МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ДНІПРОВСЬКА ПОЛІТЕХНІКА»

С.М. Ткаченко

АРХІТЕКТУРА КОМП’ЮТЕРІВ

Навчальний посібник

Дніпро
НТУ «ДП»
2026

УДК 004.21:004.22 (075.8)
Т 48

Рекомендовано вченою радою НТУ «Дніпровська політехніка»
як навчальний посібник для здобувачів ступеня бакалавра
галузі знань F Інформаційні технології
(протокол № 10 від 18.03.2026)

Рецензенти:
А.І. Гуда, д-р техн. наук, проф. (Український державний університет науки і технологій);
В.В. Герасимов, канд техн. наук, доц. (Дніпровський національний університет ім. О. Гончара).

Ткаченко С. М.
Т 48 Архітектура комп'ютерів [Електронний ресурс] : навч. посіб. / С.М. Ткаченко ; М-во освіти і науки України, Нац. техн. ун-т «Дніпровська політехніка». – Дніпро : НТУ «ДП», 2026. – 151 с.

Викладено основи архітектури комп'ютерів. Розглянуто їх історичний розвиток і класифікацію. Приділено увагу структурі та принципам роботи комп'ютерів на апаратному, традиційно машинному та операційному рівнях. Окремі розділи присвячено організації асоціативної пам'яті, роботі зовнішніх пристроїв і методам керування введенням-виведенням.

Містить питання для самоперевірки, що сприяють більш глибокому засвоєнню матеріалу, а також формуванню компетентностей у сфері побудови, аналізу й оптимізації комп'ютерних систем.

Для здобувачів вищої освіти спеціальностей галузі знань **F Інформаційні технології**, а також може бути корисним викладачам, інженерам і всім, хто цікавиться фундаментальними засадами архітектури комп'ютерів.

УДК 004.21:004.22 (075.8)

ЗМІСТ

ПЕРЕДМОВА

Дисципліна «Архітектура комп'ютерів» належить до циклу фахових і посідає важливе місце в підготовці бакалаврів галузі F Інформаційні технології. Мета дисципліни – формування у здобувачів вищої освіти компетентностей, що дають змогу виконувати обґрунтування вибору структури, архітектури, налагодження та розробки прикладного програмного забезпечення для комп'ютерів загального і промислового призначення. Завдання вивчення дисципліни є засвоєння здобувачем базових знань і навичок у таких питаннях: проєктування, впровадження та обслуговування комп'ютерних систем, мереж різного виду й призначення; використання нових технологій, включаючи технології розумних, мобільних, зелених і безпечних обчислень; модернізація та реконструкція комп'ютерних систем і мереж, вбудованих та розподілених додатків; організація й технічне оснащення робочих місць з комп'ютерним устаткуванням.

Навчальний матеріал дисципліни здобувачі опановують на першому курсі протягом першого чи другого семестру, а тому вони мають бути обізнані з програмами інформатики, математики, фізики на рівні випускника повної чи спеціальної середньої освіти. Засвоєний матеріал буде корисним у вивченні операційних систем, системного програмування, комп'ютерної схемотехніки, комп'ютерних мереж, теорії комп'ютерних систем.

В основу посібника покладено матеріал лекційного курсу дисципліни, який упродовж низки років автор викладав у Національному технічному університеті «Дніпровська політехніка». Водночас посібник є більш розширеним варіантом електронного курсу лекцій, розміщеного на платформі дистанційної освіти НТУ «ДП». Посібник можуть використовувати в самостійній роботі студенти заочної форми навчання. Список літератури включає публікації, матеріал яких допоможе майбутнім фахівцям у галузі знань «Інформаційні технології» поглибити розуміння архітектури комп'ютерів у широкому значенні цього поняття, яке включає комп'ютери загального, спеціального призначення, промислові комп'ютери й контролери, інтернет речей, віртуальні й вбудовані системи. У ньому розглянуто основні функціональні структури, технічні засоби, принципи організації апаратно-програмного забезпечення та його взаємодії з операційними системами й прикладними програмами.

Посібником мають користуватися здобувачі бакалаврського ступеня спеціальностей галузі знань F Інформаційні технології на першому курсі навчання, опановуючи базові відомості й навички, що дозволить продовжити підготовку за обраною спеціальністю. Автор з вдячністю розгляне всі побажання та зауваження від читачів цього посібника, що сприятиме подальшому удосконаленню його змісту.

1 КЛАСИФІКАЦІЯ ОБЧИСЛЮВАЛЬНИХ ЗАСОБІВ ЗА ПРИНЦИПОМ ДІЇ В РЕТРОСПЕКТИВІ ЇХ ІСТОРИЧНОЇ ЕВОЛЮЦІЇ

1.1 Первісні оптичні обчислювальні засоби

Доісторичним класом обчислювальних засобів можна вважати засіб, що використовує оптичні принципи. Це сонце та інші небесні світила. Первісна людина використовувала добове переміщення сонця на небосхилі для визначення часу протягом доби та підрахунку діб, зміни фази місяця для врахування більш значних проміжків часу – тижнів, місяців, року. Полярна зірка використовувалась людиною для визначення сторін світу, тобто, примітивного геопозиціонування. Інші природні засоби, такі як зміна пір року, зміни вигляду і властивостей рослин у часі (наприклад, соняшник або листвяне дерево), у просторі (розташування моху з північної сторони дерев чи каміння) пов'язані добовими, місячними та річними астрономічними циклами і зрештою, також використовують сонячну енергію. Всі ці засоби не є результатом цілеспрямованої діяльності людини. Вони існували і існують об'єктивно, тому не можна їх розглядати як один з етапів еволюції обчислювальних засобів і машин.

1.2 Механічні обчислювальні засоби і машини

У перших обчислювальних засобах людина використовувала те, що у неї було під рукою – відібрані й підготовані природні матеріали і власну м'язову силу. І якщо пальці на долонях – це природний засіб рахунку, то зібрані камінці, які зберігаються у виділеному для цього місці – у вибоїні гірської породи, у дуплі дерева, у шкіряному мішечку на шиї – це вже штучно створений інструмент підрахунку, наприклад, запасів їжі, та, одночасно, засіб зберігання інформації. Його еволюція призвела до виникнення шкіряних мотузок з вузлами для підрахунків, абаків, що широко використовувалися у середньовіччі і до рахівниці, яку можна було зустріти в офісах мало не до кінця XX століття.

