняно з використанням прямого уявлення: $A_1 - A_2 = A_1 + (-A_2)$. Тобто ми замінюємо операцію віднімання додаванням з негативним відніманням (зі шкільного курсу математики ми можемо пам'ятати, що перший аргумент операції віднімання називається зменшуваним, а другий — віднімається). Нижче наведено приклад.

$$5_{10} - 10_{10} = 0000\ 101_{2\ пр} - 000\ 1010_{2\ пр} = 0000\ 0101_{2\ доп} + 1111\ 0110_{2\ доп} =$$
$$= 1111\ 1011_{2\ доп} = 1000\ 0101_{2\ пр} = -5_{10}$$

Одиниці вимірювання цифрових даних. $2^{10} = 1024$ біт прийнято називати кілобітом (Kbit), оскільки 1024 дуже близько до 1000 і тут спрацьовує аналогія з іншими одиницями виміру — з кілометром, в якому 1000 метрів, кілограмом, в якому 1000 грам, і т.д. , точніше, кілобіти за секунду (Kbit/s), використовуються при вказівці швидкості передачі інформації — наприклад, для факсимільних апаратів та модемів. *Для позначення числа* $2^{10} = 1024$ *у 1998 р. Міжнародною електротехнічною комісією було стандартизовано бінарну приставку «кібі». Аналогічно, числу 2^20 відповідає "мебі-", а не "мега-", 2^30 - "гібі-", а не "гіга-" і т.д. Але десяткові приставки стосовно відповідних ступенів 2 встигли «прижитися» раніше появи двійкових, тому саме вони повсюдно використовуються досі.* $2^{10} = 1024$ байт прийнято

називати кілобайтом (Kb). Нещодавно в кілобайтах вимірювали різні види пам'яті ЕОМ: оперативну пам'ять, постійне запам'ятовуючий пристрій, відеопам'ять і т. д. Зараз для цих цілей застосовуються більші одиниці.

$2^{20} = 1\,048\,576 \cong 10^6$ байт дорівнює приблизно одному мільйону байт і становить один мегабайт (Mb).

$2^{30} = 1\,073\,741\,824 \cong 10^9$ байт дорівнює приблизно одному мільярду байт і становить один гігабайт (Gb). Декілька гігабайт складають обсяг оперативної пам'яті сучасного типового персонального комп'ютера, кілька десятків гігабайт — обсяг сучасного флеш-накопичувача (флешки). Такі обсяги з'явилися в побуті у зв'язку з активним використанням відео та появою високошвидкісного Інтернету. Наведемо наступний цікавий факт про гігабайти та флеш-накопичувачі. Файлова система FAT-32, яка використовується сьогодні на багатьох невеликих флеш-накопичувачах, не підтримує роботу з файлами розміром більше 4 гігабайт. Відповідно, виникають проблеми при копіюванні на такі флеш-накопичувачі файлів з відео у високій якості (часто розмір таких файлів перевищує 4 гігабайти). За необхідності роботи з великими файлами слід використовувати інші файлові системи (наприклад, NTFS або exFAT).

$2^{40} = 1\,099\,511\,627\,776 \cong 10^{12}$ байт дорівнює одному трильйону байт і становить один терабайт (Tb). Типовий об'єм жорсткого диска сучасного настільного комп'ютера становить 0,5–2 терабайти.

$2^{50} = 1\,125\,899\,906\,842\,624 \cong 10^{15}$ байт - один петабайт (Pb). Неважко порахувати, що це становить 1024 терабайт. Даними, які вимірюються у цих одиницях, оперують не окремі люди, а великі організації та дата-центри. Петабайт — це справді велика одиниця: для прикладу, один петабайт складає безперервний відеозапис гарної якості тривалістю близько півтора місяця. Але в окремих випадках і такі обсяги виявляються замало. Так, датчики Великого Адронного Колайдера під час експериментів можуть генерувати до петабайта даних щомиті. На жорстких дисках подібну кількість даних зберігати вже занадто дорого, тому для цього часто використовуються менш зручні в обігу, але набагато дешевші та ємніші магнітні стрічки.

ТЕМА 1.4. БУДОВА СУЧАСНОГО НАСТІЛЬНОГО КОМП'ЮТЕРА

План

1. Принцип модульності та його використання при конструюванні технічних систем.

2. Друковані плати та інтегральні схеми.

3. Пристрій ЕОМ на прикладі настільного комп'ютера Dell OptiPlex: системна плата, процесор, оперативна пам'ять, контролери та порти, жорсткий диск тощо.

Принцип модульності. Сучасні технічні системи, наприклад автомобілі, літаки, пароплави, побутові прилади і, зрозуміло, комп'ютери складаються з безлічі окремих деталей. Але якщо, наприклад, побутова техніка зазвичай розбирається тільки для ремонту та обслуговування, то комп'ютери можуть також розбиратися і з метою модернізації (*Upgrade*). Такі пристрої, як графічний адаптер, оперативна пам'ять, жорсткий диск, процесор, можуть підключатися до інших компонентів комп'ютера за допомогою стандартних роз'ємів і фіксуються за допомогою стандартних механічних кріплень (клапан, гвинтів і т. д.) У результаті покупець може в момент придбання комп'ютера заощадити частину коштів, а потім замінювати окремі деталі, підвищуючи продуктивність та інші характеристики ЕОМ.

Оскільки різні деталі комп'ютера сполучаються один з одним стандартними способами, їх встановлення та заміна зазвичай не потребують спеціальних знань і доступні розвиненим користувачам. Наявність достатнього досвіду дозволяє взагалі зібрати комп'ютер з цих компонент самостійно.

Щодо готових комп'ютерів можливість модернізації може обмежуватися умовами гарантії виробника. Наприклад, не втрачаючи гарантії, комп'ютери Apple можна модернізувати лише в авторизованих сервісних центрах. Для комп'ютерів, сумісних з IBM PC (більшість сучасних персональних

комп'ютерів), умови гарантії зазвичай м'якші, проте багато виробників також не схвалюють самостійну заміну деталей.

Історично можливість самостійної модернізації стала одним із суттєвих факторів популярності IBM PC на початку 1980-х років. Досить швидко цією можливістю стали користуватися не тільки користувачі та компанія IBM, але й сторонні виробники обладнання, які стали самостійно випускати компоненти для IBM PC, сумісні з оригінальними, а потім і комп'ютери, сумісні з IBM PC.

Отже, у сучасному індустріальному виробництві обчислювальної техніки різні модулі систем виробляються різними компаніями-виробниками. Наприклад, компанія Intel виробляє лише процесори, а вже інші компанії випускають підсумкові ЕОМ з використанням цих процесорів. Якщо заглянути глибше, то деталі, з яких складаються мікросхеми, у тому числі й процесори, також можуть проводитись різними спеціалізованими компаніями, а виробники мікросхем уже закуповують і використовують ці готові деталі. На сьогоднішній день принцип модульності глибоко проник в індустрію, і він важливий не так тому, що користувач може сам виконати модернізацію свого комп'ютера, а також тому, що в розробці цільового виробу можуть брати участь різні компанії-виробники. Крім того, важливо, що ті самі деталі можуть використовуватися в різних цільових системах. Все це суттєво збільшує продуктивність індустрії в цілому, оскільки виробникам немає потреби робити всі деталі для своїх систем самостійно.

Інтегральні схеми та друковані плати. Багато компонентів ЕОМ, що реалізують складні логічні функції — процесор, пристрої, що запам'ятовують, різні контролери і т. д. — виробляються у вигляді інтегральних схем.

Інтегральна схема (мікросхема, чіп, Chip) — це електронна схема, виготовлена на основі напівпровідникових технологій та розміщена на одній кремнієвій пластині.

Будучи єдиним цілим, інтегральна схема не допускає обслуговування та ремонту: при виході з ладу вона може бути замінена лише цілком. Але саме

«інтегральність» дозволила налагодити ефективний індустріальний серійний випуск недорогих інтегральних схем. Більше того, вони також мають невеликі розміри: пригадаємо, що великий розмір перших ЕОМ суттєво ускладнював їх обслуговування та радикально скорочував їх кількість користувачів. На рис. 2 наведено приклад інтегральної схеми.

Рисунок 2 - Інтегральна схема - мікропроцесор Intel Core i5

Ряд компонент сучасних ЕОМ є єдиним цілим з точки зору розв'язуваних функцій і зв'язків між собою і складаються з безлічі мікросхем, а також окремих електронних елементів, таких як транзистори, конденсатори і т. д. Для зручності інтеграції всі ці елементи поміщають на окремі друковані плати.

Друкована плата дозволяє інтегрувати набір мікросхем та одиничних електронних елементів шляхом розміщення на пластині з діелектрика, на поверхні якої розташовані провідники, що дозволяють з'єднувати інтегровані елементи один з одним. У ролі діелектрика у друкованих платах зазвичай виступають різні види склопластику. Термін «друкована» пов'язаний з технологією виготовлення плат, про яку йтиметься нижче.

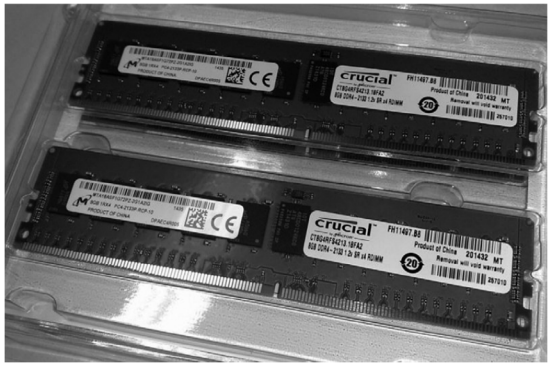

Рисунок 3 - Дві друковані плати з оперативною пам'яттю об'ємом по 8 Гб

Важливо розуміти, що друкована плата виконує не лише сполучну роль у плані передачі сигналів по провідникам, але й є важливим конструктивним елементом. Саме плати становлять значну частину сучасних електронних пристроїв - комп'ютерів, роутерів, модемів, телевізорів та смартфонів (у смартфоні плата зазвичай одна). І саме вони забезпечують модульність комп'ютерів: більшість компонентів комп'ютера, що допускають заміну користувачем, таких як оперативна пам'ять (рис. 3) або відеокарта, випускаються та продаються у вигляді друкованих плат. Більше того, друковані плати можуть вставлятися одна в одну, наприклад, оперативну пам'ять вставляють в материнську плату комп'ютера.

Пристрій настільного комп'ютера Dell OptiPlex. Далі, як приклад реальної ЕОМ, ми розглянемо типовий персональний комп'ютер Dell OptiPlex 7060, призначений для виконання офісних завдань. Готовий для роботи комп'ютер (робоче місце) включає системний блок, а також ряд периферійних пристроїв, наприклад, монітор, клавіатуру і мишу. Тут ми розглянемо лише системний блок, що є основним елементом персонального комп'ютера.

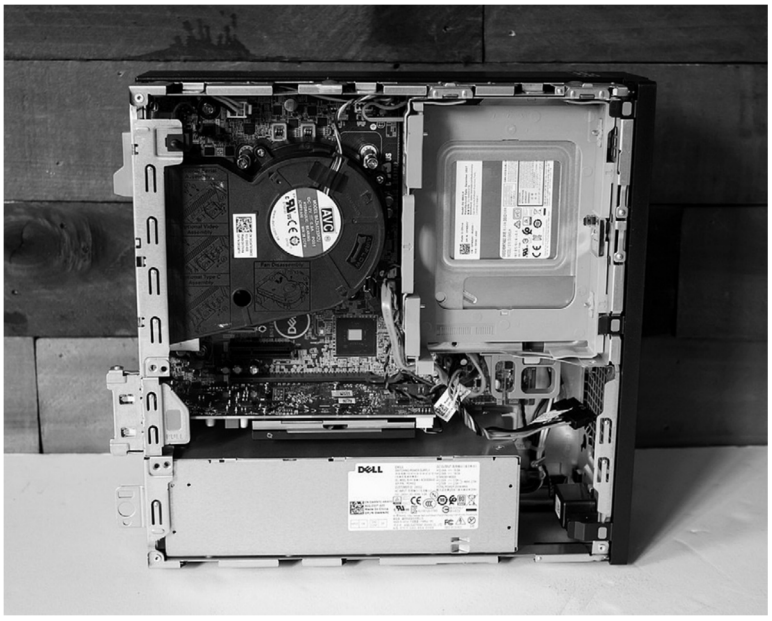

Рисунок 4 - Системний блок Dell OptiPlex 7060 із знятою кришкою

На рис. 4 представлена фотографія системного блоку комп'ютера Dell OptiPlex 7060 із кришкою. Видно частина материнської плати, вентилятор

(кулер) для охолодження материнської плати, відсік для жорстких дисків з одним диском, блок живлення, а також додатковий графічний адаптер.

Системний блок Dell OptiPlex 7060 складається з перерахованих нижче елементів.

• Материнська (системна) плата (System Board, Mother Board).

• Центральний процесор (Processor). • Оперативна пам'ять (Random Access Memory).

• Постійний пристрій пам'яті (Read-Only Memory, ROM).

• Чіпсет (Chipset).

• Графічний адаптер (Graphics Adapter).

• Роз'єми розширення (Extension Slots).

• Жорсткий диск та (HDD, Hard Disk Drive).

• SATA-контролери (Serial ATA Controllers).

• Вбудований звуковий адаптер (Sound Adapter).

• Телекомунікаційні пристрої (Network Adapters).

• Контролери зовнішніх портів (External Device Port Controllers).

• Пам'ять для налаштування комп'ютера.

• Годинник та елемент живлення (батарейка).

• Оптичний привід (Optical Disk Drive).

• Система електроживлення (Power Supply System).

• Вентилятори (кулери, Fans, Coolers).

Давайте розглянемо ці елементи докладно.

Материнська плата — це друкована плата, яка є центральним пристроєм системного блоку та всього комп'ютера, з'єднуючи та координуючи всі інші пристрої. Останні підключаються до материнської плати за допомогою паяння або шляхом встановлення спеціальних роз'ємів.

Центральний процесор є інтегральною схемою, яка здійснює виконання програм, що працюють на даному комп'ютері. Залежно від комплектації на материнську плату цієї моделі можуть бути встановлені процесори Intel Core i3, i5 або i7. Це сучасні процесори Intel різних цінових

категорій, що мають різну кількість ядер та обсяг внутрішньої кеш-пам'яті. Ядро (Core) - ця окрема частина процесора, що має здатність виконувати команди програми; кілька ядер дозволяють виконувати одночасно кілька програм. Зазначимо, що чим вищий номер маркування процесора, тим потужнішим є процесор, і, тим більше у нього можливостей і тим вища його ціна.

Оперативна пам'ять є набором друкованих плат для зберігання інформації, безпосередньо необхідної під час роботи процесора: на цих платах зберігаються програми та їх дані, що виконуються в даний момент (рис. 11). Ці плати вставляють у спеціальні роз'єми на материнській платі. Змінюючи або підключаючи нові плати з оперативною пам'яттю, можна збільшувати її загальний обсяг комп'ютера. Це буває важливо, оскільки обсяг оперативної пам'яті значно впливає на швидкодію комп'ютера. Комп'ютер Dell OptiPlex підтримує встановлення від 1 до 4 таких плат з оперативною пам'яттю загальною ємністю від 4 до 64 Gb. Для офісних завдань типовий обсяг оперативної пам'яті становить від 8 до 16 Гб.

Крім процесора, з оперативною пам'яттю можуть також взаємодіяти і вузли комп'ютера - звукові адаптери, різні контролери локальної мережі Ethernet або Wi-Fi, контролери зовнішніх портів. Наприклад, процесор готує для звукового адаптера чергову порцію цифрового аудіозапису, поміщає в оперативну пам'ять, і, поки звуковий адаптер самостійно читає з пам'яті і відтворює запис, процесор може підготувати наступну порцію і зайнятися іншою роботою. Ця можливість реалізується за допомогою технології DMA (Direct Memory Access).

Постійний запам'ятовуючий пристрій — це спеціальна друкована плата, що реалізує постійну пам'ять комп'ютера, і зміст цієї пам'яті залишається постійним і не перезаписується під час роботи комп'ютера, під час його перезавантаження та увімкнення/вимкнення. Постійний запам'ятовуючий пристрій містить BIOS і EFI — вбудовані програми, призначені для запуску операційної системи комп'ютера після увімкнення або

перезапуску. Останнім часом постійний пристрій, що запам'ятовує, зазвичай реалізовано у вигляді флеш-пам'яті і допускає програмне перезаписування з метою оновлення, тобто запису нових версій BIOS і EFI. Однак такі оновлення виконуються рідко в порівнянні з тим, як часто виконується запис даних на жорсткий диск або оперативну пам'ять.

Чіпсет є набором інтегральних схем, розміщених на материнській платі, які реалізують системну шину комп'ютера, а також деякі інші функції. Нагадаємо, що системна шина - це загальний канал передачі даних, за допомогою якого процесор може обмінюватися даними з оперативною пам'яттю та периферійними пристроями.

Як уже говорилося в минулих лекціях, в архітектурі фон Неймана системна шина є вузьким місцем: через неї передаються і дані, і машинний код, а також виконується управління та обмін даними з периферійними пристроями. Для підвищення продуктивності система шина еволюціонувала від «набору проводів» до багаторівневого пристрою. Нині вона влаштована значно інакше й складніше, ніж у перших комп'ютерах. Як розповідалося в попередніх лекціях, у комп'ютерах перших поколінь вона ділилася на шину даних, адресну та керуючу шини. У більшості сучасних комп'ютерів системна шина складається з швидкої та повільної шини.

• Швидка системна шина пов'язує процесор, оперативну пам'ять та вбудований у центральний процесор графічний адаптер; у випадку з Intel Core i3, i5 та i7 ця шина, як правило, реалізується всередині кремнієвої пластини центрального процесора (для Intel-процесорів використовується технологія Sandy Bridge). До швидкої шини підключено комутатор, через який підключені до швидкої шини пристрою зв'язуються з повільною шиною. У процесорів Intel Core i3, i5 та i7 цей комутатор також розташований на кристалі.

• Повільна системна шина (поширений варіант реалізації – технологія PCI Express) розташована на материнській платі та забезпечує взаємодію процесора з іншими пристроями, які не потребують передачі даних з високою

швидкістю. Повільна шина PCI Express не поділяється фізично на шини даних, адреси та керування. Натомість вона передає по загальному каналу окремі повідомлення (пакети), що містять у собі необхідні дані, адреси та сигнали керування. Крім того, вона не з'єднує пристрої один з одним, а підключає їх до комутатора. Таке рішення зручне для додавання до комп'ютера різноманітних пристроїв – контролерів жорсткого диска, USB-контролерів, контролера клавіатури, адаптерів Wi-Fi та Ethernet тощо.

Поділ шини на швидку та повільну доцільно, оскільки швидкість взаємодії з пристроєм важливо поєднувати зі швидкістю його роботи, а остання у різних пристроїв виявляється різною. При цьому швидка шина, процесор та оперативна пам'ять розміщуються на платі за можливістю компактно, щоб скоротити відстані між ними.

Графічний адаптер є спеціальним пристроєм, призначеним для обробки зображення та формування відеосигналу під час виведення зображення на монітор комп'ютера. Сформований відеосигнал передається з графічного адаптера на монітор через стандартні роз'єми — HDMI та VGA, які можна бачити на звороті системного блоку. Зокрема, в ці роз'єми можна підключати замість монітора інші пристрої, наприклад телевізор або проектор.

Графічний адаптер може бути інтегрований у центральний процесор чи чіпсет. Також він може бути виконаний у вигляді окремого (discrete) пристрою та бути вставленим у материнську плату через спеціальний слот розширення. Такий адаптер може вставлятися в комп'ютер, якщо продуктивність вбудованого адаптера недостатня - наприклад, у разі потреби підтримки ресурсомістких відеоігор. Саме так ситуація з комп'ютером Dell OptiPlex 7060, зображеним на рис. 12: окремий графічний адаптер вставлений у роз'єм синього кольору на материнській платі, незважаючи на те, що в комп'ютерах сімейства Intel x86 Intel Core i3, i5, i7 та вище графічний адаптер вже інтегрований у центральний процесор.

Сучасний графічний адаптер має власний процесор, який може бути використаний не лише за прямим призначенням, тобто для обробки відео, але

й для довільних обчислень. Така можливість забезпечується GPGPU-технологіями (General Purpose Graphical Processing Unit), що дозволяють програмістам використовувати графічний процесор комп'ютера для обчислень, які у звичайній ситуації виконує центральний процесор. Такі технології виявляються корисними, коли виникають проблеми з продуктивністю, наприклад, при моделюванні фізичних процесів, у криптографічних задачах, при навчанні та використанні нейронних мереж, при вирішенні графових завдань методами лінійної алгебри тощо.

Роз'єми розширення дозволяють підключати до материнської плати додаткові пристрої, якими цей комп'ютер не комплектується, наприклад, пристрій відеозахоплення або адаптер для оптичної локальної мережі.

Графічний адаптер і контролери різних пристроїв можуть бути вбудовані в материнську плату, але не мати достатньої продуктивності для вирішення завдань користувача даного комп'ютера, а можуть бути зовсім відсутніми. У таких випадках потужніші аналоги можна встановити в роз'єми розширення. Наприклад, вбудовані графічні адаптери добре підходять для вирішення офісних завдань, але бувають недостатньо продуктивними для комп'ютерних ігор, і тоді в системний блок, в слот розширення, вставляється окремий, винесений графічний адаптер.

Жорсткий диск використовується для довгострокового зберігання даних та програм. На відміну від оперативної пам'яті, швидкодія жорсткого диска суттєво нижча, а обсяг даних, що зберігаються, — значно більший. Програми (програми), встановлені на комп'ютері, зберігаються саме на жорсткому диску, а перед виконанням завантажуються в оперативну пам'ять. Залежно від комплектації, дана модель ЕОМ допускає жорсткий диск об'ємом від 500 Gb до 2 Tb. Жорсткий диск часто називають вінчестером. Вінчестер — це сленговий термін, який з'явився в 1970-х роках у компанії IBM для позначення жорсткого диска IBM 3340. Максимальна конфігурація цього виробу включала два такі жорсткі диски об'ємом по 30 Mb. Два числа 30 викликали у розробників вироби асоціацію з відомою мисливською

гвинтівкою Winchester, патрони до якої так і називалися — 30–30. Перше число означало калібр у сотих частках дюйма, друге — вага порохового заряду у спеціальних одиницях (гранах). Надалі цей термін став використовуватися назви всіх жорстких дисків.

Замість жорсткого диска на комп'ютері Dell OptiPlex 7060 може бути встановлений твердотільний накопичувач (SSD, Solid State Drive) від 512 Gb до 1 Tb. Твердотільний накопичувач зберігає дані у вигляді флеш-пам'яті великого об'єму. Він коштує дорожче механічного жорсткого диска, але має велику швидкодію.

SATA-контролери керують енергонезалежними накопичувачами даних – жорсткими дисками, твердотілими накопичувачами, оптичними приводами. Ці накопичувачі використовуються для зберігання програм і даних як системних, так і користувальницьких, причому, на відміну від оперативної пам'яті, вони не «забувають» інформацію, коли комп'ютер вимкнено. SATA-контролери підтримують механізм DMA (Direct Memory Access), який дозволяє їм зв'язуватися з оперативною пам'яттю без участі центрального процесора: завдяки цьому механізму поки контролер завантажує дані з накопичувача в пам'ять або зберігає дані з пам'яті на накопичувачі, процесор може виконувати іншу роботу.

Вбудований звуковий адаптер є пристроєм, який реалізує роботу комп'ютера зі звуком: підтримує відтворення (програвання) звукозапису в динаміках або навушниках, а також дозволяє виконувати запис аудіоматеріалу з мікрофона. Сучасні ноутбуки комплектуються вбудованими динаміками та мікрофоном.

Мережні адаптери дозволяють підключати комп'ютер до комп'ютерних мереж. Dell OptiPlex 7060 має адаптери для підключення до найпоширеніших видів мереж — дротової мережі Ethernet і бездротової мережі Wi-Fi.

Контролери зовнішніх портів є пристроями, що керують зовнішніми портами комп'ютера, які зазвичай виведені зі зворотного боку системного блоку — один контролер на один або кілька портів. Dell OptiPlex має такі

порти: 10–12 USB-портів (залежно від комплектації), два PS/2-порти, призначені для підключення миші та клавіатури, а також кілька інших видів портів для сумісності зі старими пристроями*.

**Йдеться про послідовні та паралельні порти, які призначаються для підключення периферійних пристроїв, в основному вироблених 20 років тому і раніше. Послідовний порт використовувався для підключення до комп'ютера пристроїв, що не вимагають передачі великих об'ємів даних та високої швидкості взаємодії (миша, факс-модем), а паралельний порт використовувався для підключення високошвидкісних пристроїв (сканер, принтер). Аналогічні сучасні пристрої зазвичай підключаються до комп'ютера за допомогою USB-портів, а багато сучасних ПК послідовних і паралельних портів немає зовсім.*

До USB-портів можуть підключатися будь-які сумісні з ними зовнішні пристрої, наприклад, принтер або сканер. Значення терміна «порт» у обчислювальній техніці варіюється залежно від контексту. Портом називають і роз'єм на системному блоці комп'ютера, і спосіб сполучення контролера зовнішнього пристрою з системною шиною, і точки з'єднання довільних пристроїв, розташованих на певній платі (не обов'язково материнській), з різними каналами, і, нарешті, довільну «зовнішню» розетку» різного електронного обладнання.

Пам'ять для налаштувань комп'ютера є особливим видом пам'яті, призначений для зберігання основних налаштувань ЕОМ, що використовуються при її запуску. Йдеться про стан адаптера Wi-Fi, варіанти завантаження операційної системи і т. д. Ці настройки потрібно зберігати після вимкнення комп'ютера з електричної мережі. Для їх зберігання не використовується жорсткий диск, оскільки частина цих налаштувань використовується при його ініціалізації. Більш того, в деяких комплектаціях комп'ютер, що розглядається, може завантажуватися по локальній мережі і зберігати свої дані там же, відповідно, жорсткого диска у нього може не бути зовсім. У комп'ютері Dell OptiPlex 7060 пам'ять для налаштувань реалізована

за допомогою технології CMOS (Complementary Metal-Oxide-Semiconductor), яка потребує постійної електроживлення, що забезпечується батарейкою. Часто цю пам'ять так і називають CMOS, хоча для її реалізації використовуються й інші технології - наприклад, флеш-пам'ять.

Годинник та джерело живлення (батарейка). Годинник (і календар) потрібний комп'ютеру для правильної обробки даних (наприклад, у кожного файлу записано час створення та зміни), для роботи офісних програм (наприклад, нагадування про зустрічі), а також для системних утиліт (наприклад, періодичного запуску антивірусу) та ін. Для годинника необхідне автономне джерело живлення (батарейка), щоб не встановлювати його щоразу після запуску комп'ютера.

Оптичний привід є дисководом для роботи з оптичними дисками, тобто дисками CD (зараз вони використовуються все рідше) та DVD. Спочатку (а це був кінець 1970-х) технологія оптичних дисків призначалася для зберігання звукозаписів. Пізніше ця технологія була вдосконалена і стала застосовуватися для зберігання відео, а потім і довільних даних.

Оптичний диск складається з кількох шарів. Зовнішні шари, виготовлені з пластику, забезпечують його механічну міцність, а внутрішні, що виготовляються з різних металів або сплавів, призначені для зберігання даних. Читання та запис на оптичні диски виконуються за допомогою лазерного променя. Ділянки диска, залежно від того, чи записані на них нулі або одиниці, по-різному відображають світло лазера, що й для читання даних: лазер підсвічує потрібний біт, а фотодатчики за відображеним сигналом визначають його значення. Для запису існують різні технології, але всі вони ґрунтуються на принципі зміни оптичних властивостей внутрішнього шару диска за допомогою впливу потужнішим, ніж при читанні, лазерним променем.

Система електроживлення перетворює напругу електричної мережі в набір напруги живлення для ПК, а також має набір кабелів живлення і роз'ємів, що підключаються до різних пристроїв.

Вентилятори призначені для тепла від вузлів комп'ютера. Сучасний комп'ютер виділяє значну кількість тепла через такі причини.

• При проходженні струму через резистори та напівпровідникові елементи вони виділяють тепло.

• ККД механічних частин жорсткого диска, а також системи електроживлення менше 1. Жорсткий диск не всю електроенергію, що надходить йому, переводить в механічну, і підсистема електроживлення комп'ютера не всю споживану електроенергію перетворює за призначенням. «Залишок» енергії ці підсистеми перетворюють на тепло, внаслідок чого також суттєво нагріваються під час роботи.

При цьому всі компоненти комп'ютера є досить вимогливими до температурного режиму: при виході за межі робочих температур змінюються характеристики напівпровідників, що призводить до відмови електроніки, а при сильному перегріві може розплавитися олов'яний припій, який використовується для з'єднання мікросхем із системною платою. Щоб підтримувати робочу температуру, Dell OptiPlex 7060 має два кулери: один є загальним на весь системний блок, другий забезпечує охолодження центрального процесора.

ТЕМА 1.5. ІНТЕГРАЛЬНІ СХЕМИ ТА ДРУКОВАНІ ПЛАТИ

План

1. Напівпровідники та напівпровідникові елементи.
2. Виробництво інтегральних схем: напівпровідникові властивості кремнію, виготовлення кремнієвих пластин, фотолітографія.
3. Виробництво друкованих плат: фотолітографія, склеювання шарів.

Напівпровідники та напівпровідникові елементи. Нагадаємо, що перші ЕОМ конструювалися з використанням радіоламп та електромагнітних реле, але після винаходу транзистора наприкінці 1940-х років відбулася зміна поколінь, і починаючи з другого покоління ЕОМ створювалися вже на базі напівпровідникових елементів. Останні забезпечили суттєвий прогрес обчислювальної техніки, оскільки дозволили радикально зменшити розміри комп'ютерів, а також зменшити витрати електроенергії та підвищити надійність ЕОМ. Крім того, напівпровідникові комп'ютери виявилися істотно дешевшими за комп'ютери попередніх поколінь. Напівпровідникова елементна база використовується і зараз.

Звернемося до курсу шкільної фізики та згадаємо, що таке напівпровідники: *Напівпровідник — це речовина, електрична провідність якої суттєво залежить від складу домішок, доданих до неї; також його провідність може суттєво змінюватись під впливом температури, електричних та магнітних полів.*

Напівпровідники з різними характеристиками виготовляються шляхом додавання домішок кристалічні речовини (зазвичай кремній). Поєднуючи різні напівпровідники, виготовляють транзистори, діоди, тиристори і т. д. Напівпровідникові елементи в електронних схемах підсилюють та перетворюють електричні сигнали. Зокрема, здатність транзисторів змінювати свою провідність залежно від характеристик електричного керуючого сигналу дозволяє створювати на їх основі логічні вентилі.

Електрична провідність речовини забезпечується електронами на зовнішній оболонці його атомів. Ця ж оболонка переважно визначає і *валентність* речовини — здатність його атомів зв'язуватися з атомами інших речовин і, для кристалічних речовин, здатність формувати кристалічну решітку. Тому електрони, що знаходяться на зовнішній електронній оболонці, називаються *валентними*. Метали легко обмінюються валентними електронами, що дозволяє їм бути *провідниками*, тобто мати високу *електричну провідність*. Чисті напівпровідники, які не містять домішок, мають набагато менш «рухливі» валентні електрони і, як наслідок, низьку електричну провідність.

Зупинимося докладніше на технології виготовлення напівпровідникових елементів. Основним хімічним елементом, який використовується при їх виробництві, є кремній – природний напівпровідник. Кремній є не єдиним природним напівпровідником, але він — один із найпоширеніших хімічних елементів на Землі і становить понад 27% земної кори. Тому його легко видобувати, а також нескладно переробляти — чистий технічний кремній ефективно і недорого виготовляється з оксиду кремнію, тобто піску (рис. 13).

Як уже обговорювалося вище, власна електрична провідність кристалічного кремнію мала, оскільки всі валентні електрони атомів кремнію задіяні в кристалічній решітці, реалізуючи зв'язки з чотирма сусідніми атомами. Для зміни електричних властивостей кремнію додають *легуючі домішки*. Ці домішки дозволяють «розворушити» частину його валентних електронів, таким чином збільшити електричну провідність кремнію, а також вплинути на характер цієї провідності. Як домішки використовуються фосфор, миш'як, бор та деякі інші елементи з відповідною конфігурацією електронних оболонок. Зокрема, миш'як має п'ять валентних електронів, тому при додаванні невеликої кількості миш'яку в кремній, який чотиривалентний, один з його електронів залишається вільним. Такі вільні електрони можуть переміщатися від одного атома миш'яку до іншого, переносячи заряд, і в

результаті кремній, легований миш'яком, набуває властивостей *електронної провідності або n-провідності*. Бор, навпаки, має три валентні електрони, тому кремній з домішкою бору містить вільні місця, які можуть займати електрони прилеглих атомів. В цьому випадку по кристалу переміщаються, переносячи заряд, вже не електрони, а так звані позитивно заряджені дірки, а кремній набуває *p-провідність, або діркову провідність*. Зазначимо, що, як «відсутність електрона», дірка є не фізичним об'єктом, а зручною умоглядною абстракцією, фактично ж заряд, як і раніше, переносить електрон.

У транзисторах, діодах та інших напівпровідникових елементах *p- і n-напівпровідники* з'єднуються, і поблизу їх межі утворюється *p-n-перехід*, який може реагувати на електричне поле, відкриваючи або замикаючи рух електричного струму. На основі одного *p-n-переходу* виготовляється напівпровідниковий *діод*, що пропускає струм лише в одному напрямку - від *p-напівпровідника до n-напівпровідника*. Якщо напівпровідники чергувати в послідовності *n-p-n*, то вийде *NPN-транзистор*, який проводить струм між крайніми n-напівпровідниками (відкривається) тільки при подачі високого електричного потенціалу на середній p-напівпровідник. Його «близнюк» – *PNP-транзистор* – працює навпаки: проводить струм між крайніми p-напівпровідниками лише при подачі низького потенціалу на n-напівпровідник.

Слід зазначити, що працездатний транзистор з характеристиками, що допускають його практичне використання, виходить лише за дотримання безлічі додаткових умов — товщини та об'єму напівпровідників, концентрації легуючих домішок тощо. Дослідження властивостей напівпровідників проводилися фізиками в різних країнах. 1920-х років, але біполярний транзистор, аналогічний сучасним, був створений лише 1947 року. У 1956 році творці біполярного транзистора У. Шоклі, У. Браттейн і Д. Бардін були удостоєні Нобелівської премії. Дослідження напівпровідників та їх застосування у мікроелектроніці тривають і зараз. Один із найвідоміших сучасних вчених у цій галузі — наш співвітчизник академік Ж. І. Алфьоров (1930–2019), у 2000 році також удостоєний Нобелівської премії з фізики. На

рис. 5. зображено оксид кремнію, тобто пісок у пустелі (ліворуч), монокристал кремнію (по центру) та зріз монокристалу із заготовками для мікросхем на ньому (праворуч).

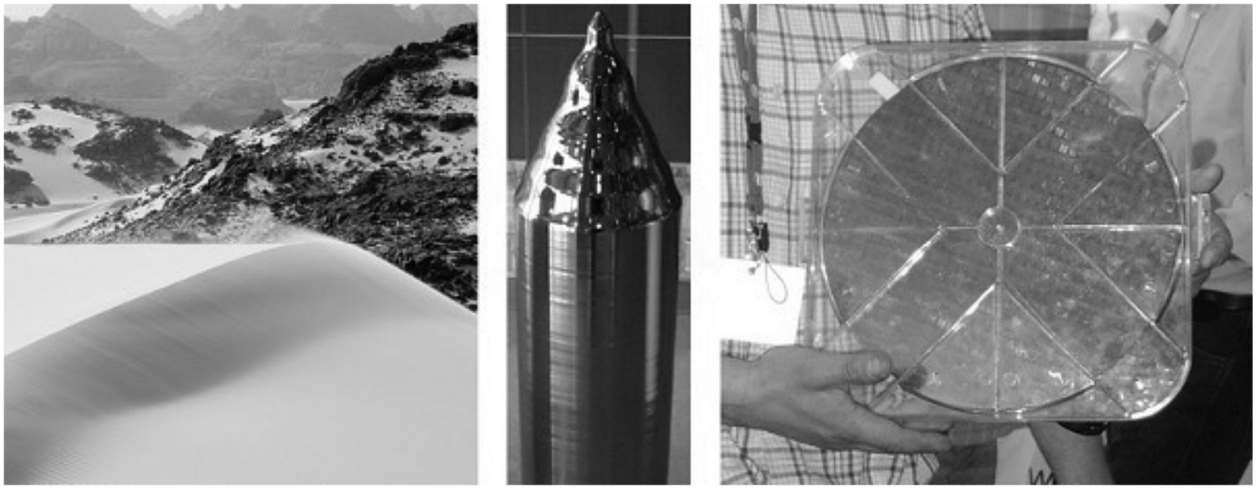

Рисунок 5 – Оксид кремнію

Крім кремнію в природі є інші природні напівпровідники, наприклад, германій. Існують транзистори та інтегральні схеми на основі германію, але доступність кремнію та наявність відпрацьованих технологій виробництва зробили його кращим базовим матеріалом.

Технологія виробництва інтегральних схем. Хоча окремі транзистори, діоди та інші елементи і сьогодні безпосередньо монтуються на друковані плати, основою функціонал сучасних комп'ютерів реалізують вже інтегральні схеми. Вони групують на кремнієвих пластинах величезну кількість транзисторів, діодів та інших елементів, створюючи спеціалізовані пристрої, наприклад, процесори. Інтегральні схеми дозволили суттєво здешевити комп'ютери, а також зменшити їх розміри та збільшити продуктивність.

Монокристал — це стан кристалічної речовини, який характеризується вкрай незначною кількістю порушень геометрії його кристалічної решітки, тобто майже всі атоми решітки розташовані в ідеальному порядку.

У природі монокристали мало зустрічаються. Але для виробництва мікроелектроніки вони важливі, оскільки розміри електронних елементів, що створюються на базі кремнію, вже співставні з відстанню між атомами в

кристалічній решітці (близько 0,5 нм). Отже, окремі дефекти в кристалі можуть призвести до появи непрацездатних напівпровідникових елементів.

Щоб зробити природний кремній придатним для використання у виготовленні друкованих плат, його очищають від домішок, досягаючи концентрації кремнію близькою до 100%. Далі виробляють (вирощують) монокристали кремнію у вигляді циліндрів. Ці циліндри ріжуть на пластини, на яких розміщують інтегральні схеми (див. рис. 13). Зараз виготовляють циліндри діаметром 300 міліметрів, але розробляються технології для створення 450 міліметрових циліндрів. Збільшення діаметра циліндрів потенційно дозволяє збільшити площу інтегральної схеми і, отже, розмістити на одній пластині більше транзисторів. Це, своєю чергою, дозволяє випускати більш високопродуктивні процесори. Але збільшення діаметра кремнієвих кристалів пов'язані з великими технологічними труднощами. Для того, щоб сформувати на кремнієвій пластині окремі електронні елементи, використовується спеціальна високоточна технологія під назвою фотолітографія.

Фотолітографія - це технологія нанесення малюнка на різні поверхні за допомогою таких дій: суцільне нанесення «фарбуючої» речовини на поверхню, нанесення захисного шару, фотохімічна стабілізація захисного шару в необхідних місцях, розчинення «фарбуючої» речовини на інших ділянках поверхні (тобто там, де його не було стабілізовано).

При виробництві інтегральних схем фотолітографія виконується в такий спосіб. На кремнієві пластини напиляються легуючі домішки, які, як і у разі виготовлення окремих напівпровідникових елементів, створюють на поверхні кремнію ділянки з *р- або n-провідністю.* Спочатку створюються всі ділянки з р-провідністю. Для цього вся поверхня кристала на невелику глибину легується р-домішкою. Потім кристал покривається фоторезистом — речовиною, яка набуває хімічної стійкості до розчинника під дією світла. Потім фоторезист «засвічується» через спеціальний оптичний трафарет, завдяки чому потрібні ділянки кремнієвої пластини набувають хімічної

стійкості. Нарешті, розчинником змивається незасвічений фоторезист та домішки на незахищених ділянках. У результаті p-домішка фіксується у верхніх шарах кремнію тільки на необхідних ділянках майбутньої інтегральної схеми. Після синтезу p-ділянок цей процес повторюється з іншим трафаретом, вже для n-домішки. У результаті різні ділянки кремнієвої пластини легуються різними домішками. Перелічені вище кроки можуть повторюватися кілька разів або навіть кілька десятків разів. Як і у випадку з окремими напівпровідниковими елементами, на межах цих ділянок утворюються p-n-переходи, на основі яких створюються мініатюрні транзистори, діоди та інші елементи. Слід зазначити, що в залежності від технології, яка використовується на конкретному виробництві, синтез p- та n-ділянок на кремнієвій пластині може виконуватися в іншому порядку. Також можливе застосування фоторезистів, які, навпаки, втрачають стійкість під впливом світла.

У сучасних процесорах використовується велика кількість напівпровідникових елементів, кількість яких обчислюється мільярдами. Зрозуміло, всі вони працюють не самі по собі, а у зв'язці один з одним. Подібно до того, як в електромеханічних приладах для з'єднання окремих елементів використовуються дроти, для з'єднання напівпровідникових елементів інтегральної схеми на кремнієву пластину наносяться провідники. Ці провідники є струмопровідними доріжками з атомів металу з високою електричною провідністю, наприклад із золота.

Нанесення на кремнієву пластину легуючих домішок та провідників проводиться у декілька шарів. Ця багатошаровість необхідна, оскільки навіть найпростіші електричні схеми далеко не завжди можна зробити плоскими. Наприклад, якщо складну схему розташувати на площині, різні її з'єднання будуть перетинатися, не утворюючи при цьому електричного з'єднання в місці перетину. Але цього можна досягти, якщо у схеми з'являється третій вимір, тобто з'явиться можливість розмістити одне з глибше, що перетинається, — і

тоді воно перестане перетинати попереднє, але загальна геометрія схеми при цьому збережеться.

На сьогоднішній день досягнуто великих успіхів у мініатюризації електронних елементів. У перші десятиліття виробництва інтегральних схем саме ця мініатюризація забезпечувала виконання закону Мура, але зараз цього вже недостатньо: надалі зменшити у 100 разів розмір транзистора, який і зараз складається з кількох десятків атомів, не вдасться. Подальше збільшення продуктивності ЕОМ (і, як і раніше, виконання закону Мура) здійснюється шляхом розпаралелювання — розміщення на кристалі блоків процесора, що працюють паралельно.

Технологія виробництва друкованих плат. Основними компонентами сучасних обчислювальних пристроїв є друковані плати. На платах розміщують інтегральні схеми (наприклад, процесори) та окремі елементи (транзистори, діоди, резистори, конденсатори). Останні зазвичай потрібні в тих випадках, коли, наприклад, елемент занадто великий за розміром для приміщення на інтегральну схему (наприклад, конденсатор з великою ємністю), або коли потрібно змінювати елемент або набір елементів залежно від різної версії плати (варіювати вміст інтегральної схеми при серійному випуску не можна). Крім цього, на платі монтуються роз'єми для з'єднання з іншими платами та пристроями, а також допоміжні механічні компоненти – кулери, радіатори. Нарешті, безпосередньо на платі є набір струмопровідних доріжок, які в потрібних поєднаннях з'єднують електричні висновки перерахованих компонентів між собою та з інтегральними схемами, що дозволяє їм обмінюватися сигналами один з одним.

Друкована плата є листом діелектрика — зазвичай використовується *склопластик*, який є жорстким і міцним матеріалом зі скловолокна, просоченого будь-яким полімером. На лист діелектрика наносяться мідні «доріжки», що з'єднують висновки розміщених на платі інтегральних схем та інших компонентів.

Коротко опишемо технологію виробництва друкованих плат.

Для виготовлення друкованих плат, подібно до виготовлення інтегральних схем, також використовується технологія, заснована на фотолітографії. Як заготовки для друкованої плати використовують лист склопластику. Його покривають шаром міді для створення струмопровідних доріжок, що з'єднують між собою різні елементи друкованої плати — інтегральні схеми, елементи, роз'єми тощо. Для того, щоб сформувати із суцільного шару міді окремі доріжки, виконуються такі дії. На мідний шар наноситься шар фоторезиста (тут звичайно застосовується інший склад, ніж при виготовленні інтегральних схем). На фоторезист виконується проекція зображення доріжок, після чого засвічені ділянки фоторезиста, під якими ховаються майбутні доріжки, стають хімічно стійкими. Потім виконується травлення: плата міститься в розчинник, який розчиняє мідь, не захищену фоторезистом. У результаті відсікається все непотрібне, тобто з усього мідного шару на платі залишаються тільки мідні доріжки, а також монтажні майданчики. Вони є спеціальними ділянками плати, до яких у певному порядку підходять «доріжки», утворюючи на цих ділянках набір електричних контактів. На контактах виконується паяння інтегральних схем, роз'ємів та інших компонентів. При паянні здійснюється електричне з'єднання висновків компонентів (так званих «ніжок») з контактами і, отже, «доріжками» на платі. Також паяння забезпечує жорстке кріплення компонентів до плати за ті самі «ніжки». Залежно від особливостей компонентів монтаж виконується різними способами. Коли потрібна висока механічна міцність або можливість проводити великий струм, застосовується наскрізний монтаж, при якому «ніжки» компоненти проходять крізь плату, а монтажний майданчик є набором контактів з отворами для ніжок. Наскрізний монтаж застосовується для кріплення на плату роз'ємів та великих потужних пристроїв, наприклад, реле або трансформаторів. Для дрібних компонентів — окремих мікросхем та елементів — висока міцність і можливість проводити великі струми зазвичай не потрібні. У цьому випадку може застосовуватися більш дешевий і компактний *поверхневий монтаж*, який полягає в простому припаюванні ніжок до контактів або безпосередньо до «доріжок».

Електрична схема друкованої плати може, як і у випадку з інтегральними схемами, бути об'ємною, тобто вимагати більше одного шару. Для цього зі зворотного боку заготівлі також наноситься мідний шар, який використовується для створення доріжок, і в результаті схема стає двоповерховою. Але для складних друкованих плат (наприклад, сучасних системних плат) двох шарів недостатньо. У цій ситуації виготовляють кілька друкованих плат, а потім склеюють їх між собою, отримуючи в результаті єдину багатошарову плату. Електричні з'єднання між доріжками у різних шарах плат виконують за допомогою свердління наскрізних отворів у платі та заповнення цих отворів струмопровідним припоєм (оловом).

Роз'єми, інтегральні схеми та інші елементи припаюються до монтажних майданчиків на платі. Для цього також використовується олов'яний припій. Температура плавлення олова складає всього 231,90 °C і легко досягається при перегріві друкованих плат під час роботи комп'ютера. Перегрівання електронних компонентів та протікання через паяне з'єднання великого струму можуть призвести до розплавлення та витікання припою, що спричиняє порушення контакту. Один із простих прийомів ремонту друкованих плат у такій ситуації — це повторне нагрівання за допомогою технічного фена, в результаті чого крапельки припою плавляться і затікають назад на свої місця.

Для запобігання перегріву інтегральних схем використовуються вентилятори (кулери). Вони кріпляться на друковану плату за допомогою гвинтів або клямок, які вставляються в спеціально просвердлені на платі отвори кріплення. Приклад готової друкованої плати показано на рис. 6.

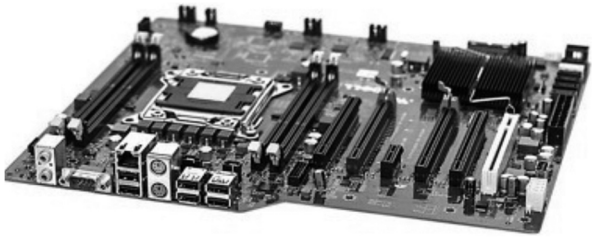

Рисунок 6 – Друкована плата (материнська плата Dell Precision T3600) з безліччю монтованих елементів та роз'ємів, у тому числі роз'єму для встановлення процесора та кріпленнями для радіатора з кулером

ТЕМА 1.6. ОСНОВИ СХЕМОТЕХНІКИ

План

1. Булеві функції.
2. Таблиці істинності булевих функцій.
3. Система базових булевих функцій.
4. Вентилі.

Булеві функції. Нехай $B=\{0,1\}$. Тоді n-вимірною булевою функцією називатимемо функцію, що діє з $B \times \ldots \times B \ldots y\ B$. Розглянемо наступні найпростіші булеві функції.

• $A \wedge B$ — «і» (кон'юнкція), видає 1, коли обидва аргументи дорівнюють 1. Інакше видає 0.

• $A \vee B$ — «або» (диз'юнкція), видає 1, коли хоча б один аргумент дорівнює 1. Інакше видає 0.

• $\neg A$ — «ні» (заперечення), видає 1, якщо аргумент дорівнює 0, і 0, якщо аргумент дорівнює 1.

• A → B – «імплікація», видає 0, коли перший аргумент дорівнює 1, а другий — 0 (тобто з істини не може слідувати брехня), а в інших випадках видає 1.

• $Xor(A,B)$ (інший варіант запису — $A \oplus B$) — «виключне або» (додавання за модулем 2), видає 1, коли рівно один аргумент дорівнює 1, в інших випадках видає 0.

Таблиці істинності булевих функцій. Булеві функції, на відміну від функції математичного аналізу, є дискретними. Більше того, кількість можливих унікальних поєднань аргументів, а також відповідних значень є невеликою і може бути перерахована явно.

Значення булевих функцій відповідно до значень їх аргументів можна подати за допомогою ***таблиць істинності****. Кожен рядок такої таблиці відповідає поєднанню аргументів та відповідному значенню функції.*

Таблиця 1 представляє таблиці істинності для низки відомих булевих функцій.

Таблиця 1 – Таблиці істинності для деяких найпростіших булевих функцій

A	B	$A \wedge B$	$A \vee B$	$\neg A$	$Xor(A,B)$
0	0	0	0	1	0
0	1	0	1	1	1
1	0	0	1	0	1
1	1	1	1	0	0

Система базових булевих функцій. Існує наступна важлива теорема. **Теорема 1.** Будь-яка функція булева може бути представлена у вигляді суперпозиції наступних булевих функцій — «і», «не» або «і», «або».

Незважаючи на те, що вистачає двох функцій із трьох, для подання довільної булевої функції зручно використовувати всі ці три функції (у цьому, до речі, полягає ідея доказу наведеної вище теореми).

Наприклад, $A \to B = \neg A \vee B, Xor(A,B) = (A \wedge \neg B) \vee (\neg A \wedge B)$.

Набір булевих функцій, за допомогою якого можна виразити решту булевих функцій, називатимемо системою ***базових булевих функцій***.

Набори {“і”, “не”} та {“або”, “не”} є системами базових булевих функцій. Слід зазначити, що існують інші аналогічні системи. Наприклад, кожна з наведених нижче функцій утворює базову систему булевих функцій (тобто кожна з цих систем складається з єдиної функції).

• $A|B = \neg(A \wedge B)$ – штрих Шефера («і-ні»).