Більш складні прадавні інструменти – кам'яні, глиняні, шкіряні, дерев'яні поверхні, куди можна було заносити у тому числі і кількісну інформацію, у тому числі і для обчислень. Механічні обчислювальні засоби дали людству першу поширену в Європі (але не історично першу) систему числення. Римська система числення, яка насправді запозичена римлянами у стародавній Греції, у своїй основі використовує метод малювання паличок або зарубок по кількості відігнутих пальців (римська п'ятірка означає долоню). Тим не менш тут використовувались глиняні таблички, пергамент, стилоси як засоби обробки і зберігання даних. Розвиток механічних обчислювальних засобів дав людству таблицю Піфагора, яка широко використовувалась в феодальній Європі для інженерних та банківських розрахунків.

Окрім системи числення та засобів запису і розрахунків механічні засоби започаткували більш складні механічні системи і машини. Різноманітні важелі, зубчаті, пасові передачі, вали, ексцентрики, гвинт Архімеда – все це елементи машин, які можна розглядати і як обчислювальні. Таким чином лебідки, підйомники, терези, балісти та онагри під час виконання свого призначення виконують

перетворення механічної енергії або роботи з використанням математичної функції додавання, множення, ділення, порівняння, інтегрування та диференціювання. На сьогодні будь-яка приватна квартира або навчальна аудиторія використовує механічний обчислювальний пристрій – дверний замок. Цей пристрій отримує дані, нанесені механічним способом на бороду ключа, порівнює їх з еталонними на секреті і, використовуючи м'язове зусилля руки людини, встановлюється дозвіл на допуск у приміщення. Заміна секрету замка може розглядатися як перепрограмування обчислювальної машини в частині зміни коду доступу. Як видно, механічні обчислювальні машини і елементи використовуються і будуть використовуватись, хоч на перший погляд вони і не сприймаються у такій якості.

Механічні обчислювальні машини використовувались і для виключно обчислень. Перші дослідження у цій сфері виконувались ще Архімедом Сіракузським. Вченим була зроблена спроба створити обчислювальну машину на базі зубчатих коліс і передач. Донедавна вважалося, що робота не була закінчена не стільки через смерть Архімеда та недоліки технологій і матеріалів, скільки через відсутність належного інформаційного забезпечення. Використовувана Архімедом римська, точніше грецька, система числення була непозиційною, а отже не мала вагових розрядів і була непридатною для формалізації великих обчислень. Тим не менш, на сьогодні знайдений в Егейському морі біля острова Антикітера так званий **антикітерійський механізм** – прилад для обліку руху небесних тіл на зубчатих колесах – деякі фахівці приписують саме Архімеду [1].

Позиційною системою числення є сучасна десяткова система, яку ми знаємо як арабську завдяки хрестовим походам середньовічної Європи. Насправді це індійська система числення, яка з'явилась у 1-4 ст. н.е. [2]. Довгий час вона не визнавалась Європою і була приводом питань з боку Інквізиції, оскільки без використання таблиць Архімеда з невідомо чиєю поміччю дозволяла швидко у стовпчик робити складні розрахунки. Лише розвиток суспільства з переходом до ранньокапіталістичних відносин та посиленням попиту на обробку економічних та технічних даних дозволив еволюційним шляхом перейти до арабської десяткової системи числення. Наприклад, повний перехід Франції до десяткової системи числення відбувся лише у 1799 році з переходом до метричної системи [3].

Десяткова позиційна система числення дозволила повернутися до ідей Архімеда і у 1642 відомий вчений Блез Паскаль створив механічну обчислювальну машину, яка дістала назву «паскаліна» [4].

Ще раніше Паскаля механічну обчислювальну машину у 1623 році розробляв інший вчений – німець Вільгельм Шиккард [5]. Але невідомо, чи була ця машина створена за його життя. Сам вчений помер 1635 році від холери, а діюча модель була побудована лише у XX ст.