• $A \downarrow B = \neg(A \vee B)$- стрілка Пірса ("і-або").

Неважко довести, що інших систем базових функцій, що складаються лише з однієї функції, більше немає.

Вентилі й логічні схеми. Тепер перейдемо від булевих функцій до внутрішнього пристрою комп'ютерів — до вентилів, логічних схем, функціональних блоків процесора та команд машинної мови.

Фізична реалізація найпростішої булевої функції за допомогою напівпровідникових елементів (транзисторів, резисторів, діодів тощо) називається ***логічним вентилем*** *(Gate).*

Таким чином, вентилі є елементарними логічними схемами, і складніші схеми, що реалізують різні функціональні блоки процесора, будуються на їх основі.

На рис. 15 наведено позначення основних логічних вентилів відповідно до стандарту ANSI/IEEE91-1984. ANSI (American National Standards Institute) - американський інститут, який здійснює нагляд за розробкою та використанням різних комп'ютерних стандартів. Так ось, у стандарті ANSI/IEEE91-1984 описані графічні зображення як вентилів, так і складніших логічних схем, а також деяких інших схем цифрової електроніки. Історично він базується на стандартах міністерства оборони США 1950–1960-х років — вентилі ранніх ЕОМ створювалися на базі ламп та електромеханічних реле, але позначалися на схемах аналогічним чином. Нині цей стандарт є міжнародним та використовується у різних сферах виробництва.

Тепер визначимо складніший елемент під назвою логічна схема.

Логічна схема — це функція, яка на вхід одержує деякий двійковий вектор, а на виході видає перетворений двійковий вектор.

Процесор складається з набору функціональних блоків, таких як арифметико-логічний пристрій, пристрій для читання машинного коду, набір регістрів, кеш-пам'ять і т. д. У свою чергу, ці функціональні блоки складаються з безлічі різних логічних схем.

При побудові логічних схем використовують атомарні елементи - вентилі, які є реалізацією найпростіших булевих функцій. Демонстрацію цього факту буде проведено в наступній лекції, проте вже тепер ми проведемо деяку підготовку.

Для роботи основних функцій процесора дуже важливими є логіко-арифметичні операції. З їх допомогою реалізуються найважливіші складові команди процесора: завантаження/вивантаження в оперативну пам'ять,

присвоєння, умовні переходи, цикли та ін. Крім того, різні математичні функції (наприклад, log або sin) зазвичай обчислюються сучасними процесорами за допомогою часткового підсумовування рядів Тейлора. Такі обчислення не тільки вимагають безпосередньо арифметичних операцій, але також використовують цикли, а отже, і логічні операції. У свою чергу, математичні функції використовуються для обробки мультимедіа даних — зображень, відео, звуку. Зрозуміло, використовуються вони й під час виконання ЕОМ свого «вихідного призначення» — фізичних і математичних розрахунків.

Отже, логічні операції ми маємо — це булеві функції, що реалізуються вентилями. А те, як ідеться з арифметичними операціями, буде показано в наступній лекції, де описується логічна схема для додавання (тобто суматор).

Ну і, нарешті, відносно невеликий набір вентилів може нагадувати нам систему базових булевих функцій — з тією різницею, що до такої системи додано більше функцій, ніж містить мінімальна система базових функцій, оскільки потрібно будувати реальні (не теоретичні) логічні схеми і потрібні зручні будівельні блоки, що реалізують найчастіше зустрічаються операції.

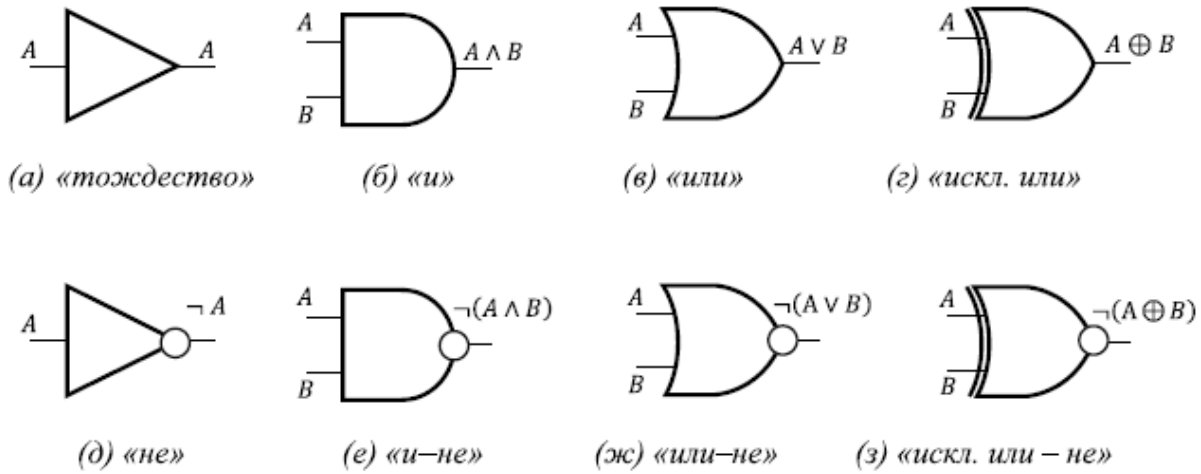

Рис. 6 - Схематичне зображення вентилів відповідно до стандарту ANSI/IEEE91-1984

ЗМІСТОВИЙ МОДУЛЬ 2
ЕЛЕМЕНТИ КОМП'ЮТЕРНОЇ СХЕМОТЕХНІКИ

ТЕМА 2.1. ЛОГІЧНІ СХЕМИ З ПРИКЛАДУ ДВІЙКОВОГО СУМАТОРА

План

1. Однорозрядний двійковий суматор;
2. Чотирирозрядний двійковий суматор;
3. Оптимізація суматора з метою прискорення перенесення для арифметики великої розрядності;
4. Арифметико-логічний пристрій (блок) процесора.

У цій лекції ми розглянемо приклад важливої логічної схеми - двійкового суматора, який виконує додавання двох чисел, поданих у двійковій системі обчислення.

Однорозрядний суматор

Спочатку розглянемо однорозрядний суматор, який дозволяє складати два біти. Для цього згадаємо алгоритм складання «в стовпчик», універсальний для систем обчислення з будь-якою основою. Цей алгоритм пропонує складати числа, починаючи з молодших розрядів, причому на вході кожного розряду крім доданків є ще й перенесення з попереднього, що дорівнює 0 або 1, а на виході крім суми також дорівнює 0 або 1 перенос до наступного розряду.

```
  1              1
 115            101
 116           +001
 ---           ----
 231            110
```

Рисунок 7 - Додавання 3-розрядних чисел «у стовпчик» з перенесенням (перенесення виділено червоним) у десятковій (ліворуч) та двійковій (праворуч) системах обчислення

Очевидно, що для двійкової системи (мова про один розряд) не тільки перенесення, а й доданки, і сума представлені одним бітом (одною двійковою).

Оскільки ми вільні інтерпретувати значення 0/1 і як ціле число і як брехню/істину, то можемо розглядати логічні схеми як арифметичні функції, які оперують числами в двійковому записі (тобто представленими цифрами 0 і 1). При однорозрядному додаванні ми керуємося такими правилами:

• S містить результат суми двох доданків *a* й *b* та переносу з попереднього розряду *c* за модулем 2: $s = a \oplus b \oplus c$;

• c' містить значення перенесення; перенесення виникає при складі двох однорозрядних двійкових чисел, коли їх сума плюс значення перенесення з попереднього розряду перевищує 1: $c' = (a \wedge b) \vee (b \wedge c) \vee (c \wedge a)$.

Отже, маємо функцію $f(a, b, c) = (s, c')$. Таблиця істинності цієї функції представлений у табл. 2. Очевидно, що заданий однорозрядний суматор є одним кроком для складання багаторозрядних двійкових чисел.

Таблиця 2 - Таблиця істинності для однорозрядного суматора

a	*b*	*c*	*s*	*c′*
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

Реалізація однорозрядного суматора за допомогою вентилів, тобто логічна схема суматора, представлена на рис. 7. Переконаємося, що ця схема справді робить те, що потрібно, тобто відповідає табл. 2.

• Розряд *s* суми. Входи *a* і *b* схеми надходять на вхід вентиля (1) «що виключає або», а його вихід і вхід, у свою чергу, - на вхід вентиля (2) «що виключає або». Вихід вентиля (2) є одночасно і виходом схеми. Таким чином,

отримуємо, що $s = (a \oplus b) \oplus c = a \oplus b \oplus c$, тобто на виході, ми дійсно маємо розряд суми.

• Перенесення до наступного розряду $c' = (a \wedge b) \vee \big(c \wedge (a \oplus b)\big)$. Ця форма відрізняється від раніше запропонованої для c', але легко переконатися, що вона дає той же результат, причому з використанням меншої кількості вентилів. Дійсно, $c \wedge (a \oplus b) = c \wedge (a \vee b) \wedge \neg(a \wedge b) = \neg(a \wedge b) \wedge \big((b \wedge c) \vee (c \wedge a)\big)$ отже, $(a \wedge b) \vee \big(c \wedge (a \oplus b)\big) = (a \wedge b) \vee \big(\neg(a \wedge b) \wedge \big((b \wedge c) \vee (c \wedge a)\big)\big) = (a \wedge b) \vee (b \wedge c) \vee (c \wedge a)$, тобто ми отримуємо саме те, що нам і потрібно.

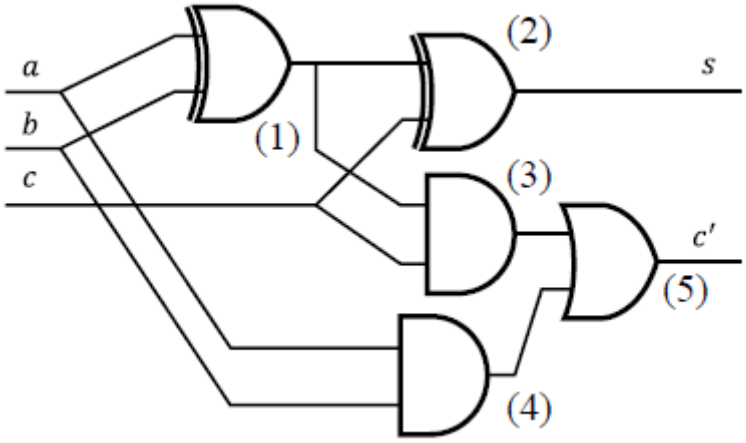

Рисунок 8 - Логічна схема однорозрядного суматора

Чотирирозрядний двійковий суматор

Тепер, коли ми вміємо складати однорозрядні числа, перейдемо до додавання двійкових багаторозрядних чисел. Без втрати спільності розглянемо складання 4-розрядних чисел.

Знову пригадаємо додавання «в стовпчик» багаторозрядних чисел. Зауважимо, що воно «сконструйоване» з безлічі однорозрядних складень із перенесенням. Якщо взяти до уваги, що система обчислення у нас двійкова та логічна схема додавання однорозрядних чисел у нас вже є, то можна сконструювати з набору однорозрядних суматорів багаторозрядний. Для зручності позначимо вже збудований нами однорозрядний суматор так, як показано на рис. 9.

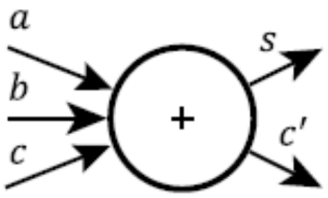

Рисунок 9 - Умовне позначення однорозрядного суматора

Чотирьохрозрядний суматор, що складається з цифр, що складаються з цифр $a_0a_1a_2a_3$ і $b_0b_1b_2b_3$ відповідно (молодші розряди ми записали зліва), можна представити як функцію наступного виду:

$$f(a_0, a_1, a_2, a_3, b_0, b_1, b_2, b_3, c_{\to 0}) = (s_0, s_1, s_2, s_3, c_{3\to}).$$

Логічна схема для цієї функції показана на рис. 10. Кожен із чотирьох однорозрядних суматорів на цій схемі отримує два біти вихідних доданків та біт перенесення з попереднього розряду, а видає біт суми та біт перенесення до наступного розряду.

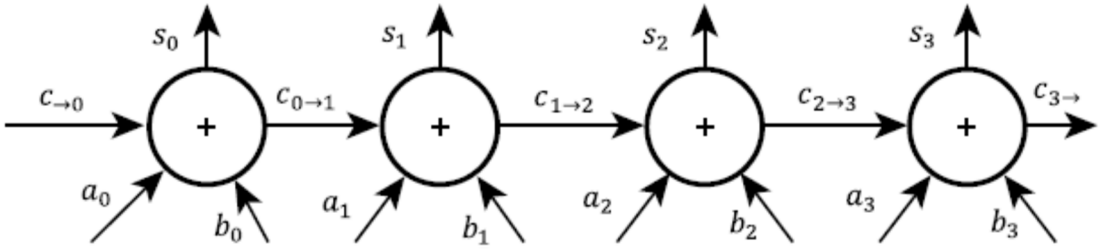

Рисунок 10 - Умовне позначення однорозрядного суматора

Уважні здобувачі могли помітити вхід $c_{\to 0}$, на який подається 0, і вихід $c_{3\to}$. Вони залишені на схемі не лише для підтримки належного ступеня спільності. За допомогою них можна об'єднувати 4-розрядні суматори в «довші», приєднуючи вихід перенесення попереднього до входу перенесення наступного так само, як ми з'єднували однорозрядні суматори, отримуючи 8-розрядні, 12-розрядні, 16-розрядні і так далі. Крім того, оскільки в сучасних комп'ютерах ці вхід та вихід доступні для програм (за допомогою так званого *прапора переносу*), то, користуючись ними, можна програмно реалізувати довгу арифметику, тобто запрограмувати функції, які на комп'ютері з 64-бітною арифметикою зможуть виконувати операції над цілими числами довільної довжини.

Оптимізація суматора з метою прискорення перенесення

Сучасні комп'ютери оперують, як правило, 32- або 64-бітними цілими числами. При цьому при додаванні таких чисел вузьким місцем є перенесення: сигнал перенесення повинен бути *послідовно* оброблений у всіх розрядах двійкового представлення числа починаючи з молодшого і закінчуючи старшим. Послідовне виконання перенесення є причиною суттєвого уповільнення роботи суматорів. Для прискорення перенесення при проектуванні сучасних суматорів використовується спеціальний прийом розпаралелювання, який ми розглянемо з прикладу 64-бітного суматора.

64-бітовий суматор реалізується як два 32-бітові — один для молодшої 32-бітної секції числа, а інший — для старшої. Молодша секція, що є звичайним 32-бітовим суматором, складає біти $a_0, a_1, \dots, a_{31} + b_0, b_1, \dots, b_{31}$ і видає біти суми $s_0, s_1 \dots, s_{31}$, а також біт переносу $c_{31\to32}$. Старша секція складає з урахуванням перенесення $c_{31\to32}$ старші біти $a_{32}, a_{33}, \dots, a_{63} + b_{32}, b_{33}, \dots, b_{63}$. Але щоб не чекати перенесення з молодшої секції, старша організована наступним чином: вона складається з двох 32-бітних суматорів, які одержують на вхід одні й ті ж доданки $a_{32}, a_{33}, \dots, a_{63}$ і $b_{32}, b_{33}, \dots, b_{63}$, і при цьому один із суматорів завжди отримує біт переносу $c_{31\to32} = 0$, а інший — $c_{31\to32} = 1$. Суматори старшої секції «запускаються» одночасно один з одним і з суматором молодшою та одночасно з нею закінчують роботу. На момент закінчення роботи цих трьох 32-бітних суматорів (одного «молодшого» і двох «старших») стає відомо дійсне значення перенесення $c_{31\to32}$, і в підсумкову суму потрапляють результуючі біти лише одного зі «старших» суматорів — того, який працював в умовах правильного припущення значення вхідного перенесення $c_{31\to32}$.

Описаний підхід можна описати словами «Зроби все, що можеш, якнайшвидше, частина з цього стане в нагоді, решту викинемо». Така стратегія обчислень називається *спекулятивною*. Спекулятивність ускладнює суматор у

нашому прикладі: вона призводить до того, що фактично результати 1/3 обчислень (одного з трьох 32-розрядних суматорів) виконуються даремно. Тим не менш, вона популярна, оскільки дозволяє прискорити роботу суматора майже вдвічі. Іноді суматори поділяють і на більшу кількість секцій.

Арифметико-логічний пристрій

Суматор, мультиплікатор (логічна схема для множення двійкових чисел), а також модулі для виконання логічних операцій та деякі інші логічні схеми складають арифметико-логічний пристрій (АЛП), який є одним з основних блоків процесора.

Про важливість арифметичних та логічних операцій ми вже говорили у минулій лекції. Арифметико-логічний пристрій — це функціональний блок процесора, який у порівнянні зі звичайним суматором можна вважати своєрідним «швейцарським ножем»: «зовні» він багато в чому нагадує звичайний суматор, але функціональність у нього набагато ширша. З іншими блоками процесора його зв'язують входи та виходи, описані нижче.

1. Як і у звичайного суматора, *АЛП* має два входи для першого і другого аргументів бінарних операцій (наприклад, додавання або множення); для унарних операцій (зміна знака, логічне заперечення) використовується одне із них.

2. Як і у звичайного суматора, *АЛП* має вихід для результату операції.

3. Як і у звичайного суматора, *АЛП* має вхід і вихід перенесення.

4. *АЛП* також має керуючий вхід, який задає, яку саме наступну операцію слід виконати, тобто який із інструментів швейцарського ножа потрібно зараз використовувати.

Залежно від значення на керуючому вході (4) до входів та виходів (1–3) підключається та чи інша логічна схема, що входить до складу АЛП:

• суматор;

• побітові логічні операції;

• побітове зрушення;

• мультиплікатор та ін.

Зробимо невеликий огляд реалізацій згаданих арифметико-логічних операцій.

Додавання виконується за допомогою суматора. Віднімання виконується також суматором, оскільки очевидно виражається через складення, що ми обговорювали в попередніх лекціях.

Побітові логічні операції — це операції, які виконуються побітово, наприклад, побітове заперечення двійкового числа виконує заперечення кожного його біта, побітова операція «і» двох чисел виконує «і» для кожної пари відповідних бітів і т. д. Таким чином, для побітового заперечення, отримуючи на вхід 32-бітове двійкове число $a_0, a_1, \dots, a_{31}$, видає наступний результат $\neg a_0, \neg a_1, \dots, \neg a_{31}$. Побітове «і» (операція & в мові С), отримуючи на вхід два 32-бітові числа $a_0, a_1, \dots, a_{31}$ і $b_0, b_1, \dots, b_{31}$, видає число $a_0 \wedge b_0, a_1 \wedge b_1, \dots, a_{31} \wedge b_{31}$. Аналогічно діють й інші побітові операції — «або», «що виключає або» тощо. Оскільки, на відміну від суматора, всі біти обробляються незалежно, проблем із затримкою перенесення не виникає і побітові операції завжди виконуються за один такт.

Зсув двійкового числа — це операція, що зміщує розряди (біти) числа від молодшого до старшого, дописуючи праворуч нуль і втрачаючи старший біт (лівий зсув, позначається “<<” в мові С) або від старшого до молодшого, дописуючи зліва нуль і втрачаючи молодший біт (правий зсув “>>” в мові С). Щоб зрушити аргумент a на 1 біт вліво, АЛП приєднує вихідні біти до відповідним вхідним. Зсув числа вліво на 1 біт еквівалентний множенню на 2, але виконується набагато швидше. Аналогічно працює зрушення вправо, і він еквівалентний поділу на 2 націло. Оскільки при рухах АЛП лише з'єднує виходи з відповідними входами, рухи, так само як і побітові операції, виконуються за один такт. Зазначимо, що сучасні АЛУ за один такт вміють зрушувати числа і на довільну кількість бітів, а також виконувати деякі інші різновиди операцій, що зсувають біти.

Множення виконується мультиплікатором. Якщо ми згадаємо алгоритм множення «в стовпчик», то побачимо, що він складається із зрушень, складень та множень на однорозрядне число. У разі двійкової системи множення на однорозрядне число реалізується тривіально: оскільки однорозрядне число може бути або 0, або 1, то й добуток дорівнює або іншому множнику, або нулю. Таким чином, мультиплікатор можна виготовити у вигляді логічної схеми. Але вона вийде дуже громіздкою: у ній буде вже дуже багато вентилів та зв`язків між ними. Тому в деяких АЛП операція множення виконується не єдиною логічною схемою, а як послідовність мікрооперацій (про мікрооперації ми розповімо в наступних лекціях) із запам'ятовуванням проміжних результатів у спеціальних внутрішніх регістрах, які використовуються тільки в АЛП. Щоб краще уявити собі роботу такого АЛП, спробуйте написати програму будь-якою мовою програмування, яка множить два числа, користуючись для цього лише операціями зсуву та додавання. За аналогією з множенням розподіл також можна реалізувати за допомогою логічної схеми, але вона вийде ще складніше, тому розподіл також часто реалізується у вигляді послідовності мікрооперацій.

Зазначимо, що програмна реалізація множення, яку ми пропонуємо виконати читачам, — не просто «навчальна» вправа. АЛП деяких простих процесорів, наприклад Intel 8080, дозволяло виконувати лише додавання, віднімання, зрушення та побітові операції, а збільшення цих процесорів дійсно доводилося реалізовувати вже програмістам.

ТЕМА 2.2. СУЧАСНІ ПРОЦЕСОРИ

План

1. Основні характеристики процесора: тактова частота, розрядність, машинна мова та система команд.

2. Багатоядерні процесори;

3. Короткий огляд сучасних сімейств процесорів - Intel x86, ARM, MIPS, AVR.

Як ми вже говорили вище, комп'ютер складається з великої кількості електронних, а також певної кількості механічних пристроїв. Більшість із них призначені або для зберігання та передачі даних (оперативна пам'ять, жорсткий диск, мережевий адаптер), або для виконання службових функцій (електричне харчування, охолодження). Головним завданням комп'ютера є виконання програм, і її виконує центральний процесор. Наявні в сучасних комп'ютерах інші пристрої, які можуть бути досить інтелектуальними (наприклад, веб-камера або звуковий адаптер), також можуть мати власні процесори та самостійно перетворювати дані, проте функціональність цих пристроїв обмежена і вони виконують лише вузький набір спеціальних завдань.

Центральний процесор (Central Processing Unit, CPU) є основним модулем комп'ютера і призначений для виконання програм.

Основними характеристиками процесора є:

• тактова частота;

• розрядність;

• машинна мова та система команд.

Тактова частота процесора

Важливо, щоб процесор та інші вузли комп'ютера виконували свої дії синхронно, оскільки інакше їхня діяльність виявиться неузгодженою. Так, наприклад, різні вузли комп'ютера повинні знати, коли саме в регістрах процесора або на виході логічних схем будуть готові коректні дані. Якщо

почати зчитувати ці дані раніше за сорок (наприклад, зчитувати дані на виході описаного в попередньому розділі суматора, не дочекавшись повного проходження перенесення), то вони виявляться некоректними і в роботі ЕОМ відбудеться збій.

Звичайного годинника для вимірів часу з цією метою недостатньо, оскільки потрібна дуже висока точність, зокрема, у зв'язку з тим, що вимірювати доводиться невеликі проміжки часу — частки наносекунд і навіть менше. Зрештою, в даному випадку не потрібен абсолютний час, достатньо лише відносного — тобто потрібно відповідати на запитання на кшталт «скільки пройшло часу з моменту виникнення тієї чи іншої події?», «скільки часу треба чекати?» тощо.

Облік часу в комп'ютері здійснюється спеціальним способом — за допомогою підрахунку кількості тактових імпульсів, що були згенеровані в період від однієї події до іншої. Тактові імпульси генеруються через рівні проміжки часу тактовим генератором - спеціальною електронною схемою, вбудованою в центральний процесор комп'ютера або його материнську плату. Таким чином, потік тактових імпульсів утворює потік часу всередині комп'ютера.

Кількість тактових імпульсів в одиницю часу, що генеруються тактовим генератором, називається ***тактовою частотою*** *процесора. Тактова частота вимірюється в герцах (Hz). 1 Hz (1 Гц) відповідає одній події за секунду - таким чином, 1 Hz = 1 c^{-1}.*

Тактова частота є однією з важливих характеристик процесора і комп'ютера в цілому: чим вона вища, тим більше операцій процесор виконує за одиницю часу, тим вища його продуктивність. Зважаючи на важливість цієї характеристики, часто говорять про тактову частоту всього комп'ютера, маючи на увазі тактову частоту його процесора.

Неможливо досягти значного збільшення продуктивності комп'ютера шляхом простого збільшення тактової частоти: для того, щоб електронні схеми працювали на високих частотах, вони повинні бути сконструйовані

відповідним чином. Однак до певної міри підвищити тактову частоту готового комп'ютера можливо це називають розгоном (Overclocking) процесора. Розгін дозволяє підвищити продуктивність комп'ютера, але він вимагає спеціальних знань і досвіду для акуратного підбору компонентів, що допускають розгін компонентів, підстроювання параметрів електроживлення, організації додаткового охолодження тощо. буд. Подібна практика є досить поширеною, але виробниками електроніки вона не рекомендується, оскільки підвищує ризик виходу комп'ютера з ладу.

Процесор є найвищим модулем ЕОМ, тому тактова частота, що використовується для синхронізації різних блоків усередині процесора, є занадто високою для інших модулів комп'ютера. Зокрема ця тактова частота виявляється високою для синхронізації взаємодії процесора з іншими модулями. Тому в сучасних ЕОМ, як правило, є дві тактові частоти — внутрішні та зовнішні.

Внутрішня тактова частота ЕОМ використовується всередині процесора для синхронізації його блоків і становить на сьогоднішній день 1 GHz і більше.

Зовнішня тактова частота ЕОМ використовується для синхронізації процесора та інших пристроїв комп'ютера, з якими він взаємодіє, і становить на сьогоднішній день сотні MHz.

Для ЕОМ першого покоління внутрішня тактова частота була обмежена значением100 KHz, і це означає, що вони могли здійснювати до 100 000 операцій на секунду. Оперативна пам'ять цих комп'ютерів будувалася на базі електромеханічних реле, і її швидкодія становила всього кілька Hz, тому і зовнішня тактова частота ЕОМ першого покоління становила ті самі кілька Hz. Для наступних ЕОМ, що мали електронну оперативну пам'ять, зовнішня і внутрішня тактові частоти відрізнялися вже не так значно. Типовою була відмінність у кілька разів, в окремих випадках частоти могли збігатися. Для ЕОМ другого покоління верхня межа внутрішньої тактової частоти збільшилася до 1 MHz, що відповідало мільйону операцій на секунду, для

третього покоління — 10 MHz і, відповідно, 10 мільйонів операцій на секунду. Тактова частота сучасних процесорів, які у ЕОМ четвертого покоління, становить до 5 GHz. Значення тактової частоти ряду відомих процесорів наведені у таблиці 3.

Таблиця 3 - Внутрішня тактова частота ряду відомих процесорів

Назва процесора	Рік випуску	Тактова частота
Intel 4004	1971	740 KHz
Motorola 6800	1974	2 MHz
Intel 80186	1982	6 MHz
Intel 80486 DX	1989	20 MHz
Intel 80486 DX4	1994	100 MHz
Pentium 4	2000	1,6 GHz
Intel Xeon Westmere	2010	3,6 GHz
IBM zEC12	2012	5,5 GHz

Зазначимо, що довільно підняти тактову частоту не дозволяє обмежити передачу імпульсів швидкістю світла та неможливістю зробити розміри транзисторів та інших напівпровідникових елементів меншими, ніж розмір атома.

Крім цих обмежень необмежено збільшувати тактову частоту не дозволяють властивості напівпровідників, що використовуються. Кремнієві діоди та транзистори управляють проходженням електричних сигналів, змінюючи свою електричну провідність. Вони роблять це дуже швидко, але в масштабах сучасної електроніки аж ніяк не миттєво: змінити свій статок вони можуть «лише» кілька мільярдів разів на секунду. В останні роки фізиками ведуться дослідження, присвячені можливості заміни традиційних кремнієвих транзистори на транзистори із застосуванням графену, очікувана швидкість «спрацьовування» яких буде в сотні разів вищою. Але поки що можна говорити лише про цікаві перспективи прискорення електронних схем, до впровадження нових технологій в індустрію ще далеко.

Розрядність процесора

Іншою важливою характеристикою процесора є його розрядність, яка визначає наступні найважливіші характеристики комп'ютера.

1. Кількість біт, які процесор може обробити за одну операцію над цілими числами. Це можуть бути як арифметичні операції (наприклад, додавання, віднімання, множення), так і побітові логічні (наприклад, «і», «або», «не»).

2. Розмір арифметичних регістрів процесора, тобто регістрів, призначених зберігання цілих чисел. Регістри процесора, як найшвидший вид пам'яті, інтенсивно використовуються для зберігання аргументів цілих операцій, приклади яких ми наводили вище, та їх результатів.

3. Кількість біт у шині даних і, отже, кількість біт, яку процесор може за одну операцію (за один такт) прочитати з оперативної пам'яті або записати до неї.

4. У більшості сучасних процесорів розрядність визначає максимальний обсяг оперативної пам'яті, що безпосередньо адресується процесором. Ця величина залежить від розміру адресних регістрів, призначених для зберігання адрес даних (у тому числі й машинного коду) в оперативній пам'яті.

5. При програмуванні мовою C, зазвичай, розрядність задає розміри довгого цілого числа (*long int*) і вказівника (*void* *).

Процесор із розрядністю 64 біта та тактовою частотою 1 GHz може обробити за секунду стільки ж даних, скільки процесор із розрядністю 32 біта та тактовою частотою 2 GHz. Практично всі сучасні мікропроцесори, що випускаються компаніями Intel та AMD, є 64-розрядними, але для сумісності з попередніми версіями підтримують 32-бітовий режим.

Сумісність процесорів означає, що у нових, 64-розрядних процесорів з'являється можливість виконувати старі програми, скомпільовані в 32-розрядному режимі для старих 32-розрядних процесорів.

Як і у випадку з тактовою частотою, процесор також має *внутрішню* і *зовнішню* розрядність. Внутрішньою розрядністю називають розрядність

регістрів процесора. Зовнішня розрядність — це розрядність шин даних та адреси, до яких підключено процесор. У багатьох сучасних процесорів ці розрядності збігаються, але це не завжди так. Так, наприклад, у Intel 8086 і внутрішня, і зовнішня розрядність - 16 біт, а у його більш дешевого аналога Intel 8088 внутрішня розрядність 16 біт, а зовнішня - 8. Поділ внутрішньої і зовнішньої розрядностей дозволяє Intel8 виконує ті ж програми), але при цьому здешевити шину даних та оперативну пам'ять, знизивши їхню розрядність (ціною зниження пропускної спроможності). Ще одним прикладом є процесор Intel Pentium I: внутрішня розрядність у нього, як і у попередніх процесорів сімейства, складає 32 біти, а зовнішня — 64. З одного боку, це дозволило зберегти зворотну сумісність з 32-бітними програмами, з іншого — прискорити обмін даними між оперативною пам'яттю в оперативній пам'яті. процесорах завантажуються не безпосередньо з оперативної пам'яті в регістри, а спочатку потрапляють у кеш-пам'ять, про яку ми розповімо нижче).

Машинна мова та система команд процесора

Сучасні високорівневі мови програмування дозволяють представляти і алгоритми, і дані таким чином, що програміст може, працюючи з програмним кодом, мислити в термінах вихідних завдань. Для процесора, який зрештою виконує написані програми, такий рівень абстракції недоступний. Процесор може виконати лише програму *машинною мовою.*

Машинна мова (Machine Language) процесора *є засобом представлення програми в пам'яті комп'ютера за допомогою двійкового машинного коду, який розуміється центральним процесором, тобто може бути ним виконаний. Машинна мова складається з (і) системи команд (Instruction Set) – набору операцій, які може виконувати процесор; (іі) видів даних (різні числа, адреси та ін.), для яких визначено ці операції; (ііі) способу кодування команд та даних у двійковому коді.*

Система команд визначає операції, які може виконувати процесор. Насамперед, це часто використовувані операції:

• додавання/множення/розподіл у цілісній арифметиці;

• поширені математичні функції (експонента, логарифм, тригонометричні функції та ін.);

• завантаження/вивантаження даних у процесор;

• операції над масивами даних тощо. буд.

Операції орієнтовані певні види даних, із якими процесор може працювати. Наприклад, це можуть бути цілі числа різної довжини зі знаком і без, речові числа різної точності, символьні дані, покажчики. Нарешті, кодування визначає, яким чином ці операції та дані повинні зберігатися в оперативній пам'яті, адже там зберігаються в результаті масиви байтів. Наприклад, різні сімейства процесорів можуть по-різному впорядковувати у пам'яті байти у багатобайтових цілих чисел — від молодшого до старшого чи старшого до молодшого. Подання чисел, при якому спочатку в пам'яті зберігаються байти зі старшими розрядами, а потім з молодшими, називається Big-endian. 32-бітове число ABCD321016 в оперативній пам'яті комп'ютера Big-endian (наприклад, сімейства System/360) буде представлено чотирма байтами: AB16, CD16, 3216, 1016. У комп'ютері Little-endian (наприклад, Intel x86 і більшість інших сучасних сімейств) 3216, CD16, AB16. Терміни Little- і Big-endian є посиланням на роман Дж. Свіфта «Подорожі Гуллівера» і були введені в 80-ті роки XX століття.

Машинні мови різних сімейств процесорів суттєво відрізняються, подібно до того, як відрізняються мови різних народів. Особливості природних мов визначаються середовищем проживання та історією їх носіїв, а машинних — призначенням відповідних комп'ютерів та спадкоємністю всередині сімейств. Наприклад, процесори комп'ютерів сім'ї System/360 хоч і відносяться до архітектури фон Неймана, проте підтримують крім двійкової та десяткової арифметики, що важливо для фінансових обчислень. Інший приклад — поступове (протягом понад 40 років) розширення підтримки речової арифметики та обробки масивів чисел у процесорах сімейства Intel x86 у міру зростання кількості мультимедіа-додатків для персональних комп'ютерів.

Багатоядерні процесори

Такі фундаментальні обмеження, як розмір атома та швидкість світла, не дозволяють довільно збільшувати тактову частоту та розрядність процесора. Зараз збільшення продуктивності комп'ютерів в основному здійснюється шляхом розпаралелювання різних операцій, що виконуються процесором.

Розпаралелювання означає організацію паралельної роботи різних блоків процесора, а також паралельне виконання окремих команд або груп команд програми.

Застосовуються різні техніки підвищення продуктивності процесорів за допомогою розпаралелювання. Однією з найвідоміших популярних зараз технік є створення багатоядерних процесорів.

Багатоядерний процесор (Multicore CPU) *- це процесор, який дозволяє одночасно (паралельно) виконувати кілька програм, використовуючи кілька ядер.*

Ядро (CPU Core) *є частиною процесора, яка може самостійно декодувати прочитані з пам'яті команди програми та послідовно їх виконувати.*

Багатоядерний процесор не стільки швидше виконує звичайні (однопотокові) програми, скільки дозволяє швидко виконати кілька програм одночасно. Однак багатопотокові програми багатоядерний процесор виконує істотно швидше, ніж одноядерний, надаючи різним потокам ядра, що паралельно працюють. Тому в тих випадках, коли необхідно підвищити продуктивність однопотокових програм, потрібно використовувати паралельні алгоритми та переписувати ці програми у багатопоточному стилі.

Популярність багатоядерних процесорів зумовлена їхньою вигідними економічними характеристиками. При виробництві один багатоядерний процесор коштує дешевше кількох одноядерних процесорів з такою самою сумарною кількістю ядер. Причина в тому, що ядро процесора це ще не весь процесор. Крім ядер у процесорі є ще набір блоків, що здійснюють звернення до шин, що обробляють різні сигнали, кешують дані і т. д. Подібні блоки

реалізовані в багатоядерному процесорі в єдиному екземплярі, тоді як ядер, що одночасно виконують різні послідовності команд, в одному процесорі може бути до декількох десятків. Зазначимо, що значно простіше і дешевше випускати однопроцесорний комп'ютер, ніж багатопроцесорний. Зрештою, реальна потреба збільшення продуктивності комп'ютерів ціною розумного подорожчання визначила нинішню популярність саме багатоядерних процесорів.

На сьогоднішній день майже всі процесори, у тому числі мобільні, мають кілька обчислювальних ядер. Наприклад, процесори Intel Core i7, поширені в сучасних персональних комп'ютерах, можуть мати 2, 4, 6 або 8 ядер, а процесор A13 сучасних смартфонів Apple має 6 ядер. Ядра мобільних процесорів можуть мати різні характеристики — наприклад, виділяються повільні ядра і швидкі ядра. Повільні ядра мають економічне енергоспоживання, але відносно невисоку продуктивність і процесор використовує їх постійно. «Швидкі» ядра мають високу продуктивність, але споживають багато енергії і використовуються процесором лише за необхідності. Наприклад, згаданий вище мобільний процесор A13 має 6 ядер, з яких 4 є "повільними" і 2 - "швидкими".

Короткий огляд деяких сучасних сімейств процесорів

Розглянемо наступні сучасні сімейства процесорів: Intel x86, ARM, MIPS, AVR.

Цей набір дозволяє отримати уявлення про різні сучасні процесори, призначені для настільних комп'ютерів та мобільних пристроїв, для вбудованих та телекомунікаційних систем, а також для суперкомп'ютерів.

Сімейство процесорів Intel x86

Процесори цього сімейства протягом сорока років є основою для комплектації комп'ютерів IBM PC та наступних сумісних із ними комп'ютерів. Ці комп'ютери становлять зараз більшість персональних комп'ютерів на планеті. Також завдяки достатній обчислювальній потужності та багатим функціональним можливостям процесори цього сімейства часто

використовуються при побудові багатопроцесорних суперкомп'ютерів, які використовуються для високопродуктивних наукових обчислень. Основним виробником цих процесорів традиційно є компанія Intel, але крім цієї компанії сумісні процесори випускаються такими компаніями, як AMD, Cyrix, VIA Technologies. Тому, строго кажучи, коректніше позначати це сімейство як x86, тому що в цьому випадку ми говоритимемо про процесори різних виробників, не тільки компанії Intel. Але в рамках нашого курсу нам такий рівень спільності не потрібний.

Сімейство процесорів ARM.

На ринку мобільних пристроїв в останні десятиліття домінують процесори сімейства ARM. Компанія Arm Holdings, яка розробила їх архітектуру в 1990-х роках, сама процесорів не виробляє, але ліцензує їх виробництво іншими компаніями — наприклад, Qualcomm, Mediatek, NVIDIA. Архітектура ARM показала свою придатність не лише мобільних пристроїв. В останні роки на базі процесорів ARM також випускаються ноутбуки (наприклад Chromebook), настільні комп'ютери (наприклад, сучасні комп'ютери Apple на основі ARM-сумісного процесора Apple M1), одноплатні комп'ютери Raspberry Pi. Нарешті, в Японії створено суперкомп'ютер Fugaku, побудований на базі більше ніж 150000 процесорів Fujitsu A64FX архітектури ARM. Цей суперкомп'ютер створено в Центрі обчислювальних наук Інституту фізико-хімічних досліджень у місті Коба, Японія. Він став переможцем низки рейтингів суперкомп'ютерних обчислювальних систем у червні 2020 року. Офіційно Fugaku було введено в експлуатацію у березні 2021 року.

Сімейство процесорів MIPS було розроблено в середині 1980-х років у Стенфордському університеті. Наразі їх виробництво ліцензує компанія MIPS Technologies, Inc. Як і у випадку з ARM, компанія, яка має права на архітектуру процесорів, сама не займається їх виробництвом. Серед виробників процесорів архітектури MIPS можна перерахувати такі відомі компанії як Siemens, Philips, Toshiba, NEC. Зараз процесори MIPS використовуються переважно у вбудованих комп'ютерах, керуючих, наприклад, верстатами і ліфтами, а також

для обладнання обчислювальних мереж - в роутерах, шлюзах і т. д. Зараз багато виробників обчислювальних пристроїв створюють свої процесори на базі архітектури MIPS, додаючи в них спеціалізовані функції, наприклад, у разі телекомунікації – функції для обробки сигналів.

Сімейство процесорів AVR. Для розробки простих пристроїв популярними є однокристальні комп'ютери сімейства AVR компанії Atmel. Однокристальними вони називаються тому, що в одній інтегральній схемі (тобто на одному кристалі) містяться всі пристрої, необхідні для функціонування комп'ютера. Крім власне процесора інтегральна схема включає кілька кілобайт оперативної пам'яті і флеш-пам'яті для зберігання програми, тактовий генератор, аналогово-цифрові і цифрово-аналогові перетворювачі для безпосереднього з'єднання з датчиками і схемами управління різними пристроями. *Аналогово-цифровий перетворювач (АЦП)* - це електронна схема, що перетворює значення фізичної величини (як правило, напруги) у цифровий вигляд, тобто в машинне число. *Цифрово-аналоговий перетворювач (ЦАП)* - електронна схема, що виконує зворотне перетворення. Наприклад, аудіосигнал, що видається звуковим адаптером, — аналоговий, і генерує його цифрово-аналоговий перетворювач, який одержує на вхід оцифрований звуковий сигнал.

Однокристальні обчислювальні пристрої, що поєднують функції процесора та периферійних пристроїв, називають ***мікроконтролерами***.

Цей термін, як і у випадку з мікропроцесорами, що вміщаються на одному кристалі, підкреслює їх «самодостатність»: фактично для роботи мікроконтролеру потрібно лише електроживлення. Мікроконтролери використовуються для керування материнськими платами, жорсткими дисками, для керування пристроями побутової техніки (мікрохвильові печі, холодильники, пральні машини та ін.). Крім компанії Atmel мікроконтролери випускаються багатьма іншими компаніями, наприклад, Siemens, Microchip, Fujitsu.

Повертаючись до мікроконтролерів AVR, відзначимо, що вони дуже економічні у споживанні електроенергії: пристрій, що складається з мікроконтролера AVR та кількох датчиків, може кілька місяців працювати, живлячись від трьох «пальчикових» батарейок. Існують мікроконтролери з ще меншим споживанням електроенергії, які можуть працювати кілька років від батареї для наручного годинника.

Простота, дешевизна, «самодостатність» і низьке енергоспоживання зробили мікроконтролери AVR популярними серед інженерів-початківців та радіоаматорів. У радіо- та робототехнічних гуртках вони часто застосовуються в освітніх цілях.

Нарешті, відзначимо, що мікроконтролери AVR мають гарвардську архітектуру. У випадку з однокристальними комп'ютерами, у яких всередині процесу знаходиться пам'ять як для машинного коду, так і для даних (тобто шин немає!), гарвардська архітектура дозволяє спростити пристрій.

ТЕМА 2.3. CISC- ТА RISC-АРХІТЕКТУРИ ПРОЦЕСОРІВ

План

1. CISC- та RISC-архітектура: визначення, особливості машинної мови, переваги та обмеження.

2. Приклад машинного коду програми на С для процесора Intel x86 (CISC-архітектура) та процесора ARM (RISC-архітектура).

Як ми переконалися вище, різні архітектури процесорів можуть помітно відрізнятися, але завдання підвищення швидкодії розумною ціною є актуальним для будь-яких архітектур. У рамках цієї лекції ми розглянемо дві поширені архітектури процесорів – CISC- та RISC.

Визначення, особливості, переваги та недоліки

Традиційний підхід до проектування процесорів ставив одним з головних завдань ефективну апаратну реалізацію операцій, що часто зустрічаються, і зручність написання програм мовою Асемблер і машинною мовою. Цей підхід називається CISC-архітектурою (Complicated Instruction Set Computer); цей термін перекладається як «комп'ютер зі складною системою команд».

CISC-архітектура *характеризується великою кількістю різноманітних команд, здатних виконувати складні, багатокрокові дії.*

При використанні даного підходу система команд і машинна мова мають властивості, перелічені нижче.

Виразність системи команд. Використовується велика кількість різноманітних команд — від найпростіших арифметичних до складних, що поєднують, наприклад, арифметичні операції та різні перевірки операндів (наприклад, на рівність і нерівність або стан окремих бітів у них). Системи команд деяких процесорів також підтримують роботу з масивами даних — копіювання, поелементне порівняння тощо.

Гнучкість під час завдання аргументів команд. Одна операція, наприклад додавання, може бути представлена в CISC-процесорі цілим

сімейством команд. Всі ці команди дозволяють різними способами задавати аргументи додавання, які, таким чином, можуть братися з машинного коду, а також з оперативної пам'яті — або за заданою адресою, або за адресою, заданою в регістрі. Подібні можливості дозволяють мінімізувати допоміжне копіювання даних (наприклад, з пам'яті в регістр) і допомагають ефективно кодувати за допомогою невеликої кількості команд, а часто за допомогою однієї єдиної команди, такі поширені операції, як, скажімо, порівняння двох змінних або складання змінної з елементом масиву.

Економне кодування – можливість створювати машинний код, який займає мінімальний обсяг пам'яті. Двійкове представлення різних команд в оперативній пам'яті має різну довжину та різний формат: найчастіші команди вміщаються в один байт, а на нечасто використовуваних не економлять, кодуючи їх п'ятьма або навіть більш байтами. Це дозволяє в середньому значно скоротити розмір машинного коду.

Істотно різна тривалість виконання команд. Прості команди, такі як обнуління регістру, виконуються в CISC-процесорах швидко, тоді як складні команди, такі як поділ, потребують значного часу.

Як приклади процесорів, що мають CISC-архітектуру, можна навести процесори знаменитих мейнфреймів IBM System/360, а також процесори сімейства Intel x86. Тепер перерахуємо недоліки CISC-архітектури:

• Розробка процесорів з виразною, але великою та складною системою команд є трудомістким завданням, що підвищує вартість розробки процесорів та ризик великої кількості помилок.

• Процесори, що випускаються, будучи складними, мають високі собівартість і ринкову ціну.

• Складна система команд ускладнює внутрішньо процесорні оптимізації, наприклад, автоматичне розпаралелювання виконання машинних команд.

Багато CISC-процесорів реалізують складні команди за допомогою *мікрокоду* – спеціалізованої машинної мови, що дозволяє задавати

послідовність дій (мікрооперацій), яку виконує одна машинна команда процесора. Таким чином, команди CISC-процесорів виявляються настільки складними, що їх доводиться описувати за допомогою додаткової машинної мови, яка виконується спеціалізованим процесором усередині процесора.

RISC-архітектура (Reduced Instruction Set Computer) була націлена на вирішення цих проблем.

***RISC-архітектура** процесора має на увазі невелику систему команд, кожна з яких має ефективну апаратну реалізацію, а складні операції, необхідні для прикладних програм, складаються з багатьох простих команд.*

Ось основні властивості процесорів із RISC-архітектурою.

• *Мінімалізм системи команд*. Машинні команди RISC-процесора мають просту функціональність, відповідно, час на їх декодування та виконання виявляється невеликим, а блоки процесора, які відповідають за декодування, є простими.

• *Відсутність гнучкості завдання аргументів*. Обробку та пересилання даних з оперативної пам'яті до регістрів процесора виконують різні команди, тобто немає арифметичних та логічних операцій, чиї операнди знаходяться в оперативній пам'яті та підтримують різні варіанти адресації. Відповідно, арифметичні та логічні операції реалізуються командами, які працюють лише з регістрами процесора. Наприклад, не можна за допомогою однієї команди скласти два числа, одне з яких перебуває в регістрі, а інше — у пам'яті, а потрібно окремою командою завантажити значення з пам'яті в певний регістр і лише після цього запустити команду додавання. Цей принцип у літературі часто називається **load-store architecture**, що саме означає обмін даними процесора з пам'яттю за допомогою окремих команд. Щоб компенсувати це обмеження, RISC-процесори зазвичай мають велику кількість регістрів загального призначення. Велика реєстрова пам'ять дозволяє постійно тримати багато даних усередині процесора і тим самим мінімізувати кількість операцій обміну даними з оперативною пам'яттю.

• *Уніфіковане кодування*. Двійкове представлення різних машинних команд (тобто відповідний ним машинний код) має однаковий формат та фіксований розмір. Більшість RISC-архітектур одна команда займає одне машинне слово. Таким чином, є проста і «одноподібна» машинна мова.

• *Фіксований час виконання команд*. Всі команди виконуються за одну і ту ж кількість тактів, що спрощує різні оптимізації — зокрема, конвеєризацію, про яку ми детальніше розповімо нижче. — вони вимагають виконання великої кількості зрушень, додавання/різниця і різних перевірок. Оскільки такі команди вибиваються з стрункої концепції RISC, для їх реалізації застосовують спеціальні прийоми. програма «знає», через скільки тактів розподіл завершиться, і також знає, де (у яких регістрах) будуть розташовані приватне та залишок від розподілу.

Ціною втрати виразності та економності машинної мови процесори з RISC-архітектурою досягають наступних переваг у порівнянні з CISC-процесорами:

• спрощення та здешевлення процесорів як при конструюванні, так і при виробництві;

• великі можливості для реалізації різних оптимізацій, зокрема внутрішньопроцесорного розпаралелювання.

Щоб наочно проілюструвати різницю між CISC- та RISC-архітектурами, розглянемо наступний приклад. Команда додавання (add) у процесорах сімейства Intel x86 (CISC-архітектура) має два аргументи — перший і другий доданок, а результат виконання команди поміщається на місце першого доданку. При цьому будь-який із доданків може бути або регістром, або числом, що зберігається в оперативній пам'яті. Усі можливі поєднання типів аргументів відповідають різним командам add. Таким чином, в процесорах Intel x86 є три різні команди add: регістр/регістр, регістр/пам'ять, пам'ять/регістр. Команди add типу пам'ять/пам'ять у процесорів Intel x86 відсутня, оскільки в рамках виконання арифметичних команд передбачається не більше одного звернення до пам'яті. Це зумовлено наступністю в рамках

сімейства Intel x86: у перших процесорах сімейства (Intel 8086 та Intel 8088) кілька звернень до даних в оперативній пам'яті суттєво уповільнили б реалізацію арифметики та вимагали б ускладнення процесора. Слід зазначити, що крім перерахованих вище команд складання у процесорів Intel x86 є й інші команди add, які призначені для роботи з константами, для використання різних варіантів адресації тощо.

У RISC-процесорі є одна команда add, і вона працює тільки з регістрами, тобто має вигляд регістр/регістр. Якщо один із доданків знаходиться в пам'яті, то він спочатку в явному вигляді (тобто окремою командою) має бути завантажений у відповідний регістр, а вже потім має бути викликана команда додавання. Замість цих двох команд у процесорах Intel x86 просто викликається відповідна команда add, яка спочатку завантажує відсутній операнд з пам'яті в процесор, а потім виконує додавання.

Цей простий приклад показує, що машинна мова RISC-процесора набагато менш дружня для програміста, що програмує мовою Асемблер або в машинному коді, ніж машинна мова CISC-процесора, дозволяючи обходитися меншою кількістю команд за рахунок гнучкості завдання аргументів. Але, як ми вже сказали вище, проста машинна мова дозволяє спростити і керуючий пристрій процесора - блок, відповідальний за керування іншими блоками процесора. На додаток до вже згаданого здешевлення процесора, це спрощення дає конструкторам RISC-процесорів ще одну непряму вигоду: можливість зробити керуючий пристрій процесора компактнішим і, як наслідок, скоротити час проходження сигналів усередині нього, що дозволяє збільшити тактову частоту процесора.