Рисунок 1.1 – Машина Паскаля

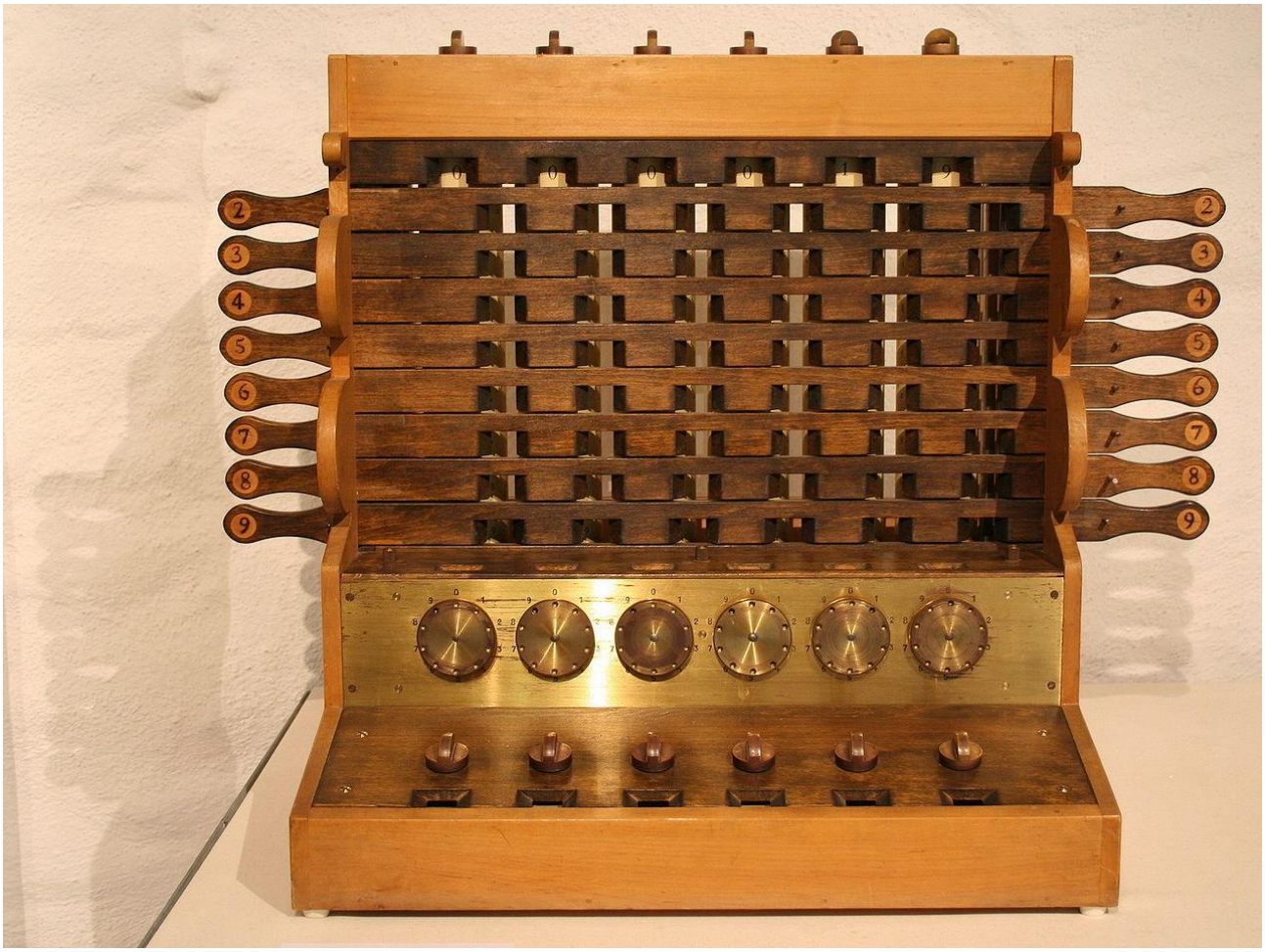

Рисунок 1.2 – Розрахунковий годинник Шиккарда

Механічні обчислювальні машини, споріднені «паскаліні» активно використовувались до середини XX століття, а побудовані на їх основі касові апарати –

і до 80-х років. Нижче зображена відома механічна обчислювальна машина «Фелікс».

Рисунок 1.3 – Машина «Фелікс»

Готфрід Вільгельм Лейбніц (1646-1716 рр.) у 1673 р. спробував використати вже відому на той час двійкову систему числення для побудови механічних обчислювальних систем. Йому вдалося удосконалити відому на той час «паскаліну». Прилад отримав назву «арифмометр Лейбніца» [6].

Двійкова система числення також була застосована на ткацькому станку Жозефа Марі Жаккара. Це станок з програмним управлінням, де поведінка виконавчих органів може бути задана просічними отворами на перфокартах, які об’єднані в стрічку. Методом нанесення отворів на перфокарту у двійковому коді задавався малюнок на тканині [7]. Як виглядає жакардова тканина, жакардова ковдра, відомо багатьом.

Метод запису даних на перфокарту має назву «метод Жаккара». Він був використаний у подальшому кроці людства до створення сучасного комп’ютера.

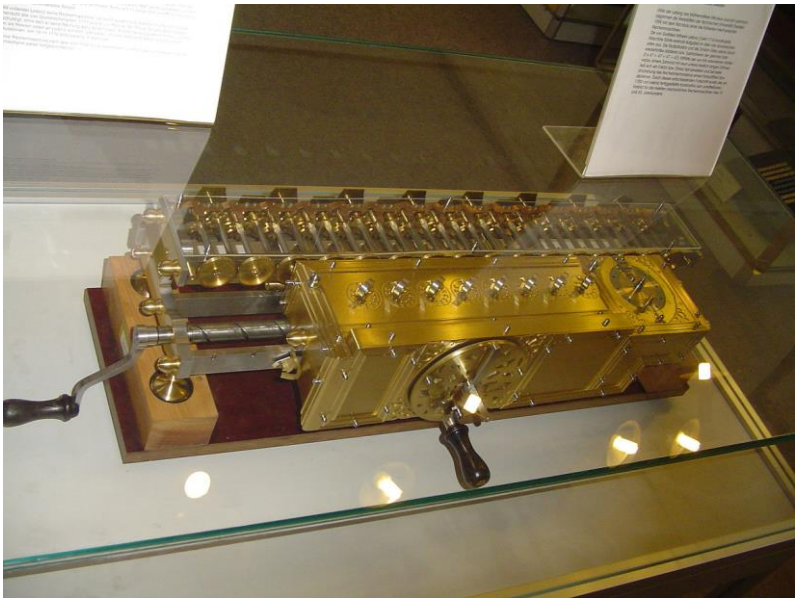

Рисунок 1.4 – Арифмометр Лейбніца

Рисунок 1.5 – Станок Жакара

Британський винахідник Чарльз Бебідж, у період приблизно з 1823 р. до 1871 р. веде розробку спочатку так званої «різницевої», а згодом «аналітичної» машини, що використовує двійкову систему числення, та є аналогом програмованої електронної цифрової машини. Архітектурно тут є механічний «млин», функціонально це процесор, «сховище» – регістри пам'яті, пристрій введення-виведення на перфокартах. На його прохання, інша дослідниця, Ада Лавлейс зробила опис цифрової обчислювальної машини та інструкції з програмування до неї, що вважається першими у світі програмами [8]. Роботи Бебіджа за його життя не були закінчені.

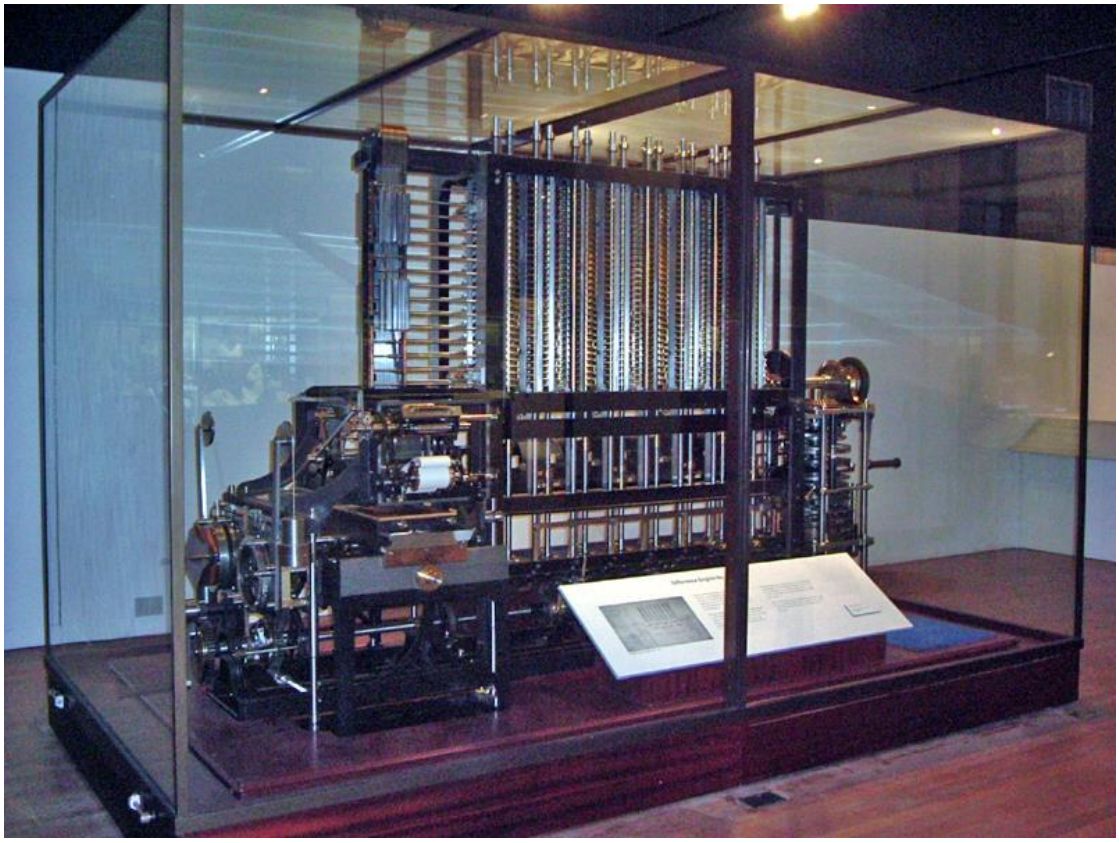

Рисунок 1.6 – Реконструкція (діюча модель) «різницевої машини 2» , виготовлена за проєктом Бебіджа