RISC-архітектура також має слабкі сторони.

• Зазначені вище оптимізації вимагають більших, ніж у випадку CISC-процесорів, зусиль зі створення компілятором із мов високого рівня оптимального машинного коду. Сучасні CISC-процесори справляються з цим «самостійно», хоч і ціною значного ускладнення. Докладніше про це ми

розповімо нижче у розділі про конвеєризацію та позачергове виконання машинних команд.

• Оскільки обсяг коду для RISC-процесора виходить більше, ніж для тих же програм, скомпільованих для CISC-архітектур, то збільшується навантаження на шини та оперативну пам'ять ЕОМ.

Найбільш відомими представниками RISC-архітектури є родини ARM (процесори цього сімейства є основою багатьох сучасних мобільних пристроїв, а також деяких персональних комп'ютерів і навіть суперкомп'ютерів) та MIPS (найпопулярніші як основа для обладнання обчислювальних мереж).

Однією з обставин, що вплинули на популярність RISC-архітектури (а це був рубіж 1970-х та 1980-х років), став той факт, що значна частина програм на той час розроблялася мовами високого рівня, і, таким чином, з машинною мовою вже мали справу компілятори, а не програмісти. Отже, необхідність зручної для людини машинної мови (одна з основних переваг CISC-архітектури) стала неактуальною. У той же час RISC-процесори хоч і потіснили CISC, але не витіснили їх остаточно. Більшість нових процесорів, що розробляються останніми роками, відносяться до вже існуючих сімейств, а більшість цих сімейств мають RISC-архітектуру. Проте для популярних CISC-родин процесорів, наприклад Intel x86 та IBM System/360 (і наступних), розроблено таку кількість ПЗ, що відмовитися від цих процесорів на користь інших, хай і досконаліших, вже неможливо з економічних міркувань. Зазначимо, що розробники сучасних CISC-процесорів не відмовляються від використання окремих ідей RISC. Так, у деяких процесорах сімейства Intel x86 (наприклад, Intel 80486) машинні команди транслюються у послідовності мікрооперацій, які вже виконує «прихований від програміста» внутрішній RISC-процесор.

Щоб наочно проілюструвати особливості CISC- та RISC-архітектур, наведемо *приклад*. Як CISC-процесора розглянемо процесор сімейства Intel x86, а як RISC-процесора візьмемо ARM. Розгляньмо наступну нескладну

програму на C і покажемо, який машинний код створюється компілятором для цієї програми для CISC- та RISC-процесорів.

```
int a, b;
bool res;
a = a + b;
b = 3 * b;
if (a > b)
  res = true;
else
  res = false;
```

Для компіляції цієї програми під платформу Intel x86 ми використовували компілятор Microsoft Visual C/C++ 19, для ARM — компілятор GNU C Compiler 8.

Результати трансляції обох архітектур представлені у табл. 4. Для кожного оператора програми на мові С надано команди машинної програми Intel x86 та ARM. Для кожної команди наводиться її адреса в оперативній пам'яті, машинний код (і те, й інше — у шістнадцятковому записі), а також асемблерне подання цієї команди (для більшої наочності).

Нижче наведено додаткові пояснення та коментарі до табл. 4.

Програма для процесора Intel x86 складається з 11 команд, а для ARM з 18. Арифметичні регістри в Intel x86 називаються символами, наприклад, *eax*, *ecx*, а у ARM вони зазвичай нумеруються, наприклад: *r0, r1, …* Це пов'язано з тим, що традиційно у RISC-процесорів арифметичних регістрів більше, ніж у CISC, оскільки першим переважно якнайбільше операндів заздалегідь розмістити в регістрах з тим, щоб під час виконання основних команд не звертатися до оперативної пам'яті.

Код команд процесора Intel x86 має різну довжину - від одного до чотирьох байтів разом з операндами, а код команд ARM займає в пам'яті чотири байти. У результаті програма для Intel x86 займає 34 байти, а ARM — 72. Ця закономірність зберігається й великих програм: машинний код для процесорів ARM зазвичай у 1,5–2,5 рази об'ємніше, ніж Intel x86.

Для Intel x86 компілятор використовує команду множення цілих чисел зі знаком (*imul*), а для ARM множення замінюється рухом і додаванням

(b*3→b*2+b→b<<1+b) — як уже обговорювалося вище, операція множення є ресурсномісткою, тому при компіляції програми під RISC-процесор скрізь, де можна уникнути використання команди машинного множення.

Для Intel x86 у командах безумовного (*jmp*) та умовного (*jle*) переходу параметр, що вказує, наскільки слід «стрибнути» щодо адреси поточної команди (так зване *зміщення*), задається в байтах (наприклад, *jmp.+4* означає: «перестрибни» через чотири байти машинного коду). У ARM усі команди, представлені у вигляді машинного коду, мають однакову довжину — по чотири байти. Тому команди переходу (*b* і *ble*) оперують не адресою, а одночасно номером команди, відповідно і зміщення задається над байтах, а кількості команд, причому крім наступної (наприклад, означає: «перестрибни» через дві команди).

Зазначимо також, що ми навмисно заборонили використаним компіляторам оптимізувати програму. При активованій оптимізації сучасні компілятори виконують багато перетворень, які в нашому випадку суттєво ускладнили б результуючий код, зробивши його непридатним для нашого ілюстративного завдання.

Таблиця 4 - Результати компіляції програми на С у машинний код для процесорів Intel x86 та ARM

Оператори на С	**Адр.**	**Маш. код**	**Асемблер**	**Коментар**
`int a, b;` `bool res;`				
`a = a + b;`	03	8b 45 f4	mov eax, DWORD PTR [ebp-0xc]	Завантаження змінної a в регістр EAX (пам’ять→регістр)
	06	03 45 f8	add eax, DWORD PTR [ebp-0x8]	Додавання змінної b до EAX, збереження результату в EAX
	09	89 45 f4	mov DWORD PTR [ebp-0xc], eax	Збереження результату в змінну a (регістр→пам’ять)
`b = 3 * b;`	0c	8b 4d f8	mov ecx, DWORD PTR [ebp-0x8]	Завантаження b у регістр ECX
	0f	6b c1 03	imul ecx, ecx, 3	Множення значення b на 3
	12	89 4d f8	mov DWORD PTR [ebp-0x8], ecx	Запис результату назад у b
`if (a > b)`	15	8b 55	mov edx,	Завантаження a у EDX

		f4	DWORD PTR [ebp-0xc]	
	18	3b 55 f8	cmp edx, DWORD PTR [ebp-0x8]	Порівняння a та b
	1b	7e 06	jle +6	Перехід, якщо a ≤ b (тобто якщо умова *не виконується*)
`res = true;`	1d	c6 45 f0 01	mov BYTE PTR [ebp-0x10], 1	Присвоєння `res = true`
`else`	22	eb 04	jmp +4	Безумовний перехід на кінець `else` (перескок наступного кроку)
`res = false;`	24	c6 45 f0 00	mov BYTE PTR [ebp-0x10], 0	Присвоєння `res = false`

ТЕМА 2.4. КОНВЕЄРИЗАЦІЯ

План

1. Визначення конвеєра.

2. Конфлікти під час конвеєрного виконання машинного коду.

3. Особливості обробки конфліктів у CICS- та RISC-архітектурах.

4. Приклад генерації машинного коду для CISC-процесора ARM та RISC-процесора MIPS для демонстрації прийомів обробки потенційних конфліктів конвеєризації на рівні компіляції.

5. Паралельне виконання команд з прикладу процесорів Intel x86.

Визначення конвеєра

Команди процесора часто використовують значну кількість блоків процесора і виконуються за кілька тактів. Наприклад, розглянемо команду складання числа з регістру з числом із пам'ят *add eax, DWORD PTR [ebp-0x8]* в процесорі Intel 80386. Ця команда виконується в такий спосіб.

1. На початку виконується читання числа з регістру до арифметико-логічного пристрою (АЛП).

2. Значення адресного регістру *ebp* і константа *–0x8* подаються на вхід адресному суматору (зауважимо, що тут йдеться не про суматора, що входить до складу АЛП, а про окремий адресний суматор, призначений для арифметичних дій над адресами; різних спеціалізованих суматорів у процесорі може бути достатньо.

3. Обчислюється адресу пам'яті другого операнда.

4. Виконується читання на цій адресі числа з пам'яті в тимчасовий регістр, з'єднаний з АЛП.

5. Виконується додавання двох чисел.

6. Результат записується в регістр.

Таким чином, ця команда виконується за 6 тактів (слід зазначити, що цей приклад суттєво спрощений для наочності). При цьому якщо не використовувати додаткові оптимізації, то на кожному такті задіяна лише

частина блоків процесора, а інші простоюють. Наприклад, при додаванні двох значень, що знаходяться в регістрах процесора, блок роботи з пам'яттю не діє.

Конвеєр – це технологія, призначена для паралельного виконання команд програми різними блоками процесора. Вона дозволяє завантажити блоки процесора під час виконання команд оптимально, без простоїв.

Наведемо приклад конвеєра. Розглянемо спрощений процесор, у котрого кожна команда виконується в три етапи, точніше, за три такти:

читання аргументів команди з пам'яті або регістру (ЧТН);

• виконання команди (ВИК);

• запис результату на пам'ять або регістр (ЗАП).

На рис. 12 зображено процес виконання цим процесором послідовності команд Команда 1, Команда 2, Команда 3, Команда 4, Команда 5 з використанням конвеєра. Розглянемо четвертий такт конвеєра, виділений на малюнку. Команда 1 вже повністю виконана, Команда 2 знаходиться на етапі запису даних (ЗАТ), Команда 3 — на етапі виконання (ВИК), Команда 4 — на етапі читання даних (ЧТН), а виконання Команди 5 процесор ще не приступав. Через те, що на кожному такті процесор виконує за допомогою конвеєра відразу кілька частин (етапів) команд, всі п'ять команд виконуються за 7 тактів, а не за 15, як було б при послідовному виконанні цього ланцюжка команд. Це виявляється можливим через те, що процесор паралельно виконує різні етапи команд.

Час →

Команди	**Такти**						
	1	**2**	**3**	**4**	**5**	**6**	**7**
Команда 1	ЧТН	ВИК	ЗАП				
Команда 2		ЧТН	ВИК	ЗАП			
Команда 3			ЧТН	ВИК	ЗАП		
Команда 4				ЧТН	ВИК	ЗАП	
Команда 5					ЧТН	ВИК	ЗАП

Рисунок 12 - Виконання машинних команд із використанням конвеєризації

Конфлікти під час конвеєрного виконання машинних команд

Результати виконання команд використовуються наступними командами програми. Подивимося, як це просте міркування узгоджується з конвеєром.

Наведений вище приклад показує, що Команда 3 не зможе використати результати Команди 2, оскільки свої вхідні дані вона читає раніше, ніж Команда 2 записує свої результати. У тому випадку, коли команді 3 дійсно потрібні результати команди 2, команди 2 і команди 3 називаються залежними.

Ситуації, що виникають під час конвеєризованого виконання команд, які перешкоджають коректному виконанню чергової команди, називаються ***конфліктами***.

Нижче наведено види конфліктів.

1. Конфлікт за даними між залежними машинними командами полягає в тому, що на конвеєрі одночасно знаходяться на різних стадіях виконання команди, які можуть бути коректно виконані лише суворо послідовно.

2. Конфлікт за ресурсами виникає в ситуації, коли двом командам на конвеєрі одночасно потрібен доступ до якогось блоку процесора, з яким одночасно може працювати тільки одна команда.

3. Конфлікт з управління полягає в тому, що наступна команда на конвеєрі є умовним переходом, але умова для нього ще не обчислена попередньою командою і незрозуміло, яку гілку умовного оператора слід завантажувати на конвеєр.

Оскільки конфлікти за даними є поширеною на практиці проблемою, наведемо низку прикладів таких конфліктів.

• Наступна команда на конвеєрі має читати дані на виході попередньої, але попередня ще не закінчила їхню обробку; при цьому порушується залежність читання після запису.

• Наступна команда записує дані, але вони використовуються (зчитаються) попередньою командою, тобто порушується залежність «запис після читання».

• Наступна команда записує (в регістри процесора або в оперативну пам'ять) свої результати раніше за попередню, через що попередня потім може перезаписати ці результати: при цьому порушується залежність «запис після запису».

Зробимо зауваження щодо третього виду конфліктів (за керуванням). Такі конфлікти додатково ускладнюються тим, що заздалегідь неясно, в яке місце програми відбудеться умовний перехід, і тому незрозуміло, з якої його гілки завантажувати на конвеєр наступні команди. У зв'язку з цим у багатьох процесорах є система *передбачення умовних переходів*, яка збирає статистику з переходів і вибирає найімовірніший варіант. У разі помилки передбачення конвеєр очищається від частково оброблених команд, що спричиняє затримки.

Для того, щоб програма могла працювати коректно із застосуванням конвеєра, конфліктні ситуації потрібно обробляти, тобто виявляти і вирішувати. Виявлення конфліктів виявляється непростим завданням, яке ми тут не розглядатимемо. Вирішення конфліктів може виконуватися у різний спосіб. Найпростішим способом є *«гальмування» конвеєра (Pipeline stall)* з метою затримки виконання наступної команди до вичерпання конфлікту. Але, по-перше, конфліктну ситуацію потрібно виявити, що непросто, а по-друге, затримки зменшують переваги конвеєра.

Низка підходів до обробки конфліктів представлена нижче.

1. Статичне перевпорядкування машинних команд під час компіляції програм з мов високого рівня в машинний код.

2. «Рознесення» команд, що конфліктують, при компіляції на безпечну відстань один від одного за допомогою вставки необхідної кількості спеціальної команди NOP (No Operation). Команда NOP нічого не робить, але сповільнює роботу програми на один такт, у деяких процесорах — і на більшу кількість тактів.

3. Динамічна обробка конфліктів під час виконання програми — ідентифікація та вирішення конфліктів виконується в момент виконання програми. При цьому процесор затримує виконання залежних команд (як,

наприклад, Intel 80486), а також самостійно переупорядковує команди, щоб унеможливити конфлікти з мінімізацією втрати часу (так діють процесор Intel Pentium та наступні процесори сімейств Intel x86).

Особливості обробки конфліктів у CISC- та RISC-архітектурах.

Розглянувши загальні підходи до обробки конфліктів на конвеєрах, зупинимося тепер на особливостях цієї процедури в CISC- та RISC-процесорах. Властивості машинної мови надають визначальний вплив те що, як можна організувати конвеєрне виконання машинного коду. Як наслідок, відмінності між CISC- та RISC-процесорами призводять до використання різних підходів до обробки конфліктів.

CISC-процесори наголошують на динамічній обробці конфліктів, оскільки в CISC-процесорах команди мають різну складність і час роботи, задіюють різні блоки процесора і звертаються до операцій різного виду. Крім того, в деяких сімействах динамічна обробка конфліктів є єдиним можливим рішенням, оскільки механізм конвеєризації в них з'явився не відразу і була потрібна сумісність з попередніми версіями процесорів, тобто механізм конвеєризації не є глибоко інтегрованим у процесор.

Розглянемо приклад. Першим конвеєризованим процесором сімейства Intel x86 був процесор Intel 80486. Для того щоб у нього була можливість коректно і по можливості швидко виконувати машинний код, призначений для попередніх версій неконвеєризованих процесорів, нічого не залишалося, крім «навчитися» вирішувати конфлікти динамічно. Ні на які «підказки» компілятора він при цьому не міг розраховувати, адже існуючий машинний код для попередніх процесорів сімейства Intel x86 про конвеєризацію нічого не знав.

Реалізація динамічної обробки конфліктів у CISC-процесорах потребує значного ускладнення конвеєра. Для порівняння можна уявити, як ускладнився б конвеєр на машинобудівному заводі, якби на ньому доводилося одночасно збирати бульдозери, комбайни та танки, до того ж різних моделей. Зазначимо, що іноді компілятори та низькорівневі програмісти все ж таки

допомагають CISC-процесорам, створюючи максимально «безконфліктний» машинний код. Це позитивно позначається на швидкості роботи програм на старих процесорах, які, наприклад Intel 80486, вміють при виявленні конфліктів лише затримувати виконання команд, але не переупорядковувати їх.

Машинний код RISC-процесорів, з погляду способу зберігання команд в оперативній пам'яті, виявляється простіше, ніж у процесорів CISC. Команди RISC-процесорів виконуються за однакову кількість тактів, а простіший машинний мову спрощує виявлення та обробку конфліктів. Деякі сучасні RISC-процесори (наприклад, процесори сімейства ARM), так само як і CISC-процесори, можуть виявляти конфлікти та переупорядковувати виконання команд. Інші ж RISC-процесори зовсім не обробляють конфлікти, наприклад, ранні версії процесорів сімейства MIPS — навіть сама назва цього сімейства розшифровується як Microprocessor without Interlocked Pipelined Stages, тобто мікропроцесор без блокувань на конвеєрі. Такі процесори вимагають, щоб підготовлений їм машинний код не містив потенційних конфліктів конвеєризації, і цю «безконфліктність» забезпечує компілятор.

Паралельне виконання команд

Як уже багаторазово вказувалося, підвищення продуктивності сучасних процесорів здійснюється значною мірою рахунок розпаралелювання. Одним з популярних способів розпаралелювання є конвеєризація. Але хоча конвеєризація дозволяє значно прискорити виконання програм, максимального ефекту вона досягає лише в ідеальних ситуаціях, коли конфлікти не виникають або коли вдається переупорядкувати команди таким чином, щоб ці конфлікти вичерпати. Для подальшого підвищення продуктивності застосовуються (іноді у поєднанні з конвеєризацією) складніші рішення.

Як першу техніку паралельного виконання команд процесором розглянемо *суперскалярність* — можливість одночасно виконувати кілька машинних команд за рахунок наявності в процесорі декількох однотипних

функціональних блоків (арифметико-логічних пристроїв, математичних співпроцесорів і т. д.) У сімействі Intel x86 першим процесором9 р). Цей процесор містив два арифметико-логічні пристрої, які дозволяли виконувати одночасно дві сусідні команди, якщо вони не залежали одна від одної. При цьому незалежні команди одночасно оброблялися двома різними конвеєрами. Для цього компілятори прагнули генерувати машинний код, сусідні команди якого не залежали б одна від одної.

Більш просунутою технікою паралельного виконання машинних команд є позачергове виконання (Out of Order Execution). Ця техніка дозволяє виконувати команди програми не в порядку прямування, а в порядку готовності до виконання. При використанні позачергового виконання кілька команд програми виконується тоді, коли для всіх цих команд готові потрібні їм вхідні дані. Першої ЕОМ, в якій було реалізовано механізм позачергового виконання команд, був суперкомп'ютер CDC 6600 компанії Cray Research, створений у 1963 році.

Наведемо такий приклад. Нехай у програмі поспіль йдуть такі команди:

```
(1) a = b / c
(2) d = b + a
(3) x = y * z
```

Поділ двох чисел (Команда 1) є складною командою, яка виконується суттєво довше, ніж, наприклад, додавання. Згадайте алгоритм поділу «в стовпчик»: у ньому послідовно виконується багато операцій та перевірок. Процесор виконує розподіл практично так само, але тільки в двійковій системі. Водночас від результатів Команди 1 залежить виконання Команди 2, а виконання Команди 3 не залежить. Тому можна розпочати виконання Команд 1 і 3 одночасно, а Команду 2 почати виконувати після обчислення значення a.

Розглянемо загальний сценарій позачергового виконання команд у сучасних процесорах на прикладі поширеного відомого процесора Intel Core i7. Після вибірки процесором чергової команди для виконання ця команда ділиться на мікрооперації – прості дії, такі як «скласти два числа», «записати

значення в регістр», «видати адресу осередку на шину адреси» та подібні до них. Мікрооперації поміщаються у спеціальний буфер переупорядкування (Reorder Buffer), у якому вони впорядковуються так, щоб, з одного боку, не порушити коректність програми, а з іншого боку, оптимально завантажити блоки процесора. Причому в буфері знаходяться і переупорядковуються одночасно мікрооперації кількох команд, що йдуть поспіль. Щоб отримати більше свободи для переупорядкування команд, процесор виконує перейменування регістрів: для різних мікрооперацій, що працюють з тими самими регістрами, призначаються нові регістри. Мікрооперації виконуються шістьма блоками процесора — трьома АЛП і трьома блоками доступу до пам'яті. При виявленні умовного переходу одночасно починають оброблятися обидві гілки then і else, тобто використовується спекулятивне виконання програми. Але після фактичного обчислення умови фіксуються лише результати команд із обраної у переході гілки.

Компанія Intel реалізувала позачергове виконання команд задовго до запуску у виробництво сучасних багатоядерних процесорів, у рамках процесорів Intel Pentium Pro на початку 2000-х років. Було помічено, що блоки процесора виконували мікрооперації ефективно, але часто простоювали, очікуючи на результати вибірки машинних команд з оперативної пам'яті. Для того, щоб оптимально завантажити ці блоки, було створено технологію Hyper-Threading. Вона вибирала машинні команди з оперативної пам'яті у двох паралельних потоках. Одноядерний процесор із застосуванням технології Hyper-Threading фактично починає працювати як двоядерний. Технологія Hyper-Threading була запатентована співробітником компанії Sun Microsystems у 1994 році, але вперше була реалізована лише у 2002 році у компанії Intel.

ТЕМА 2.5. КЕШ-ПАМ'ЯТЬ

План

1. Призначення та способи організації кеш-пам'яті.
2. Стратегії заміщення.
3. Принцип локальності.
4. Особливості організації кеш-пам'яті в багатоядерних процесорах з прикладу Intel Core i7.

Призначення та методи організації кеш-пам'яті

Як випливає з принципів архітектури фон Неймана, пам'ять ЕОМ неоднорідна і в ній є ієрархія пристроїв. Одним із елементів цієї ієрархії в сучасних комп'ютерах є кеш-пам'ять процесора.

Кеш-пам'ять процесора (Cache) *— це особливий вид швидкої пам'яті, що знаходиться в процесорі і призначається для зберігання даних, що активно використовуються при виконанні цієї програми в даний момент часу, що дозволяє суттєво прискорити роботу та знизити навантаження на оперативну пам'ять та системну шину.*

Розмір кеш-пам'яті значно менше і самої оперативної пам'яті, і її частини, що виділяється до роботи однієї програми. Оскільки ця пам'ять знаходиться всередині процесора, то обмеження на розміри кремнієвої пластини, на якій знаходиться процесор, і різні інші фактори не дозволяють зробити цю пам'ять досить великою. Тому в міру виконання програми дані в кеш-пам'яті постійно оновлюються (ця процедура називається заміщенням і буде розглянута нижче).

Фактично сучасні процесори не працюють безпосередньо з даними оперативної пам'яті, але попередньо завантажують в кеш-пам'ять. При цьому програма, що виконується процесором, не контролює роботу з кеш-пам'яттю і не знає про те, чи поміщені туди необхідні дані туди, тобто для неї кеш-пам'ять є прозорою. Cache у перекладі з англійської означає «схованку», що якраз відображає прозорість цього виду пам'яті для прикладної програми.

Обмін даними між оперативною пам'яттю та кеш-пам'яттю здійснюється блоками фіксованого розміру. Такий блок називатимемо *рядком кеш-пам'яті*. Розмір одного рядка та загальний обсяг є основними характеристиками кеш-пам'яті, що відрізняються для різних моделей процесорів. Наприклад, у процесорах Intel Core i7 розмір кеш-пам'яті варіюється від 4 до 12 Mb, а розмір рядка кеш-пам'яті становить 64 байти. За однакового загального розміру кеш-пам'ять, що складається з більшої кількості рядків, ефективніша, але складніша в реалізації.

Сучасні процесори мають багаторівневу кеш-пам'ять. Можна сказати, що в цьому випадку принцип ієрархічних пристроїв фон Неймана реалізується таким чином: рівні з меншими номерами кешують дані, що найчастіше використовуються, в рівнях з більшими номерами, і при цьому кожен наступний рівень має більший розмір і меншу швидкодію, ніж попередній.

Щоб для програми, що виконується, кеш-пам'ять була прозорою, використовується *механізм відображення кеш-пам'яті*. Цей механізм підтримує відповідність між даними в кеш-пам'яті та даними в оперативній пам'яті. Останнє важливо, оскільки в програмному коді, в операндах команд, які працюють не з регістрами, а з оперативною пам'яттю (наприклад, команда складання виду регістр/пам'ять), вказані адреси в оперативній пам'яті, а не в кеш-пам'яті, і звернення на ці адреси потрібно замінити на звернення до кеш-пам'яті. Цю заміну і реалізує механізм відображення.

Механізм відображення може бути асоціативним, прямим або гібридним.

• *Асоціативний механізм відображення* забезпечує можливість зв'язати кожен рядок кеш-пам'яті з будь-яким блоком оперативної пам'яті (див. рис. 21а). Це найбільш гнучкий механізм, але його апаратна реалізація є дорогою та споживає багато енергії. Це відбувається через те, що при зверненні процесора до оперативної пам'яті за певною адресою всі рядки кеш-пам'яті повинні опитуватися щодо того, чи не містять вони копію відповідного блоку оперативної пам'яті. Також при виділенні нового або звільнення зайнятого

рядка кеш-пам'яті процесор повинен підтримувати перелік вільних рядків, який потрібен для того, щоб при необхідності виділити черговий рядок і швидко знайти вільний, а не перевіряти всі рядки.

• *Прямий механізм відображення* дозволяє зв'язати з кожним блоком пам'яті лише один рядок кеш-пам'яті (див. мал. 21б). Пряме відображення значно дешевше в реалізації, ніж асоціативне, але воно водночас і менш ефективне.

• У багатьох сучасних процесорах, у тому числі в процесорах сімейства Intel x86, використовується *механізм гібридного відображення*: на один і той же блок оперативної пам'яті може відображатися не єдиний (як у прямому відображенні) і не довільний (як в асоціативному) рядок кеш-пам'яті, а кілька процесорів, часто — 4. Такий механізм значно гнучкіший, ніж пряме відображення, але при цьому не такий складний у реалізації, як механізм асоціативного відображення.

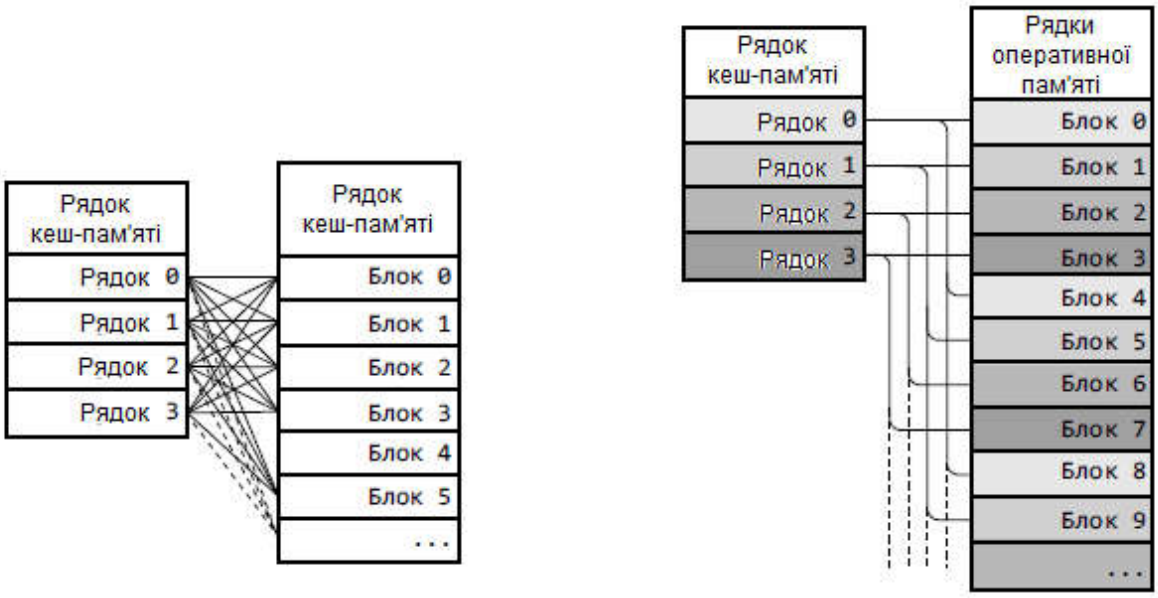

Рисунок 13 – Асоціативне (л) та пряме відображення (п) на прикладі кеш-пам'яті з чотирьох рядків

Стратегії заміщення та принцип локальності

Оскільки обсяг кеш-пам'яті процесора істотно менший за обсяг оперативної пам'яті, вже після нетривалої роботи програми кеш-пам'ять виявляється повністю зайнятою. При необхідності звернення до блоку

оперативної пам'яті, який не було поміщено до кеш-пам'яті, він переміщується туди. Але для цього потрібно вибрати рядок у кеш-пам'яті, в який потрібно позначити блок. Для цього процесор виконує *операцію заміщення*, що складається з наступних кроків:

• вибір відповідного рядка кеш-пам'яті для заміщення;

• збереження в оперативній пам'яті актуальних даних із цього рядка;

• копіювання в рядок відповідного блоку оперативної пам'яті.

Спосіб, яким у кеш-пам'яті вибирається рядок для заміщення, називається *стратегією заміщення*. При заміщенні зазвичай використовується принцип Беладі: слід замістити рядок, який містить дані, які не знадобляться процесору найдовше. Але однозначно вибрати такий рядок не можна, тому використовуються різні евристичні підходи. Найпоширенішою є стратегія *LRU (Least Recently Used)*, яка фіксує час останнього звернення до кожного рядка кеш-пам'яті та заміняє ту з них, до якої процесор найдовше не звертався. Пояснимо, як ця стратегія може бути реалізована залежно від механізму відображення, що використовується.

• Механізм асоціативного відображення дозволяє реалізувати стратегію LRU у тому вигляді, як вона описана вище.

• Для механізму прямого відображення вибір рядка для заміщення є тривіальним, оскільки з кожним блоком оперативної пам'яті може бути пов'язаний лише один рядок кеш-пам'яті, який слід замістити.

• Для механізму гібридного відображення стратегію LRU можна реалізувати, вибираючи з рядків, які можуть бути пов'язані з даним блоком оперативної пам'яті.

Незважаючи на те, що програма на C/C++, Java, Python тощо, неспроможна управляти кеш-пам'яттю процесора, розробки програм іноді її треба враховувати. Наприклад, при роботі з даними, що перевищують розмір кеш-пам'яті, слід дотримуватися принципу локальності.

Принцип локальності *стверджує, що дані, які обробляються в одному фрагменті програми, мають бути розташовані в оперативній пам'яті поруч.*

Наведемо приклад програми мовою С процесора Intel Core i7 (як було зазначено вище, кеш-пам'ять цього процесора становить від 4 до 12 Mb залежно від його моделі).

```
int my_array[10000][10000];

...

for(int i = 0; i < 10000; ++i)
  for (int j = 0; j < 10000; ++j)
    my_array[i][j] *= 2;
```

У цьому вся фрагменті оголошено двовимірний масив (матриця) цілих чисел *my_array* розміром 1000x1000. Ціле число (*int*) у сучасних компіляторах мови С представляється чотирма байтами. Відповідно, для масиву *my_array* потрібно 4*10000*10000 = 400 мегабайт. Очевидно, що цей масив не міститься в кеш-пам'яті процесора. Отже, у міру того, як програма працюватиме з цим масивом, в кеш-пам'ять процесора підвантажуватимуться з оперативної пам'яті все нові і нові фрагменти даного масиву. Далі двовимірні масиви мови З зберігаються в оперативній пам'яті рядково: *my_array[0][0],… my_array[0][9999], my_array[1][0], …, my_array[1][9999],….* Відзначимо, що саме в цьому порядку елементи масиву обробляються в циклі *for* у програмі, наведеній вище: тобто, коли в кеш-пам'ять з оперативної пам'яті завантажується рядок, програма повністю її обробляє і більше не звертається до цих даних. Потім кеш-пам'ять завантажується наступний рядок і так далі. Очевидно, що в цій програмі принцип локальності дотримується.

Змінимо цю програму так: замінимо оператор *my_array[i][j] *=2* на *my_array[j][i] *=2*. Тепер елементи масиву, що послідовно обробляються, розташовані в оперативній пам'яті далеко один від одного, і програма змушена постійно «стрибати» по масиву, що тягне багаторазове завантаження/вивантаження одних і тих же рядків кеш-пам'яті. У такій програмі принцип локальності не дотримується, і вона працюватиме повільніше, ніж попередня.

Кеш-пам'ять багатоядерних процесорів

Реалізація кеш-пам'яті багатоядерних процесорів, та і в цілому обчислювальних пристроїв з апаратною підтримкою багатозадачності, пов'язана з низкою труднощів. Одна з найбільш значних труднощів викликана необхідністю забезпечити *когерентність кеш-пам'яті.*

Когерентність *— властивість багатоядерного процесора, що передбачає узгодженість даних у кеш-пам'яті його різних ядер під час виконання програм.*

Розглянемо приклад збою, який може статися при використанні кеш-пам'яті, що не володіє властивістю когерентності. Припустимо, що в програмі на мові С є рядок *int x,y* . В оперативній пам'яті ці змінні розміщуються поруч і потрапляють в один блок, який передається в кеш-пам'ять і потім, через деякий час, «повертається» до оперативної пам'яті з кеш-пам'яті. Далі припустимо, що у нас є двоядерний процесор і кожне з ядер змінює одну з цих змінних, ядро 1 змінює змінну *x* а ядро 2 змінює змінну *y* (це може відбуватися, наприклад, у різних потоках). Таким чином, процесор виконує дії, наведені нижче.

1. Ядро 1 читає значення змінної x, записуючи до своєї кеш-пам'ять відповідний блок оперативної пам'яті.

2. Ядро 2 читає значення змінної, записуючи у свою кеш-пам'ять той самий блок оперативної пам'яті (нагадуємо, що обидві змінні перебувають поруч, а кеш-пам'ять і оперативна пам'ять що неспроможна обмінюватися фрагментами пам'яті, відповідними лише однієї змінної, і захоплюють зайве.

3. Ядро 1 записує нове значення у змінну у своїй кеш-пам'яті.

4. Ядро 2 записує нове значення у змінну у своїй кеш-пам'яті.

5. В оперативну пам'ять записується рядок кеш-пам'яті ядра 1. Цей рядок містить як змінене значення змінної, так і колишнє значення змінної.

6. В оперативну пам'ять записується рядок кеш-пам'яті ядра 2; цей рядок містить як старе значення змінної , так і змінене значення змінної .

Очевидно, що на кроці 6 значення змінної виявиться незмінним, яке було до того, як ядро 1 змінило цю змінну. Таким чином, результати дій ядра 1 щодо зміни змінної виявляться втраченими, що є помилкою.

Для того щоб уникнути таких проблем, зберігши при цьому ефективність роботи кеш-пам'яті, в багатоядерних процесорах доводиться застосовувати складні технічні рішення.

Як приклад розглянемо процесор Intel Core i7 (рис. 14). Цей процесор має трирівневу кеш-пам'ять. Всередині кожного ядра знаходиться кеш-пам'ять першого рівня (L1); вона є найшвидшою. Для підвищення продуктивності кеш-пам'ять рівня L1 організована за принципом гарвардської архітектури: машинний код і дані зберігаються окремо. Кеш-пам'ять другого рівня (L2) має більший об'єм та меншу швидку дію, в ній код і дані зберігаються разом. Кожне процесорне ядро має свою кеш-пам'ять другого рівня, а інші ядра з нею працювати не можуть. Нарешті, кеш-пам'ять третього рівня (L3) є спільною для всіх ядер, і вона виявляється найбільшою, але й найповільнішою. За допомогою кеш-пам'яті рівня L3 процесор звертається до системної шини доступу до оперативної пам'яті.

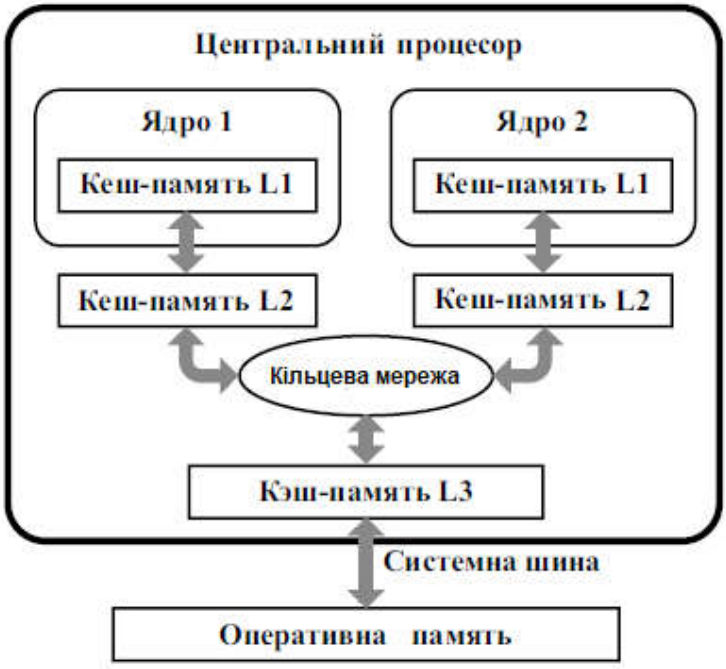

Рисунок 14 - Організація кеш-пам'яті у двоядерному процесорі Intel Core i7

Для забезпечення когерентності всередині процесора Intel Core i7 використовується внутрішня шина спеціальної конструкції, яка називається кільцевою мережею. внутрішньої кеш-пам'яті L1, цей запит обробляє його кеш L2. кеш-пам'яті L2 ядра 1 необхідні дані знайшлися, то вона обробляє запит і повертає дані в кеш-пам'ять L2 ядра 0. Якщо ні в одному з кешів L2 не знайшлося необхідних даних, запит повертається необробленим ядру 1, і кільцева мережа в промаху звертається вже до оперативної пам'яті).. Як тільки якесь з ядер вносить зміни до кеш-даних, кільцева мережа відправляє іншим ядрам повідомлення про те, що копії цих даних у їх кеш-пам'яті втратили актуальність і в разі подальших звернень їх треба звернутися до різних ядра. потоків кільцева мережа внесе наступні зміни в дії процесора (у цьому прикладі ми описуємо, як це відбувається в процесорі Intel Core i7).

• На кроці 3 ядро 1 кільцевої мережі повідомляє, що всі копії блоку пам'яті, за винятком тієї, яка знаходиться в кеш-пам'яті ядра 1, стають неактуальними.

• На кроці 4, перш ніж змінювати значення змінної y, ядро 2 запитує актуальні дані рядка кеш-пам'яті у ядра 1, потім змінює значення змінної y. А після зміни повідомляє ядру 1, що актуальна копія тепер має.

• Кроків 5 і 6 не буде, оскільки ядра процесора Intel Core i7 не взаємодіють з оперативною пам'яттю самостійно. Натомість рано чи пізно актуальна копія блоку пам'яті потрапить з кеш-пам'яті ядра 2 в загальну кеш-пам'ять L3, а ще пізніше з L3 - в оперативну пам'ять.

ТЕМА 2.6. АДРЕСНИЙ ПРОСТІР ТА ВІРТУАЛЬНА ПАМ'ЯТЬ

План

1. Фізична адреса, пряма адресація, адресний простір програми.
2. Сегментна адресація: сегменти, їх призначення, віртуальна адреса, адресне перетворення.
3. Віртуальна пам'ять: сторінкова адресація, сторінки, сторінкове адресне перетворення та їх використання при реалізації віртуальної пам'яті.
4. Заміщення сторінок фізичної пам'яті, виділення сторінок за необхідності.

Фізична адреса та пряма адресація

Під час старту програми операційна система створює для неї спеціальний процес. Можна сказати, що процес є екземпляром програми, точніше запущеним на комп'ютері екземпляром. Як уже згадувалося вище, одночасно на комп'ютері може бути запущено кілька екземплярів однієї програми, з чим ви напевно стикалися — наприклад, працюючи у Windows, легко запустити кілька екземплярів текстового редактора «Блокнот». Далі ми іноді використовуватимемо терміни «процес» і «програма» як синоніми, вважаючи, що з контексту зрозуміло, про що йдеться.

В оперативну пам'ять створеного процесу копіюється код відповідної програми, а також виділяється пам'ять даних програми. Процесор взаємодіє з цією пам'яттю у вигляді адрес. Система адресації у сучасних комп'ютерів влаштована складно, і ми почнемо вивчати її з поняття фізичної адреси.

Фізична адреса (Physical Address) *— це номер осередку в оперативній пам'яті. Знаючи цей номер, процесор безпосередньо звертається до потрібного місця оперативної пам'яті за необхідними даними.*

Коли ми говоримо про адресовану комірку в оперативній пам'яті, то ми маємо на увазі машинне слово, яке, як розповідалося в попередніх лекціях, є фрагментом пам'яті, що мінімально адресується.

Багато ЕОМ минулих поколінь, а також прості сучасні обчислювальні пристрої дозволяють своїм програмам звертатися до оперативної пам'яті безпосередньо за фізичною адресою, минаючи всілякі адресні перетворення, про які йтиметься в цій лекції. Така організація роботи з пам'яттю називається *прямою адресацією.*

Пряма адресація *– це механізм використання в машинному коді програм реальних фізичних адрес в оперативній пам'яті без застосування адресних перетворень.*

Хоча дана схема на перший погляд виглядає природною, вона не має достатньої гнучкості - адже насправді не все так просто! Тому в сучасних персональних комп'ютерах, смартфонах, планшетах та інших складних обчислювальних пристроях пряма адресація не застосовується, тому ми маємо про що поговорити в рамках даної лекції.

Дамо тепер визначення адресного простору програми. Здавалося б, це вся оперативна пам'ять, що виділяється процесу. Однак це не так, оскільки програма може оперувати з оперативною пам'яттю більшого обсягу, ніж виділено її процесу. Для цього використовується механізм віртуальної пам'яті, про який йтиметься трохи нижче. А тут ми дамо коректне визначення адресного простору програми.

Адресний простір (Address Space) *- це безліч адрес, допустиме для використання програмою.*

Зокрема, значення вказівників у програмах мовою С, що працюють на комп'ютерах сімейства процесорів Intel x86, є елементами адресного простору.

Сегментна адресація

Пам'ять, що виділяється процесу при старті операційної системи, є набором сегментів.

Сегмент *— це фрагмент оперативної пам'яті, що виділяється операційною системою процесу.*

Розмір та конкретне розташування сегментів процесу операційна система вибирає, виходячи з наступних параметрів:

• розмір машинного коду програми;

• розмір даних;

• оптимізація розподілу пам'яті цього процесу з точки зору інших запущених процесів, а також можливо процесів, які можуть бути запущені в найближчому майбутньому.

Розглянемо класичну (спрощену) схему пам'яті процесу, коли ця пам'ять складається з сегмента коду, сегмента даних та сегмента стека. Адреси всередині сегментів формуються як усунення щодо початку сегмента. Код програми знає про сегменти та оперує цими зміщеннями. Наприклад, в операції складання виду регістр/пам'ять адресу в пам'яті для другого операнда вказується у вигляді усунення від початку сегмента даних (нагадаємо, що там зберігаються всі змінні програми). Таким чином, реальна фізична адреса, за якою має звернутися процесор, обчислюється вже всередині процесора, а остання, у свою чергу, знає фізичні адреси початку кожного сегменту. Отже, ми підійшли до поняття віртуальної адреси та адресного перетворення.

Адреса, з якою безпосередньо працює програма, називається ***віртуальною адресою (Virtual Address).***

Адресне перетворення (Address Conversion) *— це механізм очищення фізичної адреси за віртуальною.*

Зазначимо, що сегментна організація пам'яті процесу не єдиним джерелом віртуальних адрес.

Розглянемо тепер детальніше сегменти процесу.

Сегмент коду зберігає в оперативній пам'яті машинний код програми.

Сегмент даних зберігає глобальні змінні програми (так звані *статичні дані*), а також використовується для купи (Heap) у тому випадку, якщо в програмі використовуються *динамічні дані*. Розмір всього сегмента даних відомий під час запуску програми, але пам'ять для купи виділяється в ньому в міру виконання програми, оскільки на момент запуску незрозуміло, скільки програмі знадобиться динамічної пам'яті — це залежить від конкретного сценарію виконання програми, оскільки динамічна пам'ять запитується з рівня

прикладної програми та її використання організує сам програміст — автор програми. Відповідно, купа є «динамічною бульбашкою» нефіксованого розміру, що «роздується» у міру запитів програми.

У програмах мовою С для того, щоб запросити динамічну пам'ять, програміст використовує функцію *malloc* , а C++ — оператор *new*. Оскільки динамічна пам'ять виділяється в «ручному» режимі, то й звільнятися вона повинна також «вручну» за допомогою спеціальних функцій (функція *free* в С, оператор *delete* в C++). Механізм динамічної пам'яті вимагає від програміста уважності та організованості, але надає гнучкість і можливість створювати більш ефективні програми. Однак при використанні цього механізму програміст може допустити (і припускає!) помилки, що складно виявляються. Це спричинило те, що в сучасних промислових мовах, таких як Java, C#, Python, використання динамічної пам'яті доступне лише через виділення об'єктів, а не масивів байтів, як у С, а звільнення — за допомогою автоматичного складання «сміття», що звільняє пам'ять об'єктів, на які більше немає посилань із програми.

Сегмент стека використовується процесом для організації викликів процедур (методів) програми. Він реалізує структуру даних під назвою *стек*.

Набір невеликих однотипних фрагментів пам'яті, об'єднаних у список, в якому новий елемент завжди додається тільки в кінець списку і видалятися з цього списку може з кінця, називається ***стеком***.

У той момент, коли у програмі виконується виклик чергової процедури, в сегмент стека поміщаються такі дані:

• фактичні параметри цієї процедури;

• адресу повернення з цієї процедури, тобто адресу інструкції у програмі, яка настає за викликом процедури;

• локальні змінні процедури та додаткова службова інформація.

Сукупність цих даних утворює так званий кадр стека (*Stack frame*). Очевидно, що після завершення процедури та продовження виконання програми у тому контексті, звідки процедура була викликана, ні її локальні

дані, ні параметри більше не потрібні. Тому кадр завершеної процедури видаляється зі стека, звільняючи місце для інших даних. Якщо процедура викликається рекурсивно, її кадри з різними значеннями параметрів, даних, адреси повернення розміщуються на стеку стільки разів, скільки разів її викликали. Якщо програма зациклюється на рекурсивному виклику, вона поміщає на стек дедалі більше даних, заповнюючи всю допустиму для стека пам'ять, і після цього аварійно завершується з помилкою «переповнення стека» (Stack Overflow). Ця помилка настільки відома і поширена, що на честь її названо найпопулярнішу платформу запитань-відповідей щодо програмування Stack Overfl ow (https://stackoverflow.com/).

Зробимо наступну ремарку. Програма може бути багатопотоковою, тобто всередині одного процесу може працювати кілька паралельних потоків. Фізично ці потоки зазвичай розміщуються на різних ядрах багатоядерного процесора. При цьому різні потоки всередині одного процесу мають різні сегменти стека, оскільки стек використовується для підтримки дзвінків та роботи процедур, а в різних потоках одночасно можуть бути запущені різні процедури. Але сегменти коду та сегменти даних у потоків одній програми є загальними, оскільки у сегменті коду зберігається весь машинний код програми та всі потоки працюють (можуть працювати) з усіма даними програми.

Віртуальна пам'ять і сторінкова адресація

Ранні ЕОМ використовувалися переважно для наукових розрахунків. Вони мали невелику оперативну пам'ять, але при цьому потрібно було, щоб вони обробляли дані, обсяг яких суттєво перевищував розмір оперативної пам'яті, а також вміли виконувати програми, обсяг коду яких перевищував обсяг оперативної пам'яті. Для цього використовувався спеціальний прийом, що дозволяє завантажувати фрагменти даних та фрагменти машинного коду в оперативну пам'ять порціями, у міру потреби. Це дуже ускладнювало код прикладних програм – програмісту доводилося самостійно реалізовувати цю функціональність. Пізніше творці ЕОМ другого та третього поколінь

виробили загальні підходи до вирішення цього завдання та винесли її рішення в операційну систему. Так виникла віртуальна пам'ять.

Віртуальна пам'ять (Virtual Memory) *— це механізм роботи програми/процесу з адресним простором, що перевершує за розміром оперативну пам'ять ЕОМ або той її фрагмент, який виділений даній програмі/процесу.*

При цьому програма/процес може працювати з фрагментами різнорідної фізичної пам'яті (включаючи жорсткий диск) єдиний, суцільний фрагмент і забезпечуючи тим самим безперервний адресний простір.

Адресний простір, доступний процесу, ділиться на *віртуальні сторінки*. Для процесорів сімейства Intel x86, що працюють у 32-розрядному режимі, розмір однієї сторінки становить 4 Kb. Ці ж процесори, що працюють у 64-розрядному режимі, підтримують розмір сторінок від 4 Kb до 1 Mb. При цьому оперативна пам'ять ділиться на *фізичні сторінки*, розмір яких збігається із розміром віртуальних сторінок.

Операційна система виконує *сторінкове адресне перетворення* для того, щоб пов'язувати кожну віртуальну сторінку з реальною пам'яттю:

• або з фізичною сторінкою в оперативній пам'яті;

• або з областю підкачки на жорсткому диску (у випадку Windows це файл, в Unix-подібних ОС - розділ жорсткого диска).

Можливо, що віртуальна сторінка не пов'язана з жодним реальним (фізичним) фрагментом пам'яті. Це означає, що ця сторінка не визначена, тобто до неї ще не було звернень із програми. Також сторінка може перейти в цей стан, якщо програмі більше не потрібні дані, що знаходяться в ній, і вона явно «повернула» цю сторінку операційній системі.

Відповідність між сторінками віртуальної пам'яті та фізичною пам'яттю зберігається в спеціальній структурі даних, яка називається таблицею сторінок і розташовується в оперативній пам'яті. Таблиця сторінок для всього адресного простору вийшла б неприпустимо великою, якщо зберігати її як масиву. Наприклад, легко порахувати, що для адресного простору 32-бітного

процесора Intel 80386 таблиця сторінок зайняла б до 4 Mb, тоді як загальний обсяг оперативної пам'яті комп'ютерів на базі цього процесора зазвичай дорівнював 2 або 4 Mb. Тому в 32-розрядному режимі Intel x86 таблиця сторінок зберігається у вигляді двох рівневих деревоподібних структур, а в 64-розрядному режимі Intel x86_64 — у вигляді чотирирівневої. Ці структури заповнюються поступово, у міру використання програмою віртуальних сторінок.