І наостанок огляду механічних обчислювальних машин згадаємо американського винахідника Германа Холеріта [9]. Його механічна обчислювальна машина була випробувана у 1887 році. Вона мала у своїй основі архітектуру Бебіджа і використовувалась для вирішення практичної задачі обліку населення під час перепису. Машина залишається програмованою механічною, але має електропривод, що підвищило швидкість обчислень у чотири рази у порівнянні з ручним аналогом. У 1896 році Холеріт створює компанію TMC, яку продав у 1911 році до корпорації C-T-R, яка у 1924 році була перейменована в IBM. Стосовно винаходу Холеріта, то варто зазначити що у 20-30 роки минулого століття банківська система США та ряду інших країн активно впроваджує механічні

обчислювальні комплекси з електроприводами, а згодом перфокарта Холеріта продовжила своє існування як засіб зберігання даних у електронних обчислювальних машинах.

1.3 Гідравлічні обчислювальні засоби і машини

Гідравлічні обчислювальні пристрої зустрічаються ще у древньому Єгипті. Вони називаються водяними годинниками. Найдавніший знайдений єгипетський водяний годинник (клепсидра) датується XIV ст. до н.е. Він міг відраховувати нічні та денні проміжки часу з урахуванням місяців єгипетського календаря [10].

Водяні годинники також були відомі в Ассирії, Вавилоні, древній Греції, та древньому Китаї. В останньому випадку водяні годинники використовувались вже за 2,5 тис. років до н.е. Схожий принцип – рівномірне переміщення, прибування або вибування речовини – використовують пісочні та вогняні годинники, в яких вибуває спалювана олія.

У середні віки водяні годинники широко використовуються до XVII ст.

У 1936 радянським вченим В. С. Лук'яновим був створений так званий гідравлічний інтегратор, який використовувався для вирішення диференційних рівнянь. Ці гідравлічні обчислювальні машини були доведені до серійного випуску, експортувались у країни соціалістичного табору, залучались до розрахунків проєктів Каракумського каналу у 1940-і роки, будівництва Байкало-Амурської магістралі у 1970-і роки, використовувались у геології, шахтобудівництві, металургії, ракетобудуванні. Гідравлічні інтегратори використовувались до середини 80-х років XX століття [11].

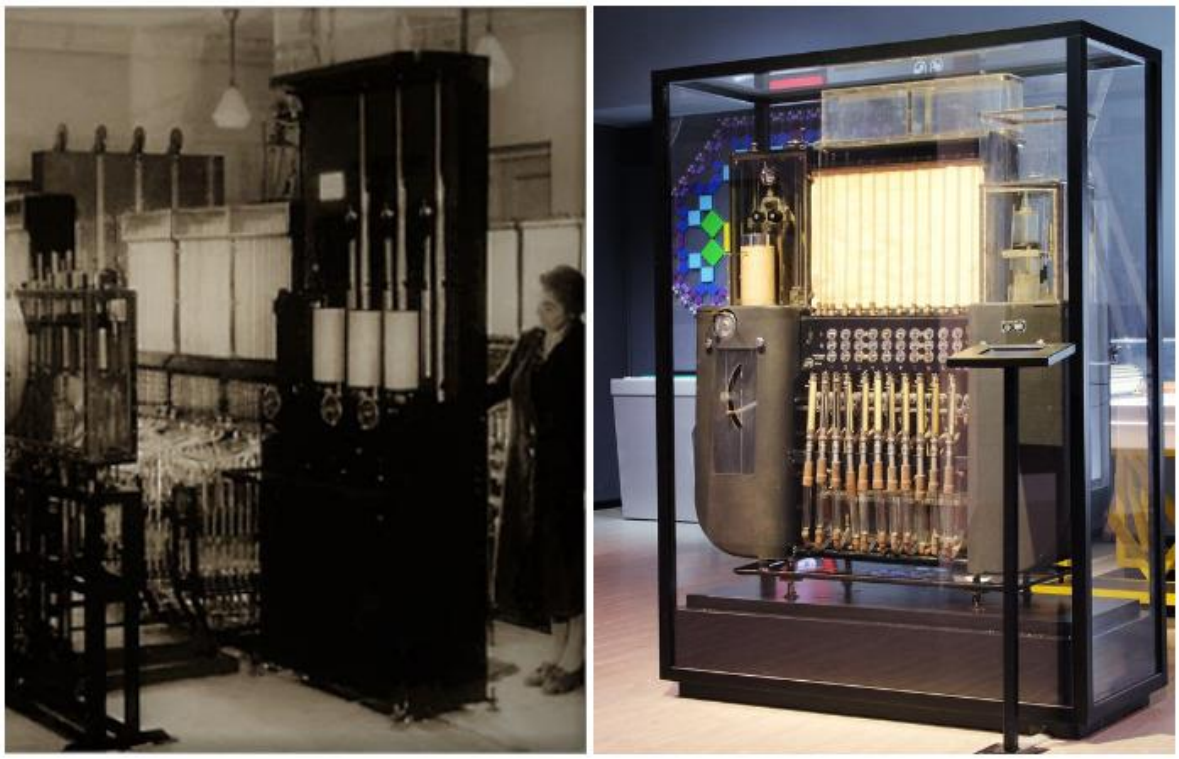

Рисунок 1.7 – Гідравлічний інтегратор Лук'янова

Також у 1949 році у Новій Зеландії Вільямом Філіпсом був створений гідравлічний комп’ютер MONIAC для моделювання економічних процесів Британської Імперії [12].

Рисунок 1.8 – Гідравлічний комп’ютер MONIAC

На сьогодні гідравлічні керуючі і обчислювальні елементи використовуються там, де потрібно розраховувати і керувати великими механічними зусиллями (наприклад, гідропривод у металургії), де застосування електронних засобів технічно недоцільне, зокрема, через вагу та об’єми обладнання, як це зустрічається у системах керування цивільної та військової авіації. Також, як окремі гідравлічні обчислювальні засоби можна відмітити звичайний автомобільний підсилювач керма (виконує множення механічного зусилля) та гідроамортизатор (інтегрує механічний імпульс).