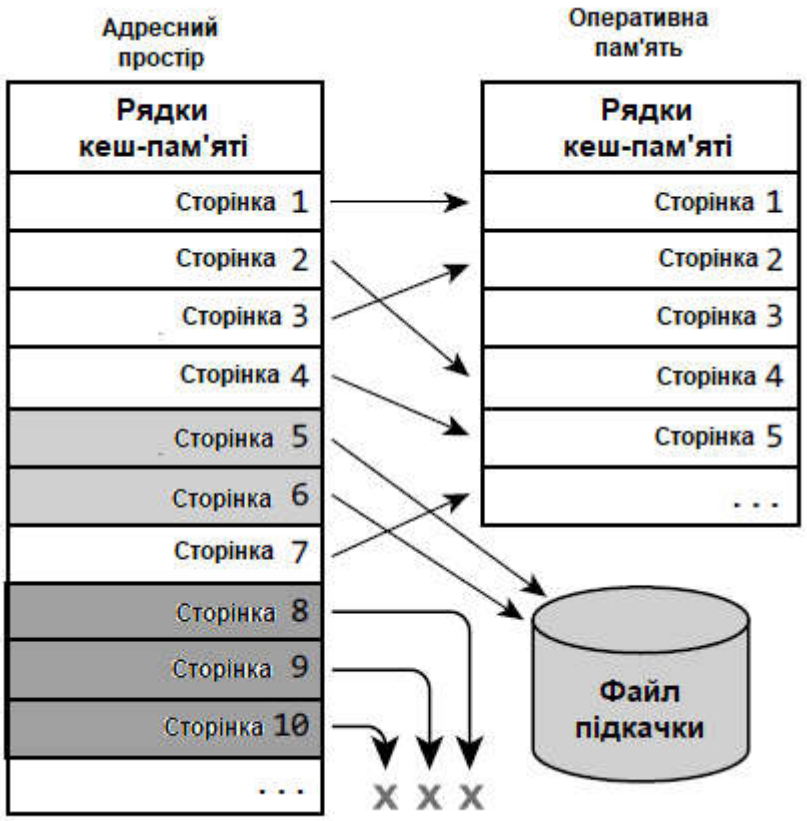

Рисунок 15 - Сторінка адресація та віртуальна пам'ять

На рис. 15 представлений механізм сторінкової віртуальної пам'яті. Зазначимо, що операційна система може для кожного процесу створити окрему таблицю сторінок, виключивши, таким чином, оперативну пам'ять інших процесів з його адресного простору. Це дозволяє посилити механізм ізоляції процесів.

У сімействі процесорів Intel x86, починаючи з Intel 80386, сегментна та сторінкова адресація можуть використовуватись одночасно. Спочатку

процесор робить сегментне перетворення, потім над адресою, що вийшла, він виконує сторінкове перетворення, результатом якого вже є фізична адреса.

Виробники операційних систем для сучасних процесорів (наприклад, Intel x86_64, ARM v7) поступово відмовляються від використання сегментної адресації, задовольняючись сторінковими можливостями. Застосування одночасно сегментного та сторінкового адресних перетворень, як у Intel 80386, є надмірним: віртуальна пам'ять будується тільки на основі сторінкового адресного перетворення. Процесори сімейства Intel x86 у 64-бітному режимі не використовують сегменти, підтримуючи сегментну адресацію лише для сумісності зі старими програмами. Багато архітектур, відмінних від Intel x86, сегментну адресацію не підтримують зовсім.

Виділення та заміщення віртуальних сторінок

Доки програма не звернулася до пам'яті, віртуальні сторінки її процесу не визначені. При першому зверненні програми до чергової сторінки процесор зупиняє роботу програми та передає керування операційною системою. Остання виділяє для віртуальної сторінки фізичну сторінку в оперативній пам'яті та оновлює таблицю сторінок. Якщо вільних фізичних сторінок немає, то операційна система вивантажує дані з деякої зайнятої фізичної сторінки в область підкачки на жорсткому диску і виділяє процесу звільнену сторінку.

Для вибору сторінки, що вивантажується, використовуються *стратегії витіснення*, аналогічні стратегіям витіснення для кеш-пам'яті. Витіснена віртуальна сторінка позначається як вивантажена, і при подальшому зверненні до неї процесор також передає керування операційною системою, щоб вона могла знову виділити для цієї сторінки фізичну сторінку. Такий порядок роботи віртуальної пам'яті дозволяє процесу використовувати велику кількість пам'яті, але реальну фізичну пам'ять отримувати лише в міру звернення фрагментів цієї пам'яті. При запуску процес може запросити в операційної системи багато пам'яті, але доки він почав з нею працювати, фізична пам'ять нічого очікувати витрачатися.

Можна легко переконатися в тому, що сучасні операційні системи виділяють пам'ять описаним чином, написавши наступну просту програму на C. Дана програма динамічно, за допомогою функції , запитує великий обсяг пам'яті (наприклад, 1 Gb), але звертається до цієї пам'яті не відразу, а через деякий час. Якщо простежити за запущеною програмою (у разі Windows для цього достатньо диспетчера процесів), то стане очевидним, що реальна фізична пам'ять виділяється не в момент виклику *malloc*, а в міру звернення до пам'яті з програми.

ТЕМА 2.7. ВВЕДЕННЯ ДО ОПЕРАЦІЙНИХ СИСТЕМ

План

1. Призначення операційної системи.

2. Основні функції операційних систем: керування пристроями, ізоляція процесів, планувальник завдань, віртуальна пам'ять, розмежування прав доступу користувачів, файлова система, мережна підтримка, інтерфейс користувача, керування живленням.

Призначення операційної системи

Сучасний комп'ютер включає безліч різних пристроїв — процесор, пам'ять, жорсткий диск тощо. До комп'ютера також можуть підключатися зовнішні (периферійні) пристрої, наприклад, принтер або сканер. Комп'ютер повинен вміти керувати цими пристроями для того, щоб прикладні програми, які на неї працюють, отримали можливість користуватися сервісами цих пристроїв, наприклад, щоб можна було з MS Word друкувати документи на принтері.

Управління зовнішніми та внутрішніми пристроями здійснюють спеціальні службові програми. Ці програми досить складні, оскільки взаємодіють із пристроями через низькорівневі інтерфейси. Більш того, для кожного пристрою існує велика кількість аналогів — наприклад, кількість моделей принтерів від різних виробників вимірюється сотнями. І кожна модель має власні особливості, що відображаються в її програмному інтерфейсі. Однак важливо, щоб прикладне програмне забезпечення не залежало від того, який конкретний пристрій підключено або встановлено на цьому комп'ютері, тобто функція друку в Microsoft Word повинна однаково працювати незалежно від виду принтера, підключеного до цього комп'ютера. Управління пристроями та приховування від прикладних програм особливостей різних моделей цих пристроїв виконує *операційна система.*

Операційна система (ОС, Operating System) *— це службове ПЗ, яке дозволяє прикладним програмам, що працюють на комп'ютері, ефективно*

використовувати як внутрішні пристрої комп'ютера — центральний процесор, оперативну пам'ять, жорсткий диск, засоби мережевої підтримки тощо, так і зовнішні — наприклад, монітор, клавіатуру, принтер.

Ідея незалежності прикладного програмного забезпечення від обладнання комп'ютера з'явилася давно, як мінімум, на початку 1950-х років. Проте поняття операційної системи увійшло широкого побуту в епоху ЕОМ другого покоління, на рубежі 1950-х і 1960-х років. З цього часу операційні системи пройшли довгий шлях розвитку. На даний момент створено велику кількість різноманітних операційних систем. Можна згадати різні версії Windows, які використовуються переважно на офісних комп'ютерах і ноутбуках, ОС Linux, яка ефективно використовується на серверах і телекомунікаціях, і ОС Android, що працює на мобільних телефонах і планшетах.

Основні функції операційних систем

Перейдемо тепер до детального розгляду основних функцій операційних систем. Спочатку перерахуємо їх.

• Керування пристроями (Device Management).

• Багатозадачність (Multitasking).

• Управління віртуальною пам'яттю (Virtual Memory Management).

• Розмежування прав доступу користувачів (Access Rights).

• Файлова система (File System).

• Підтримка мережі (Network support).

• Інтерфейс користувача (User Interface).

• Управління живленням (Power Management).

Керування пристроями. Історично ця функція є основною для операційної системи.

*Різні пристрої комп'ютера взаємодіють з прикладним програмним забезпеченням, що працює на цьому комп'ютері, за допомогою спеціальних програм — **драйверів (Drivers)**; останні є частиною операційної системи.*

Операційна система включає велику кількість драйверів для найбільш популярних моделей пристроїв, проте не може покрити всіх можливих варіантів. Ось чому після покупки, наприклад, нового принтера, вам може знадобитися знайти в Інтернеті та встановити на вашому комп'ютері відповідний драйвер.

Багатозадачність. На сучасному комп'ютері миттєво може виконуватися більше однієї програми. Підтримка цієї можливості називається багатозадачністю. Для забезпечення багатозадачності необхідно, щоб процесор комп'ютера виконував програми відповідно до деякого розкладу — спочатку одну програму, потім іншу, потім третю тощо, повертаючись через деякий час до першої, і так по колу. Це підхід, призначений спочатку для одноядерних процесорів, називається *псевдопаралелізмом*, оскільки він має на увазі не паралельну роботу кількох програм на одному процесорі, а виконання програм послідовно, але з частим перемиканням. Псевдопаралелізм використовується і для багатоядерних процесорів, оскільки в них одночасно працює більше програм, ніж ядер у процесора.

Існує багато стратегій реалізації псевдопаралелізму. У операційній системі Windows запущені на даний момент на комп'ютері програми (процеси) можна побачити в Task Manager. Для того, щоб мінімізувати наслідки впливу помилок в одній програмі на інші програми, запущені на тому ж комп'ютері, ОС запускає кожен процес в ізольованому оточенні. Це означає, що запущені процеси не мають доступу до даних один одного. Один із основних механізмів такої ізоляції — режим адресації, при якому процес може адресувати лише свої дані, і за однаковими адресами, зазначеними у відповідних програмах, різні процеси мають різні дані, кожен — свої.

Управління віртуальною пам'яттю. Запущені на комп'ютері програми можуть вимагати більше оперативної пам'яті, ніж наявні або чим ОС може виділити для цієї програми. У такій ситуації ОС використовує віртуальну пам'ять, організуючи для програми з використанням жорсткого диска більшу пам'ять. Ця ілюзія для прикладної програми є достовірною — остання не

вникає в деталі розподілу пам'яті, просто використовуючи всю ту пам'ять, яку їй виділяє ОС, хоча, безумовно, активне перевищення реальних, фізичних лімітів (тобто інтенсивне використання жорсткого диска через механізм віртуальної пам'яті) позначається на виробнику.

Файлова система. Жорсткий диск комп'ютера є складним пристроєм, але він зберігає лише масив байтів. Однак безпосередня робота з масивом байт вкрай незручна як кінцевого користувача, так прикладних програм. У зв'язку з цим ОС підтримують абстракцію файлу.

Файл (File) *— це фрагмент пам'яті на жорсткому диску, який має ім'я, тип, дату створення та іншу службову інформацію. Файл є елементом збереження інформації на жорсткому диску. Більшість підсистем операційної системи, прикладні програми та користувачі мають справу саме з файлами.*

Різні види файлів - текстові, аудіо / відео, docx і т.д. буд. — операційна система та прикладні програми обробляють по-різному.

Для операційної системи особливо важливими є файли, що виконуються. Формат файлів, що виконуються в різних операційних системах, різний. Для Windows основними виконуваними файлами є:

• exe-файли (мають розширення exe) - самостійні програми машинною мовою, які Windows завантажує з жорсткого диска в оперативну пам'ять і передає на виконання процесору;

• dll-файли (мають розширення dll) подібно до exe-файлів містять код машинною мовою і точно також виконуються, але не є самостійними програмами: для реалізації модульності програм, що дозволяє компонувати код у різні файли, що виконуються (для ефективного виконання, а також, багато в чому, для зручностей розробки).

Файлова система підтримує функції збереження файлів на жорсткому диску та доступу до них як для програм, так і для безпосередніх користувачів комп'ютера. Вона ж підтримує ієрархію папок. Для операційної системи Windows загальним файловими системами є FAT і NTFS.

• Історично FAT (File Allocation Table) була першою файловою системою для IBM PC і призначалася для роботи з гнучкими та жорсткими дисками. В даний час ця файлова система витісняється сучаснішими файловими, але для сумісності зі старими ОС вона досі часто використовується на знімних накопичувачах - "флешках" і зовнішніх жорстких дисках. Можна зіткнутися з наступним обмеженням FAT: ця файлова система не дозволяє працювати з файлами, що мають розмір більше 4 Gb, а також вона не підтримує розмежування прав доступу. Заснована на FAT сучасна файлова система exFAT дозволяє працювати з файлами розміром більше 4 Gb.

• NTFS (New Technology File System) з'явилася в Windows NT і використовується зараз в операційних системах сімейства Windows як основна. NTFS підтримує розмежування прав доступу до файлів та папок, а також стиснення та шифрування файлів. Файлова система NTFS також може відновлюватися після збоїв. Наприклад, якщо з папки було видалено файл, але через раптове відключення електроживлення звільнене на жорсткому диску місце не було позначене як вільне, то при відновленні живлення неможливо знайти цей файл або використовувати відповідне місце. NTFS вміє коректно вирішувати цю та подібні до неї проблеми.

Розмежування прав доступу користувачів. У випадку, коли одним комп'ютером можуть користуватися різні люди, потрібно забезпечити розмежування доступу до ресурсів і даних цього комп'ютера — файлів і каталогів, тими чи іншими додатками (програмами), встановленими на комп'ютері, принтерами і т. д. Частина цих можливостей може бути доступна всім користувачам даного комп'ютера. користувачі, яких він вибере. Це завдання є актуальним і для локальних мереж, адміністрування яких також здійснюється засобами операційної системи.

Для вирішення завдання розмежування прав доступу кожному користувачеві комп'ютера створюється *обліковий запис (Account)*. Крім облікових записів для «справжніх» користувачів (людей), в ОС часто є кілька

службових облікових записів, «від імені» яких запускаються службові програми — засоби резервного копіювання даних, веб-сервера, сервера баз даних і т. д. Захист цих даних забезпечується механізмом розмежування прав не може їх навмисне чи випадково змінити. Наприклад, звичайний користувач Windows не може безпосередньо через Провідник записувати та видаляти файли у системних папках, але може робити це лише через інсталяцію або деінсталяцію нових додатків, тобто за допомогою спеціальних пакетних процедур. У момент роботи таких процедур і активізуються службові облікові записи.

Мережева підтримка забезпечує можливість підключення комп'ютера до локальних та глобальних мереж зв'язку (сьогодні це в основному Internet, для мобільних ОС – мобільні мережі). Мережева робота ведеться з використанням складної багаторівневої системи мережевих протоколів, що реалізують передачу пакетів і відстеження їх цілісності, обробку різних мережевих помилок тощо. Мається на увазі, що з боку мережі аналогічні стеки реалізуються асамблеями різних мережевих пристроїв - роутерів, фаєрволів, концентраторів та ін.

Інтерфейс користувача - це підсистема ОС, призначена для взаємодії з кінцевим користувачем, яка надає йому можливість використовувати різні ресурси комп'ютера. Настільні та мобільні ОС (Windows, Mac OS X, настільні версії Linux, iOS, Android) надають користувачам графічний інтерфейс, використовуючи метафору Робочого столу: комп'ютер імітує на екрані різні «офісні» предмети — папки, картотеки, документи і т. д. Значна увага в останні можливостями. Так, наприклад, багато сучасних настільних ОС підтримують голосове введення та озвучку тексту на екрані. Для серверних ОС традційним інтерфейсом користувача є текстова консоль - підсистема ОС, що дозволяє вводити з клавіатури дані та команди і відображати результати на моніторі в текстовому вигляді *(Історично термін «консоль» позначав пристрій, що поєднує функції введення та виведення даних та керуючих команд комп'ютера, наприклад комплект з клавіатури та монітора. До кінця*

1970-х років. пристрої введення та виведення часто випускалися в загальному корпусі, що і спричинило виникнення терміну. До широкого поширення моніторів як консолі використовувалися телетайпи (клавіатура та принтер, які могли підключатися до ліній зв'язку)). Консоль забезпечує діалог користувача з командною оболонкою ОС, що працює на сервері, дозволяючи керувати СУБД, веб-серверами тощо. Також консоль дозволяє запускати і інтерактивні програми — наприклад, текстові редактори для редагування системних налаштувань.

Система керування живленням особливо актуальна для мобільних ОС, оскільки ноутбуки, смартфони та планшети працюють на акумуляторах. У цих пристроях важливо забезпечити оптимальну витрату живлення, щоб комп'ютер міг працювати максимально довго без підключення до електричної мережі. Також важлива підтримка контролю циклу заряджання/розрядки акумуляторів для продовження їх терміну служби.

ТЕМА 2.8. ОГЛЯД ВІДОМИХ ОПЕРАЦІЙНИХ СИСТЕМ

План

1. Операційні системи сімейства мейнфреймів IBM.
2. Unix - ОС, що переноситься.
3. Linux ОС, що вільно розповсюджується (Open Source).
4. DOS і Windows - ОС для комп'ютерів сімейства IBM PC.
5. Android — найпопулярніша мобільна ОС.

Сучасні комп'ютери активно розвиваються, відповідно, змінюються і операційні системи, оскільки вони безпосередньо пов'язані з апаратурою та мають її ефективно обговорювати. При цьому і комп'ютерні архітектури, і відповідні операційні системи утворюють сімейства. Наприклад, Windows призначаються для комп'ютерів Intel x86.

Крім того, ОС для нових комп'ютерів часто створюються на основі вже існуючих: наприклад, поширена ОС для мобільних пристроїв Android була створена на основі Linux.

У цій лекції ми розглянемо ряд відомих сімейств операційних систем, частиною з колишніх часів і вже історією, частиною сучасних і активно розвиваються:

• сімейство операційних систем для менфреймів компанії IBM;

• сімейство переносних операційних систем Unix;

• сімейство операційних систем Linux, що вільно розповсюджуються;

• сімейство операційних систем DOS для комп'ютерів IBM PC;

• сімейство операційних систем Windows для процесорів сімейства Intel x86;

• сімейство мобільних операційних систем Android.

Операційні системи сімейства мейнфреймів IBM

Мейнфрейм (Mainframe) — це високопродуктивний сервер, орієнтований на обробку великої кількості даних та запитів, тобто для задоволення потреб великого бізнесу. Мейнфрейм не слід плутати з

суперкомп'ютером, який орієнтований на високопродуктивні пакетні обчислення (інженерні та наукові завдання). Історія мейнфреймів почалася наприкінці 1960-х років із випуску компанією IBM відомого комп'ютера System/360. Мейнфрейми випускалися й іншими виробниками, наприклад, Burroughs та General Electric. Слід зазначити, що мейнфрейми використовуються та випускаються досі.

Мейнфрейми компанії IBM 1960-1980-х років працювали з великою кількістю терміналів (робочих місць): їх кількість сягала кількох десятків тисяч. Основними покупцями таких комп'ютерів були різні установи — від університетів та дослідницьких інституцій до великих компаній та банків. Відмінною особливістю цих комп'ютерів була висока гнучкість при комплектації: наприклад, покупець, набуваючи такого комп'ютера з невеликою кількістю оперативної пам'яті та нешвидким процесором, міг розраховувати з часом, коли його потреби зростуть, придбати більш потужну модель, яка буде сумісна з вже купленою. Сумісність передбачала, що на новому комп'ютері працюватимуть ті самі програми, що й на старому, і це значною мірою гарантувалося операційною системою. Подібна гнучкість забезпечила сімейству цих комп'ютерів довге життя: різні моделі комп'ютерів, що випускаються протягом більш ніж 40 років, могли виконувати програми, написані у 1960-х роках.

Операційні системи цього сімейства, починаючи з OS/360, дозволяли прикладному забезпеченню абстрагуватися від специфіки окремих апаратних вузлів — для програм було неважливо, чи вони читають дані з перфокарт у 1960-х роках або з жорсткого диска в 2000-х. Операційні системи підтримували також файлові системи, реалізовували багатозадачність та віртуальну пам'ять, підтримували роботу віртуальних машин. Останнє дозволяло замінити одним потужним комп'ютером кілька комп'ютерів меншої потужності, що було зручно та вигідно для великих обчислювальних центрів.

Цікаво відзначити, що програми, створені для мейнфреймів IBM, пережили ці комп'ютери і продовжують успішно функціонувати досі у

багатьох великих установах США та Європи. За роки експлуатації та супроводу ці програми «вросли» до бізнес-процесів відповідних установ, і останні вкрай неохоче погоджуються на заміну їх на нове ПЗ. Фактично на ринку розробки ПЗ перемогла така модель: операційні системи та інше програмне оточення, в рамках яких працюють ці програми, емулюються для виконання на сучасних серверах. Протиставити цьому підходу можна рух автоматизованим реінжинірингом застарілого ПЗ (тобто переписування старих програм на нових мовах програмування), який активно розвивався в кінці 1990-х — на початку 2000-х років, але в результаті так і не зміг завоювати ринок.

Сімейство Unix-подібних ОС

Іншим знаменитим сімейством операційних систем є Unix. Вихідна версія Unix була розроблена наприкінці 1960-х років у компанії AT&T. Надалі було створено величезну кількість різних версій Unix, а також значну кількість Unix-подібних ОС. Нижче наведено нові можливості ОС Unix, що забезпечили її популярність.

• Ієрархічна файлова система із деревом каталогів. В інших ОС тієї епохи дані на жорсткому диску зберігалися у форматі, що нагадував таблиці баз даних, що забезпечувало високу продуктивність доступу до даних, але не мало гнучкості — такий формат був фіксований для окремої ОС, а файли, якими оперував Unix, мали різні формати.

• Наявність набору зручних і простих системних програм для реалізації базових функцій, які часто використовуються в прикладних програмах: сортування даних, пошуку та перетворення тексту, стиснення даних тощо.

• Зручна інтерактивна командна оболонка, яка дозволяла комбінувати програми в ланцюжки, наприклад, відправляючи виведення однієї програми на вхід іншої або заміщаючи введення з консолі введенням з файлу тощо. Така командна оболонка дозволяла створювати ансамблі із взаємодіючих програм, розроблених різними мовами, не вдаючись до деталей їх реалізації. Аналогічні

можливості хоч і надавалися раніше (наприклад, OS/360), але в Unix вдалося реалізувати їх набагато зручніше для кінцевого користувача.

Зупинимося детально на останній властивості - переносимості. Як і інші ОС того часу, Unix була реалізована мовою Асемблер. Але до середини 1970-х років її було переписано мовою С. Використання мови С дозволило швидко створювати нові версії Unix для різних комп'ютерних архітектур. При черговому перенесенні потрібно було реалізувати новий С-компілятор, що не є складним завданням через простоту цієї мови. Також потрібно було переписати невелику частину спеціалізованого коду Unix, який відповідав за роботу з конкретним сімейством процесорів. Компанія AT&T відкрила вихідні коди Unix для науково-дослідних установ, університетів та державних установ. В результаті було створено багато версій цієї Unix для різних комп'ютерних архітектур.

Основні підсистеми ОС Unix наведені нижче.

• Ядро, яке реалізовувало основні функції і включало мінімальний базовий набір драйверів апаратури.

• Набір утиліт — програм, які виконували сервісні функції, наприклад пошук файлів та стиснення даних.

• Набір програмних бібліотек та компіляторів, наприклад, компілятор мови С та її стандартні бібліотеки.

На закінчення відзначимо, що саме переносимість (використання мови С для реалізації та відкритість вихідних кодів) сприяла широкому поширенню Unix. Однак, незважаючи на те, що компанія AT&T забезпечила відкриті ліцензії для університетів та держустанов, ОС Unix не була вільно розповсюджуваною і нюанси її використання ставали предметом судових розглядів.

Сімейство вільно розповсюджуваних операційних систем Linux

У 1980-х роках Річардом Столлманом було запущено проект GNU для створення вільної Unix-подібної ОС. В результаті було реалізовано стандартні для ОС Unix утиліти, а також створено сімейство компіляторів GCC (GNU

Compiler Collection). Ядро ОС GNU Hurd, що розробляється в рамках цього проекту, виявилося менш вдалим, ніж реалізоване незалежно Лінусом Торвальдсом ядро Linux. У результаті популярність набув «тандем», що складається з ядра Linux і набору утиліт GNU, за яким закріпилася назва ядра — Linux. Перша версія Linux з'явилася 1991 року. Вона була сумісна з Unix (програми для Unix працюють під її керуванням) і поширюється під вільною ліцензією GPL. Майже за 30 років свого існування ОС Linux стала дуже популярною для серверів, а також для вбудованого та мережного обладнання. Перелічимо причини такої популярності.

• Можливість безкоштовно використовувати ОС Linux — ліцензія GPL, під якою поширюється Linux, надає значно більше свободи, ніж ліцензія Unix.

• Можливість самостійно комплектувати ОС, тобто створювати різні дистрибутиви, звільнені від функціональності, непотрібної в даному контексті; крім того, Linux можна доопрацьовувати для потреб, що також передбачено його архітектурою.

• Linux має скромні вимоги до обладнання – продуктивності процесора, обсягу оперативної пам'яті та жорстких дисків.

Для серверів використання ОС, яка має такі властивості, є істотною перевагою. Адже серверу не потрібні ні потужний драйвер графічного адаптера, ні «настільні» програми на кшталт веб-браузера. Зате можуть знадобитися специфічні системи зберігання даних і, відповідно, особливі драйвери їм. Останні, у разі Linux, можна вбудувати прямо в ядро ОС, що підвищує продуктивність. Можливість створювати свої дистрибутиви допомагає оптимізувати комп'ютери під керуванням Linux для різних завдань - наукових обчислень, специфічних веб-серверів, СУБД тощо.