1.4 Пневматичні обчислювальні машини і засоби

Широкий розвиток пневматика отримала з переходом суспільства до капіталістичних відносин. З появою парових машин, поршневих та турбінних

компресорів з’являється потреба у розробці клапанів різного призначення, регуляторів, накопичувачів тиску. Але розвиток саме обчислювальних машин, які використовують елементи пневмоніки, почався у 50…60-і роки XX століття.

Елементами пневматичної обчислювальної машини є дроселі, ємності і мембрани. Дроселі грають роль опорів, можуть бути постійними, змінними, нелінійними і регульованими. Пневматичні ємності представляють собою глухі або проточні камери, тиск в яких внаслідок стискання повітря зростає в міру їх наповнення. Мембрани використовуються для перетворення тиску повітря. До складу пневматичної обчислювальної машини можуть входити підсилювачі, суматори, інтегратори, функціональні перетворювачі і розмножувальні пристрої, які з’єднуються між собою за допомогою штуцерів і шлангів. В середньому рухливі елементи такої машини мають час спрацювання близько десятих часток мілісекунди, тобто вони можуть пропускати частоти 2..3 кГц [13].

Пневмоніка відрізняється значними погрішностями, тому застосовується там, де не можна застосовувати інші типи обчислювальних машин: у вибухонебезпечному середовищі, в умовах високих температур, в теплоенергетиці, газовій промисловості. Існують пневмоелементи з високою радіаційною стійкістю.

Сьогодні пневмокомп’ютери використовуються в галузях промисловості, де потрібна підвищена вібраційна стійкість, працездатність в дуже широкому діапазоні температур або потрібно керувати пневматичними силовими пристроями. В останньому випадку усувається необхідність в перетворювачах електричного сигналу в переміщення. Це – роботи і автоматика, що працюють в металургії, в гірничорудній промисловості. Відомі рішення для керування елементами авіаційних двигунів, автоматикою ракетних систем, силовими приводами вертольотів і літаків.

Рисунок 1.9 – Навчальний стенд Festo Didactic з елементами пневмоавтоматики

Існує також ціла категорія виробництв, агрегатів і установок, де застосування електрики небажано. Це хімія органічних сполук, нафтопереробні заводи, підземний видобуток вугілля і руди. Вони широко використовують пневматичну автоматику.

1.5 Оптичні обчислювальні машини і засоби

Історично першим оптичним обчислювальним елементом є сонячний годинник. Так, одним з найстаріших сонячних годинників є Стоунхедж в Англії. Він використовувався, у тому числі, для прогнозування змін пір року, а також сонячних та місячних затемнень.

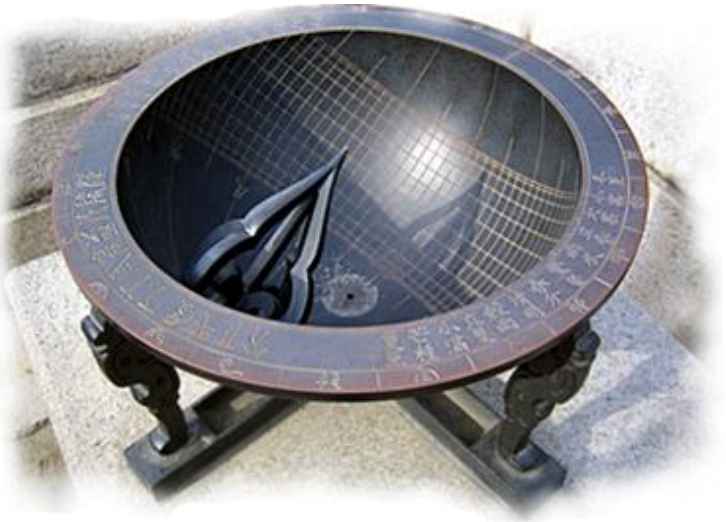

Рисунок 1.10 – Гномон древнього Китаю

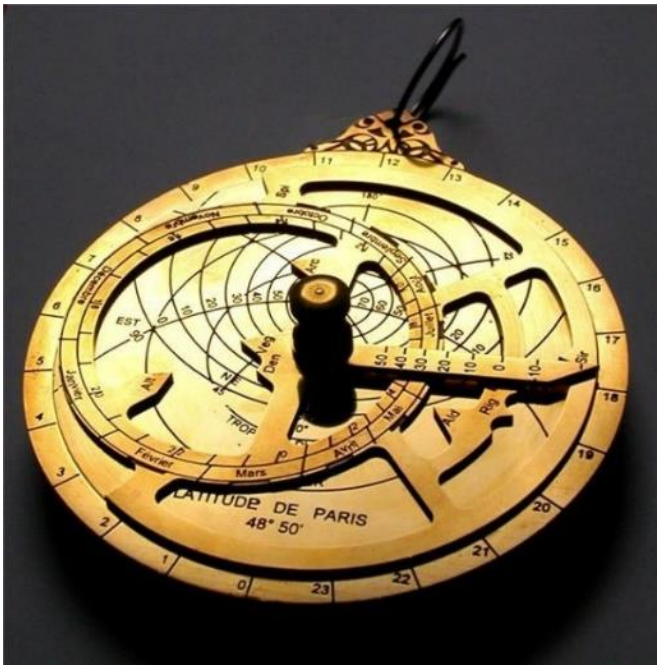

Рисунок 1.11 – Астролябія, зображення отримане з [15]

Сонячні годинники древнього Єгипту, Ассирії, Вавилону та Греції дозволяють визначити час і вдень, і вночі, спираючись на спостереження, що існує 12 сузір'їв, які двічі протягом доби з'являються на небі, а сонце як вдень, так і вночі знаходиться в одному з сузір'їв. Найдавніша згадка про сонячний годинник, так званий гномон, зустрічається у єгипетських рукописах 1521 р. до н.е., а в древньому Китаї гномон з'явився з VIII ст. до н.е. [14].

Серед оптичних обчислювальних засобів варто також згадати астролябію – прилад для визначення географічних координат.

Технічний прогрес у сфері обчислювальної техніки пішов іншим шляхом, тому сучасні оптичні засоби лише доповнюють електронні. Це перш за все оптичне волокно для передачі даних, відкриті оптичні канали, що складаються з випромінювачів та фотоприймачів, у тому числі побутові пульти дистанційного керування, елементи оптоелектроніки – фотореле, світлодіоди, оптопари та інше. Крім того, використовуються оптичні носії інформації – CD та DVD-диски.

1.6 Електричні обчислювальні машини і засоби

Представляють собою найбільш чисельний та поширений вид, який можна розділити на два підвиди: електричні, або електромашинні та електронні обчислювальні машини і засоби.

Стосовно електричних засобів, то тут варто згадати, що оперативна пам'ять обчислювальних машин у період з 50-х до 70-х років XX століття, до появи напівпровідникової пам'яті, була побудована на феритових кільцях або біаксах, що змінювали та зберігали напрям намагнічуваності.

Така пам'ять споживає досить багато електроенергії і виділяє забагато тепла. Але вона не боїться радіації та електромагнітного випромінювання, тому її продовжували і після 1970-х років використовувати у військовій та космічній сферах. Зокрема, у Шатлах феромагнітна пам'ять використовувалась до 1991 року [16].

Сучасні засоби постійної пам'яті продовжують використовувати технології електромашинних пристроїв. Зокрема накопичувач на жорсткому магнітному диску, хоча і містить електронні компоненти, використовує намагнічування феромагнітного покриття для зберігання даних. Крім того, існують змінні магнітні носії. Хоча дискети на сьогодні вийшли з ужитку, зберігання на магнітних стрічках великих обсягів даних використовується з 1951 року по сьогоднішній день і має перспективи у майбутньому. Так, сучасний картридж може мати обсяг до 300 терабайт [17]. Пристрій для роботи з магнітними стрічковими картриджами називається стример. Він використовується великими закладами для резервного копіювання.

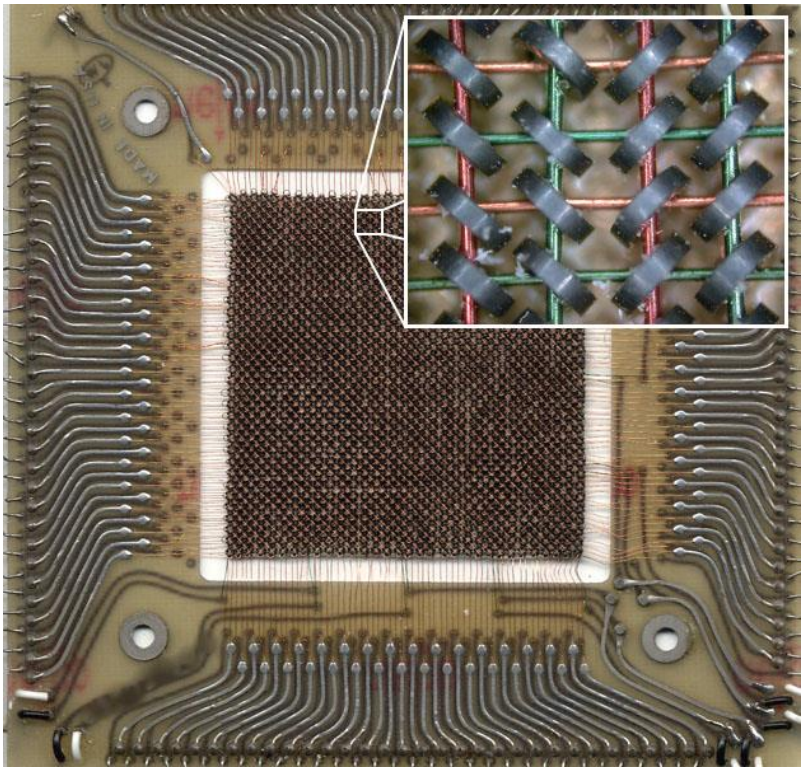

Рисунок 1.12 – Пам’ять на магнітних кільцях

Також згадаємо окремі електромашинні обчислювальні елементи. Це елементи безперервної дії: трансформатори, стабілізатори, різного виду магнітні підсилювачі, а також елементи дискретної дії: різні типи реле, пускачі, герконові кнопки тощо. Всі ці елементи використовуються здебільшого для початкового вводу даних або для видачі керуючих впливів як у виробництві, так і в інших сферах людської діяльності.

1.7 Електронні обчислювальні засоби і машини

Цей підвид розглянемо окремо, оскільки він найбільш широко використовується для обчислень. Відразу виділимо три підгрупи цих засобів: імпульсні, аналогові та цифрові.

1.7.1 Імпульсні обчислювальні засоби

Ці засоби варто вважати найповільнішими серед електронних і тому в якості повноцінних обчислювальних комплексів вони використовуються досить рідко. Але тут представлені різноманітні первинні лічильники, наприклад, матеріальних предметів та продукції, витратоміри газів та рідин, засоби передачі даних та керуючих впливів на основі частотних, фазних, широтно-імпульсних сигналів. Їх перевага – порівняно технічно нескладне узгодження цифрових обчислювальних засобів із рядом первинних джерел інформації та отримувачів керуючого впливу, таких як керований електропривід.

1.7.2 Аналогові обчислювальні засоби та машини

Такі засоби використовувались і використовуються зараз для нескладних

обчислень, де не потрібна висока точність. Їх перевага – найвища швидкодія серед електронних та можливість роботи з безперервними, аналоговими первинними джерелами інформації, хоча є можливість використання і для керуючого впливу, якщо є потреба задати його кількісну характеристику, наприклад, кут відхилення. Недолік аналогових засобів – невисока точність, зумовлена, перш за все, температурними змінами характеристик напівпровідникової, а рідше лампової, елементної бази. Для прикладу, згадаємо балістичні обчислювачі військової техніки, які зараз замінюються на цифрові, але все ще можуть використовуватись.

1.7.3 Цифрові обчислювальні машини і засоби

Це найбільш поширені засоби. Вони застосовуються при вирішенні основних обчислювальних задач людської діяльності там, де умови не обмежують їх застосування. Виділимо у цій групі дві підгрупи: цифрові ЕОМ діляться на **ЕОМ з жорстко заданою логікою** та **ЕОМ з програмно змінною логікою**.

Зараз ЕОМ з програмно змінною логікою витісняють ЕОМ з жорстко заданою логікою через швидкі темпи розвитку обчислювальної техніки, що вимагає постійного вдосконалення функцій без суттєвої зміни апаратної частини (маються на увазі найпростіші пристрої, такі як калькулятор, або плата керування пральною машиною). Тим не менш, для вирішення окремих задач, особливо при масовому виробництві жорстко задана логіка все ще використовується через економічну або технічну доцільність. Цифрові апарати з жорстко заданою логікою відрізняються більшою швидкодією порівняно з програмованими. На сьогодні розвиваються напрямки **БМК** (базових матричних кристалів) та **ПЛІС** (програмованих логічних інтегральних схем), де програмування представляє собою розробку цифрової електронної схеми, яка буде занесена на кристал і являтиме собою обчислювальний прилад з жорстко заданою логікою.