Для мережного та вбудованого обладнання ситуація загалом така ж, як і для серверів, але ще більш виражена. Розглянемо простий мережевий роутер — «коробочку над дверима», до якої підключено кілька кабелів Ethernet і яка не з'єднана ні з монітором, ні з клавіатурою, ні з мишею. Все, що потрібно від роутера, - пересилати дані по мережі та забезпечувати можливість

дистанційного налаштування. Встановлена на нього ОС Linux, в якій залишено лише все необхідне, може займати на роутері лише кілька Mb. У результаті роутер можна зробити недорогим та забезпечити оптимальне використання його скромних ресурсів «за прямим призначенням».

Linux розповсюджується під ліцензією GPL (GNU Public License). Скажімо кілька слів про цю ліцензію. GPL – це спеціальний вид ліцензій, призначений для відкритого ПЗ (Open Source Software), в рамках якої автори можуть вільно використовувати вихідний код, що має цю ліцензію, але повинні також відкрити свій власний код. Крім ОС Linux під ліцензією GPL поширюються багато популярних програмних продуктів, таких як графічний редактор GIMP, СУБД MySQL, набір компіляторів GCC (GNU Compiler Collection), система керування версіями Git, аудіоредактор Audacity. Всі ці продукти можуть працювати під керуванням різних ОС: Windows, Linux, інших Unix-подібних.

Linux є безкоштовним та ґрунтується на Open Source ідеології, закріпленої ліцензією GPL. Linux розвивається великою спільнотою розробників. Примітно, що помітну частину роботи з цих відкритих проектів зараз беруть він великі IT-компанії, такі як Microsoft і IBM, оскільки багато запропоновані ними комерційні рішення також грунтуються на Linux і проекті GNU.

Створення на основі Linux корпоративних IT-рішень та застосування Linux для управління спеціалізованим обладнанням (навігаційним обладнанням, системами зв'язку тощо), а також використання ОС Linux для потреб військово-промислового комплексу спричиняють підвищені вимоги до надійності. Одним із способів забезпечення таких вимог є верифікація ПЗ — комплекс заходів щодо аналізу ПЗ на предмет відповідності специфікаціям та відсутності помилок. Верифікація потребує вирішення складних наукомістких завдань, навколо яких зосереджено увагу багатьох дослідницьких колективів у різних країнах.

Сімейство ОС DOS

На початку 1980-х з'явилися персональні комп'ютери сімейства IBM PC. Для них було створено операційну систему DOS (Disk Operating System) — дискову операційну систему, основним завданням якої була підтримка роботи з дисками — гнучкими (дискетами) та жорсткими (вінчестерами). Ці диски прийшли на зміну перфокарт, магнітним стрічкам та іншим колишнім засобам зберігання та введення даних. Програмну роботу з дисками, будучи трудомістким і водночас типовим завданням, було винесено до рівня ОС. Крім цього, операційні системи сімейства DOS надавали прикладним програмам сервіси виділення пам'яті, а також інформацію про системні події. Прикладами таких подій могли бути вставлення дискети в дисковод, натискання кнопки миші тощо.

Операційна система DOS була дуже простою: вона не підтримувала багатозадачності, не розмежовувала доступ користувачів до даних, не перешкоджала доступу програм до системних даних. Але водночас вона була недорогою у розробці та підтримці, невимоглива до апаратних та системних ресурсів — наприклад, могла працювати на комп'ютерах з оперативною пам'яттю менше ніж 1 Mb. Таким чином, ця операційна система добре підходила для перших IBM PC, які й самі позиціонувалися як недорогі персональні комп'ютери. Перші версії DOS були придбані, а потім спільно розроблялися компаніями IBM і Microsoft. Надалі розробку цих ОС продовжила компанія Microsoft, випустивши в період з 1981 по 1995 7 версій MS DOS.

Сімейство Windows

Розглянемо тепер сімейство операційних систем Windows. У міру зростання популярності IBM PC та інших аналогічних комп'ютерів актуальною стала задача створення в ОС багатофункціонального та уніфікованого інтерфейсу користувача, а також інтерфейсу для прикладних програм.

У 1985 році компанія Microsoft випустила першу версію Windows - графічну оболонку для IBM PC, що запускається над MS DOS. У першій половині 1980-х років графічний інтерфейс користувача був прогресивною інновацією. На початку 1980-х років у комп'ютерів Apple вже були дружні користувачеві графічні оболонки, які запозичили метафору Робочого столу у розробок Xerox Research, а Microsoft зайняла цю нішу на звичному для себе ринку програмного забезпечення для IBM PC.

Система Windows, окрім функцій MS DOS, надавала кінцевому користувачеві комп'ютера зручний графічний інтерфейс на основі метафори офісного столу, а програмісту — набір системних функцій, реалізованих у вигляді добре спроектованого програмного інтерфейсу. Також для Windows був розроблений великий набір бібліотек, які полегшували розробку програмного забезпечення з «багатим» віконним інтерфейсом.

Незважаючи на активний розвиток, Windows досить довго залишалася лише оболонкою над операційною системою DOS. Але в 1993 році компанія Microsoft випустила Windows NT, яка була вже повноцінною операційною системою і не вимагала своєї роботи DOS. Windows NT підтримувала багатозадачність, розмежування доступу користувачів, ізоляцію процесів, файлову систему з можливістю розмежування доступу користувачів до даних та інші можливості повноцінних операційних систем. Сучасна операційна система Windows 10 та версії Windows, призначені для роботи на серверах, є спадкоємцями Windows NT. В даний час операційні системи сімейства Windows є найпопулярнішими ОС для настільних комп'ютерів: вони встановлені більш ніж на 75% настільних комп'ютерів у всьому світі.

Сімейство мобільних ОС Android

Вимоги до операційних систем для мобільних пристроїв істотно відрізняються від «настільних» та серверних.

• Мобільним пристроям потрібен урізаний функціонал класичних ОС: їм не потрібна підтримка багатьох користувачів, не потрібні класичні системні

утиліти на зразок Unix-утиліт, не потрібна підтримка численних підключених зовнішніх пристроїв.

• Мобільні ОС повинні бути оптимізовані для виконання специфічних комунікаційних завдань, зокрема з урахуванням нестійкого стільникового зв'язку.

• Їм потрібна лише спрощена реалізація офісних програм.

• Мобільні ОС повинні ефективно підтримувати розважальні програми - ігри, аудіо/відео, фотокамера, шопінг.

• Потрібна оптимізація енергоспоживання для роботи в автономному режимі; при цьому сучасні мобільні пристрої часто мають процесори, які лише незначно поступаються за потужністю настільним.

Перша версія ОС сімейства Android була створена зусиллями компаній Google, Sony, DELL, Motorola та деяких інших у другій половині 2000-х років на основі Linux. Перші мобільні пристрої з Android надійшли у продаж у 2008 році. Остання на момент створення цього курсу лекцій версія Android 15 вийшла восени 2024 року.

ОС Android максимально ефективно реалізує представлені вище вимоги до мобільних ОС, додаючи до них перелічені нижче функції.

• Ізоляція програм один від одного: важливі дані програми зберігаються в окремих сховищах, недоступних іншим програмам.

• Шифрування локальних даних користувача.

• Централізований мережевий магазин програм.

• Спеціальна підтримка фотокамери для забезпечення високої якості фото- та відеозйомки без повноцінного об'єктива та великої світлочутливої матриці.

• Спеціалізований звуковий драйвер, який обробляє окремо сигнали з кількох мікрофонів для ефективного придушення навколишнього шуму під час розмови.

Спочатку програми для ОС Android розроблялися переважно мовою Java (з можливим підключенням бібліотек на C/C++), а основним інструментом для

розробки було середовище Eclipse. З 2014 року Google пропонує використовувати для розробки середовище Android Studio, створене на базі відомого середовища розробки IntelliJ IDEA компанії JetBrains. У 2017 році основною мовою розробки для Android-додатків була оголошена мова Kotlin, також розроблена компанією JetBrains.

Зазначимо, що багато перерахованих властивостей ОС Android також характерні й для інших сучасних мобільних ОС, яка з'явилася дещо раніше iOS і дещо пізніше — Windows Phone.

ТЕМА 2.9. ВИСОКОПРОДУКТИВНІ СИСТЕМИ

План

1. Приклади обчислювально-складних завдань.

2. Системи із загальною пам'яттю, системи із ізольованою пам'яттю.

3. Багатопоточність та розподілені обчислення.

4. Хмарні обчислення з використанням віртуальних серверів Amazon EC2 та Microsoft Azure.

Обчислювально складні завдання

Більшість людей зараз звикли користуватися комп'ютерами для виконання виробничих та особистих канцелярських завдань, а також для розваг. Однак протягом всієї історії обчислювальної техніки для вирішення складних обчислювальних завдань були потрібні і продовжують залишатися необхідними високопродуктивні ЕОМ. З розвитком обчислювальних можливостей ЕОМ, а також прогресом науки та індустрії розширювався і клас цих завдань. Перерахуємо деякі з таких завдань, які є актуальними на даний момент.

• Моделювання фізичних процесів у гідро- та аеродинаміці, ядерній фізиці, механіці, що дозволяє обійтися без дорогих та небезпечних експериментів.

• Аналіз експериментальних даних в астрономії, метеорології, гідро- та аеродинаміці.

• Завдання біоінформатики, зокрема геноміки, які вимагають виконання алгоритмів на дуже великих обсягах вхідних даних (йдеться про сотні гігабайт); і ці алгоритми мають суттєву обчислювальну складність.

• Вирішення криптографічних завдань, які вимагають перебору великої кількості випадкових рішень, наприклад, майнінг криптовалюти.

На початку курсу розповідалося про закон Мура, згідно з яким продуктивність ЕОМ згодом зростає експоненційно. Також вказувалося, що через фізичні обмеження подальше нарощування продуктивності процесорів

неможливо виконати за рахунок більш «щільного» розміщення транзисторів на одній кремнієвій пластині. Нагадаємо, що так відбувається тому, що розміри транзисторів стають порівнянними з розмірами окремих атомів, а їх взаємодія обмежена швидкістю світла. Подальшим перспективним напрямом збільшення продуктивності є розпаралелювання.

Розпаралелювання *— це організація програм та засобів їх виконання у вигляді набору паралельних взаємодіючих активностей.*

Важливо відзначити, що сама програма може бути реалізована з використанням різних конструкцій паралельного програмування (наприклад, за допомогою засобів багатопоточності), так і виконуватися на різних паралельних архітектурах, наприклад, у хмарній інфраструктурі з використанням різної кількості обчислювальних вузлів.

Зазначимо також, що часто під розпаралелюванням розуміють додавання паралелізму для вже реалізованого алгоритму/програми, вихідно реалізованих послідовно. Таким чином, термін «розпаралелювання» відображає той факт, що створювати паралельні програми непросто і це потребує спеціальних зусиль.

Системи із загальною пам'яттю

У цьому розділі ми розглянемо типові підходи до створення паралельних обчислювальних систем. Спочатку зупинимося на системах із загальною пам'яттю.

Система із загальною пам'яттю *— це обчислювальна система, в якій паралельно виконувані обчислювальні процеси можуть одночасно працювати з одними й тими самими даними, зчитуючи та змінюючи їх.*

Розглянемо такі методи організації таких систем:

• багатоядерні процесори;

• багатопроцесорні комп'ютери.

Багатоядерні процесори. Ми вже розглядали багатоядерні процесори вище, тут зробимо акцент на використання багатоядерних процесорів для високопродуктивних обчислень.

Основна проблема, з якою стикаються інженери під час проектування багатоядерних процесорів, полягає у забезпеченні узгодженої роботи ядер. Останні одночасно працюють із загальними даними в оперативній пам'яті, що є джерелом помилок. При цьому доводиться шукати компроміс між двома такими рішеннями.

• Використання ядрами загального каналу обміну даними з пам'яттю та загальною кеш-пам'яттю (просте в реалізації, але неефективне рішення).

• Використання окремих каналів для роботи з пам'яттю та окремої кеш-пам'яті для кожного ядра, що наближає багатоядерний процесор до багатопроцесорних систем (складне в реалізації, але ефективне рішення).

Приклад пристрою кеш-пам'яті багатоядерного процесора Intel Core i7, який ми наводили вище, ілюструє складності, що виникають при пошуку описаного компромісу.

Багатопроцесорні комп'ютери. У таких комп'ютерах паралелізм реалізується через набір процесорів, які, подібно до ядра багатоядерних процесорів, працюють паралельно. Такий підхід дозволяє до певної міри подолати технологічні обмеження за кількістю ядер, які вдається розмістити на одній кремнієвій пластині процесора і ще більше збільшити продуктивність системи.

У багатопроцесорних комп'ютерах вузьким місцем є шини, які використовуються доступу процесорів до загальної оперативної пам'яті. Також при проектуванні шин багатопроцесорних комп'ютерів виникають складні завдання синхронізації різних процесорів та забезпечення когерентності (узгодженості) їхньої кеш-пам'яті. Важливо розуміти, що багатопроцесорні комп'ютери мають дуже складний пристрій, вирішуючи ті ж проблеми, що і багатоядерні процесори, але в більшому масштабі. І вирішувати їх доводиться вже на рівні системної шини, інколи ж — на рівні особливого пристрою оперативної пам'яті.

Багатопроцесорні комп'ютери з'явилися торік у 1960-х роках. Серед перших були комп'ютери Burroughs, і навіть деякі моделі System/360. Багато

процесорів сімейства Intel x86 починаючи з Pentium (1993) можна використовувати для створення багатопроцесорного обчислювального комплексу. Нині ситуація із багатопроцесорними комп'ютерами така.

• На ринку є двопроцесорні робочі стації, які використовуються для високопродуктивних завдань без залучення кластерів та суперкомп'ютерів — наприклад, для наукових обчислень чи відеомонтажу.

• За потреби забезпечення високопродуктивних серверів (наприклад, у дата-центрах) використовуються 2–4-процесорні комп'ютери (наприклад, на базі процесора Intel Xeon). Це буває необхідно для масштабування веб-сервісів і баз даних, розширення можливостей віртуалізації тощо.

• Спеціалізовані багатопроцесорні обчислювальні системи, на базі яких створюються обчислювальні кластери та суперкомп'ютери.

Багато сучасних багатопроцесорних робочих станцій та серверів належать до класу архітектур *NUMA (Non-Uniform Memory Access, неоднорідний доступ до пам'яті).* У цих архітектурах кожного процесора є власна оперативна пам'ять, відповідно, до неї процесор має швидкий доступ, а доступ до оперативної пам'яті іншого процесора здійснюється через спеціальну шину. Операційна система при цьому дозволяє відобразити адресний простір процесу, що виконується на будь-якому процесорі, на згадку про інший процесор. Як і у випадку з віртуальною пам'яттю це робиться прозоро для прикладних програм, але незмінно позначається на продуктивності, що слід враховувати при написанні програм для таких архітектур.

Слід зазначити, що нині в персональних комп'ютерах масово використовується не багатопроцесорність, а багатоядерність, що обумовлено ринковою доцільністю.

Системи із ізольованою пам'яттю

Тепер розглянемо інший спосіб організації паралельних обчислювальних систем — системи із ізольованою пам'яттю.

Системи із ізольованою пам'яттю *— це такі обчислювальні системи, у яких паралельні активності працюють лише з власними даними, а взаємодія між ними реалізується за допомогою обміну повідомленнями.*

Високопродуктивні обчислювальні системи із ізольованою пам'яттю організуються як обчислювальних кластерів. На апаратному рівні, в рамках обчислювальних кластерів та суперкомп'ютерів, розпаралелювання реалізується об'єднанням комп'ютерів у єдину систему за допомогою високошвидкісної локальної мережі. Як і у випадку з шинами у багатопроцесорних комп'ютерах, у обчислювальних кластерах вузьким місцем є синхронізація та взаємодія обчислювальних вузлів.

Суперкомп'ютери – це великі обчислювальні кластери. Історично суперкомп'ютером називали будь-яку ЕОМ (не обов'язково кластер) з дуже високою продуктивністю. Як приклад можна навести суперкомп'ютер CDC 6600 компанії Cray Research, створений 1963 року. Іноді до суперкомп'ютера для підвищення продуктивності додаються спеціалізовані обчислювальні засоби — наприклад, для швидкого множення великих матриць або для ефективної паралельної обробки масивів з цілими даними. На даний момент цей суперкомп'ютер має у своєму складі 668 двопроцесорних вузлів з 14-ядерними процесорами. Він займає кілька великих приміщень і споживає електроенергії до 640 кВт (для порівняння нагадаємо, що сучасна міська квартира споживає в середньому менше 1 кВт).

Технології реалізації високопродуктивних обчислень

Багатопоточність. *На програмному рівні паралельні системи із загальною пам'яттю можуть бути реалізовані за допомогою багатопоточності, тобто використання в програмі паралельних потоків (threads), що працюють в рамках одного процесу і спільно використовують дані програми та інші доступні ресурси.*

Засоби реалізації багатопотокових додатків вбудовані в основні мови програмування, зокрема в мову C++. В останньому є спеціальний клас thread (для його використання потрібно підключити бібліотеку thread.h), і коли

програміст створює об'єкт цього класу і ініціалізує його деякою функцією F, то стартує новий потік, в якому і виконується функція F. Бібліотека thread.h надає також засоби для синхронізації різних потоків при роботі з функціями та ін.

Багатопотокові програми дозволяють суттєво підвищити продуктивність додатків, але загрожують важко виявленими помилками. Найпоширенішою помилкою багатопотокового програмування є гонка за даними (Data Race). Ця помилка виникає в тому випадку, коли два потоки без спеціальної синхронізації (тобто в довільному порядку) звертаються до однієї і тієї ж ділянки пам'яті, і одне з цих звернень є записом даних. Таким чином виявляються можливим сценарії, коли читання з одного потоку передує записи з іншого в ту саму змінну і навпаки. При цьому програміст, коли писав програму, не припускав наявності першого вищезгаданого сценарію. Наявність такого недетермінізму означає відсутність синхронізації потоків. Воно призводить до помилкових результатів при роботі програми — наприклад, програма не робить зарахування грошей клієнта на його банківський рахунок, а повинна була... Складність ситуації пов'язана з тим, що ефекти від перегонів, що відбулися, можуть виявлятися не відразу, а після деякого часу. Існує багато підходів до пошуку гонок – на сьогодні для індустрії ця тематика є надзвичайно затребуваною.

Для спрощення розробки багатопоточних програм багатопроцесорних систем є спеціальні мови програмування. Один із відомих сучасних прикладів таких мов — OpenMP, що з'явився в 1997 році. Це мова, яку можна вбудовувати в інші мови програмування (найчастіше - в C і Fortran) для розпаралелювання програмного коду, що в основному виконує математичні обчислення.

Фактично OpenMP є вбудованою предметно-орієнтованою мовою програмування. В даний час тема предметно-орієнтованих мов програмування є дуже популярною, оскільки такі мови дозволяють досягти численних вигод і створюються не тільки для специфічних областей, але й для окремих великих

програмних проектів та лінійок продуктів. Є також спеціальні засоби для створення інфраструктури для таких мов – продукт JetBrains під назвою MPS, відкрита технологія Xtext зі світу Eclipse та ін. Поверхнево-вбудовані мови – це такі предметно-орієнтовані мови, які можна використовувати спільно зі звичайними мовами.

Розподілені обчислення. На програмному рівні розпаралелювання може здійснюватися за допомогою розподілених обчислень: прикладна програма реалізує використання та синхронізацію комп'ютерів обчислювального кластера, керуючи тим, які операції та над якими даними виконувати кожен з них, а також те, як вони цими даними обмінюватимуться. Розподілені обчислення дозволяють, наприклад, одночасно обробляти різних комп'ютерах різні порції даних чи, організувавши з комп'ютерів кластера обчислювальний конвеєр, розподілити ними різні стадії обробки даних.

Однією з найпопулярніших технологій організації розподілених обчислень є бібліотека Message Passing Interface (MPI), призначена для організації розподіленої роботи програм на обчислювальних кластерах. MPI дозволяє паралельно запустити кілька екземплярів програми (10, 100, 500) на різних процесорах кластера та синхронізувати роботу цих екземплярів із загальними даними. Функціональність бібліотеки MPI описана в однойменному стандарті. Існує значна кількість різних реалізацій стандарту MPI - наприклад, MPICH, OpenMPI тощо.

Хмарні обчислення

Один і той же програмний додаток може бути розміщений і виконується на різній кількості обчислювальних вузлів. Це буває потрібно для паралельних додатків з метою підвищити їхню пропускну спроможність (наприклад, для клієнт-серверних додатків або додатків Інтернету речей). З іншого боку, обчислювальні ресурси коштують дорого та їх збільшення для цього додатка має бути виправданим. Крім цього, слід зазначити, що обчислювальний кластер (любий, який навіть не є суперкомп'ютером) виявляється дорогим у придбанні та обслуговуванні. Окремим дослідникам, невеликим лабораторіям

та організаціям зазвичай вигідніше орендувати обчислювальні потужності за необхідності, а не купувати таке обладнання. В останні десятиліття для вирішення подібних завдань часто застосовуються послуги хмарних обчислень.

Сервіс хмарних обчислень дозволяє виділяти обчислювальні ресурси (віртуальні машини) у кількості, необхідному користувачеві на даний момент, та налаштовувати мережеві з'єднання між ними. У результаті користувач тимчасово отримує доступ до кластера потрібної потужності та конфігурації. Широко відомі такі хмарні послуги, як Amazon EC2 або Microsoft Azure.

Технології хмарних обчислень дозволяють клієнтам отримати доступ до серверів у потрібній кількості. Цю кількість можна за бажанням швидко варіювати, але, як і у випадку з реальними серверами, ці віртуальні сервери необхідно налаштовувати та супроводжувати. Це складне завдання системного адміністрування, незважаючи на наявність великої кількості спеціалізованих програмних утиліт для синхронного управління парком серверів.

Організація паралельних хмарних обчислень спрощується під час використання обчислювальних сервісів класу *Функція як послуга* (Function as Service). Найбільш відомими представниками цього класу є платформи Amazon AWS Lambda, Microsoft Azure Functions та Google Cloud Functions. Вони надають користувачеві програмний інтерфейс, що дозволяє передати сервісам програмний код та вхідні дані. Користувач реалізує функції однією з підтримуваних мов програмування (підтримуються популярні високорівневі мови програмування, такі як Java, C#, Python та інші) і передає посилання на ці функції до бібліотеки, яка, у свою чергу, передає код цих функцій на сервери Amazon, Microsoft або Google і ініціює їх віддалене виконання. Сервіси вирішують самостійно, яким чином розподілити обчислення на реальні сервери. Такий підхід позбавляє необхідності вручну настроювати сервери, але позбавляє користувача гнучкості в налаштуванні паралельних обчислень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Азаров О. Д., Гарнага В. А., Клятченко Я. М., Тарасенко В. П. Комп'ютерна схемотехніка : підручник. Вінниця : ВНТУ, 2018. 230 с.
2. Вічужанін В. В. Цифрова схемотехніка : навч. посібник. Одеса : ОНПУ, 2018. 62 с.
3. Злобін Г. Г., Рикалюк Р. Є. Архітектура та апаратне забезпечення комп'ютерів : навч. посіб. Київ : Каравела. 2016. 224 с.
4. Комп'ютерна схемотехніка та архітектура комп'ютерів : навч.-метод. посібник / О. В. Задерейко та ін. ; Нац. ун-т «Одеська юридична академія». Одеса, 2022. 288 с. URL: https://dspace.onua.edu.ua/bitstreams/d5aaa5eb-1be9-4eed-b56c-9e5c3ca42baf/download
5. Комп'ютерна схемотехніка та архітектура комп'ютерів : навч. посіб. / О. В. Задерейко, Н. І. Логінова, О. Г. Трофименко та ін. Одеса : Фенікс, 2021. 163 с.
6. Комп'ютерна схемотехніка та архітектура комп'ютерів : навч. посіб. для підгот. здобув. вищої освіти галузі знань 12 «Інформаційні технології» / О. В. Задерейко та ін. ; Нац. ун-т «Одес. юрид. академія». Одеса : Фенікс, 2024. 251 с. https://doi.org/10.32837/11300.27830
7. Матвієнко М. П., Розен В. П, Закладний О. М. Архітектура комп'ютерів. Київ : Ліра-К, 2013. 264 с.
8. Минайленко Р. М., Конопліцька-Слободенюк О. К., Гермак В. С. Комп'ютерна схемотехніка : навч. посіб. Кропивницький : Видавець Лисенко В. Ф., 2022. 153 с.
9. Сенько В. І. Панасенко М. В., Сенько Є. В. Електроніка і мікропроцесорна техніка. Київ : Каравела, 2015. 676 с.

10. Сенько В. І., Панасенко М. В., Сенько Є. В. Електроніка і мікросхемотехніка. Том 3. Цифрові пристрої : підручник. Київ : Каравела, 2017. 400 с.
11. Цирульник С. М., Азаров О. Д., Крупельницький Л. В., Трояновська Т. І. Мікропроцесорна техніка : навчальний посібник. Вінниця : ВНТУ, 2017. 123 с.
12. Multisim Education. URL: https://www.ni.com/en/support/downl+ +oads/software-products/download.multisim.html
13. SIV - System Information Viewer. URL: http://rh-software.com
14. TOP500 Becomes a Petaflop Club for Supercomputers. URL: https://www.top500.org

Навчальне видання

КОМП'ЮТЕРНА СХЕМОТЕХНІКА ТА АРХІТЕКТУРА КОМП'ЮТЕРІВ

Конспект лекцій

Укладач:

Жебко Олександр Олегович

Формат 60х84 1/16. Ум. друк. арк. 8.87.

Наклад 50 прим. Зам. № _____

Надруковано у видавничому відділі

Миколаївського національного аграрного університету

54020, м. Миколаїв, вул. Георгія Гонгадзе, 9

Свідоцтво суб'єкта видавничої справи ДК № 4490 від 20.02.2013