Сфера застосування ПЛІС:

- пристрої з великою кількістю портів введення-виведення;
- пристрої, що виконують цифрову обробку сигналу;
- цифрова відео- та аудіоапаратура;
- пристрої, що виконують передачу даних на високій швидкості;
- пристрої, що виконують криптографічні операції, системи захисту інформації;
- пристрої, призначені для проєктування інтегральних схем спеціального призначення;
- пристрої, що виконують роль комутаторів між системами з різною логікою і напругою живлення;
- нейрочіпи;
- пристрої, що виконують моделювання квантових обчислень.

Очевидно, що цифрові елементи з жорстко заданою логікою все ще будуть використовуватись.

Цифрові ЕОМ з програмно змінною логікою на сьогодні мають найширший ужиток. Вони використовуються там, де є доцільність їх використовувати. Сюди

слід віднести не лише персональні комп'ютери, планшетні пристрої, телефони. До цієї підгрупи також належать різні промислові та спеціалізовані контролери, програмовані елементи комп'ютерних мереж і систем зв'язку, інтелектуальні пристрої, у тому числі ті, що підключаються до Інтернету. Пристрої цифрового телебачення на сьогодні також програмовані і можуть програмуватись чи адмініструватись віддалено відділами телекомунікаційних компаній. Як вище згадувалось, ряд побутового та промислового обладнання також має інтегровані цифрові ЕОМ з програмно змінною логікою, що полегшує компаніям-розробникам їх модернізацію чи усунення недоліків в керуванні. Далі у посібнику буде розглядатись саме архітектура комп'ютерів та цифрових ЕОМ з програмно змінною логікою.

1.8 Висновки

Обчислювальні засоби розвивалися разом із цивілізаційним розвитком людства. Спочатку з'являлися потреби у підтримці прийняття рішень, організації обчислень, обліку, механізації чи автоматизації дій, вже потім – створювалися засоби, що забезпечують ці потреби. З іншого боку, розвиток обчислювальної техніки обмежувався рядом факторів, найсуттєвіші з яких:

1. Недосконалість методик зберігання та обробки даних, оскільки знання про час, відстань, інші фізичні величини, знання про системи числення і методи розрахунку з'являлися поступово.

2. Недосконалість технологій використання і обробки матеріалів, поступовий розвиток природничих і технічних наук.

Тим не менш, на сьогодні людство отримало комп'ютер у тому вигляді, яким він є. Було обрано шлях розвитку електронних обчислювальних машин з практично повною відмовою від досить розвинутих механічних обчислювальних засобів, пневматики та гідравліки. Рухаючись у бік уніфікації та здешевлення обчислювальних машин, було здійснено перехід до використання переважно методів цифрової електроніки і схемотехніки. Подальше використання ідеї створення програмованої логіки, раніше реалізованої у механічних засобах Жаккара, Бебіджа, Холеріта, а згодом і у гідравлічних моніторах Лук'янова та Філіпса, призвели до створення процесора та пам'яті – спочатку на окремих електронних та електромагнітних компонентах, а потім на основі кремнієвих кристалів.

1.9 Питання для самоперевірки

1. Чому первісні обчислювальні засоби не можуть розглядатись як один з етапів еволюції обчислювальних засобів і машин?

2. Чому ідеї Архімеда у механізації обчислень почали активно реалізовуватись лише у 17 столітті?

3. Які механічні обчислювальні машини мають архітектуру, найбільш наближену до сучасної ЕОМ і в чому це виявляється?

4. Які гідравлічні обчислювальні засоби використовувались у стародавньому світі і для чого?

6. Чому на Вашу думку, використання гідроустаткування для здійснення обчислень на сьогодні обмежене?

7. Які пневматичні елементи, що можуть використовуватись для обчислень, ви знаєте?

8. Які переваги та недоліки пневматичних елементів порівняно з електричними та електронними?

9. Які переваги та недоліки електричних елементів порівняно з електронними?

10. Як відрізняються задачі та області застосування для імпульсних, аналогових та цифрових обчислювальних засобів?

11. Вищу швидкодію мають цифрові ЕОМ з програмно змінюваною логікою чи з жорстко заданою?

12. Чому цифрові ЕОМ з програмно змінюваною логікою витісняють ЕОМ з жорстко заданою?

13. Чому пам'ять на феритових кільцях використовувалась на Шатлах до 90-х років XX століття?

14. Чому механічний замок можна прирівняти до механічної обчислювальної машини?

15. Які оптичні компоненти використовують сьогодні в комп’ютерній електроніці?

16. Які електромашинні та електромагнітні компоненти використовують у комп’ютерах загального призначення?

17. До якого типу обчислювальних машин слід віднести MONIAC?

18. У яких сферах сьогодні використовують пневматичні обчислювальні компоненти?

19. Яка сфера застосування ПЛІС?

20. Де і коли почали впроваджувати механічні обчислювальні машини з перфокартами для комерційних потреб та розрахунків?

21. Що Ви знаєте про хімічні обчислювальні засоби?

2 ІЄРАРХІЧНІ РІВНІ АРХІТЕКТУРИ КОМП'ЮТЕРІВ ТА ІНШИХ ЦИФРОВИХ ЕОМ

На основі поширеного підходу та відомих знань з архітектури комп'ютерів [18], у загальному випадку, комп'ютер або цифрову ЕОМ можна представити у вигляді ієрархії рівнів:

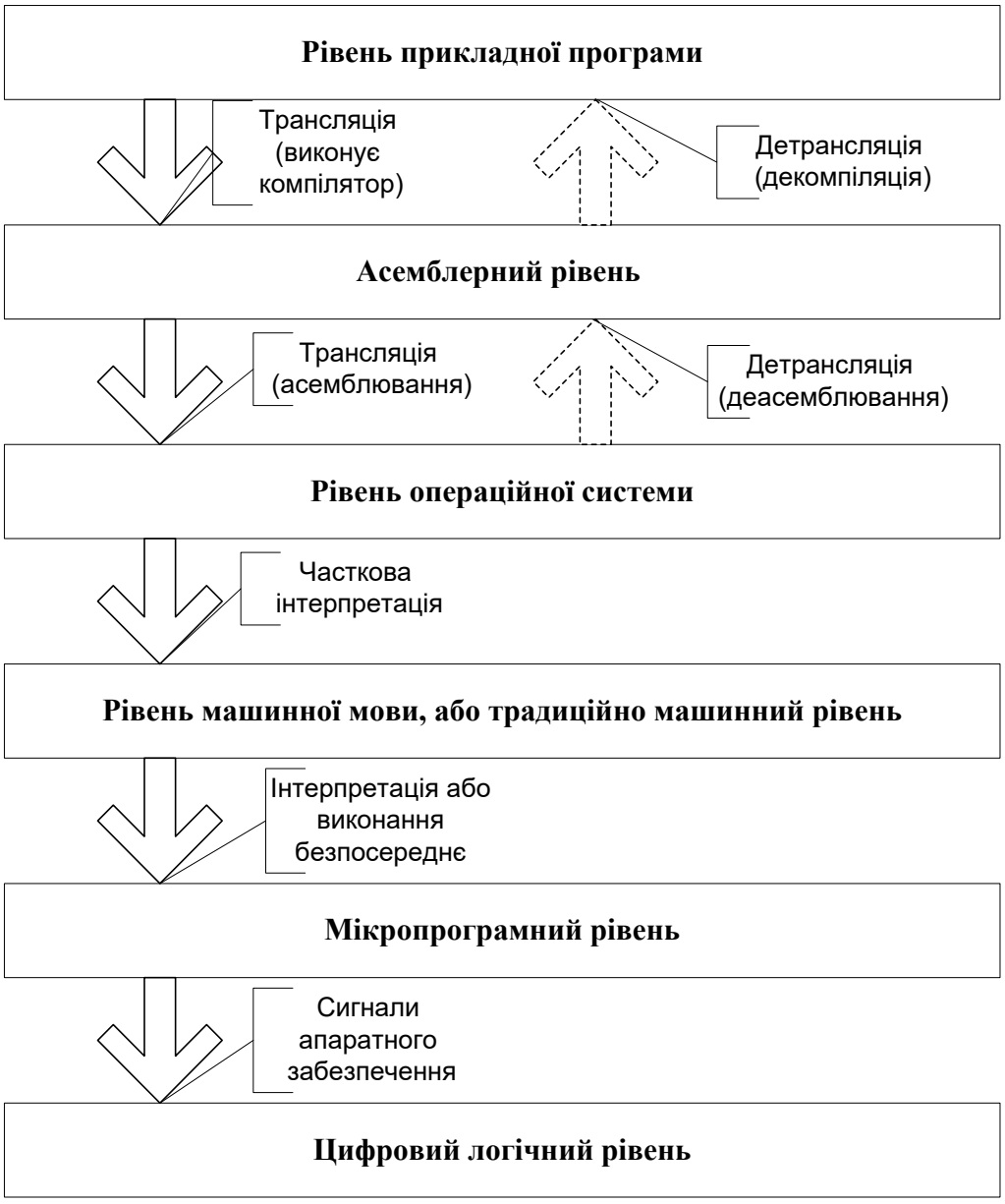

Рисунок 2.1 – Загальна багаторівнева архітектура комп'ютерів

Розглянемо кожен представлений рівень окремо.

2.1 Цифровий логічний рівень

Цифровий логічний рівень та мікропрограмний рівні іноді розглядають як один мікропрограмний рівень. Справді, на обох цих рівнях обробка даних зводиться до керування **електричними сигналами** за допомогою електронних

вентильних схем різного ступеню складності. Для цифрового логічного рівня найчастіше це **транзисторні схеми** або прості **логічні елементи** – такі елементи, що виконують прості логічні функції, наприклад «І», «АБО», «І-НЕ» тощо. Логічні елементи можуть бути присутні і на цифровому логічному, і на мікропрограмному рівні, тобто тут межа між рівнями нечітка. Докладніше робота вентильних схем і логічних елементів розглядається у курсах електроніки і схемотехніки, тому далі більше уваги буде приділятись мікропрограмному та рівню машинної мови. Єдине, на що варто звернути увагу – це на поняття сигналу і потоку сигналів.

Сигнал – у даному випадку це повідомлення, що має електричну природу, найчастіше виражений у вимірюваному рівні напруги (не завжди), про настання деякої події. Смисл цієї події є **інформаційною ознакою сигналу** і залежить від вирішуваних задач обладнанням більш високого – мікропрограмного рівня.

Потік сигналів – це група чи послідовність електричних сигналів, яка спрямована на електронні схеми цифрового логічного рівня для керування мікрооперціями процесора.

2.2 Мікропрограмний рівень

Цей рівень комп'ютера представлений у загальному випадку керуючим та операційним блоком компонент. Керуючий блок отримує із-зовні команди та видає керуючі сигнали на операційний блок, який із-зовні отримує дані для обробки, повертає результат обробки та передає на керуючий блок у вигляді зворотного зв'язку інформаційні сигнали про результат виконання команди. При цьому робота пристрою базується на наступних принципах:

- будь-яка машинна команда складається з послідовності елементарних дій над словами інформації – **мікрооперацій**;
- порядок проходження мікрооперацій залежить від значень перетворюваних слів, а також від їх інформаційних сигналів, вироблюваних операційним автоматом. Прикладами таких сигналів можуть бути ознаки результату операції, значення окремих бітів даних і т.п.;
- процес виконання машиною команди описується в вигляді деякого алгоритму в термінах мікрооперацій і логічних умов;
- **мікропрограма**, що є описом інформаційних сигналів, служить не тільки для обробки даних, а й забезпечує керування роботою всього пристрою в цілому. В цьому полягає принцип мікропрограмного керування.

На рівні мікропрограм існують апаратні композиційні компоненти, що забезпечують зберігання машинних слів, виконання над ними мікрооперацій, визначення логічних умов. Вони поділяються на шини, регістри, лічильники, сумматори, логічні пристрої, зсувачі, перетворювачі і формувачі кодів, комбіновані операційні елементи. До мікропрограмного рівня інтерпретуються «програми» рівня машинної мови. Тут інтерпретація є фактично виконанням потоку команд машинної мови.

Інтерпретація – це порядковий аналіз, обробка та виконання вихідного коду програми або запиту [19].

Терміни мікропрограмного рівня:

- **машинне слово** – це машино-залежна величина, що звичайно вимірюється в бітах або байтах, вона дорівнює розрядності регістрів процесора або розрядності шини даних. Як правило, це ступінь числа 2;

- **шина** – сукупність паралельно діючих провідників одного функціонального призначення, частіш за все шина призначена для одночасної передачі всіх бітів машинного слова;

- **регістр** – сукупність запам'ятовуючих елементів, які здійснюють паралельний прийом, зберігання, видачу бітів одного машинного слова або зсув даних в одному з двох напрямків по регістру машинного слова. В останньому випадку регістр називають **зсувачем**;

- **лічильник** – пристрій, що реалізує збільшення чи зменшення даних машинного слова на константу, наприклад, на одиницю;

- **суматор** – виконує додавання чи віднімання вмісту двох чи більше регістрів;

- **перетворювачі і формувачі** кодів – перетворюють один вигляд коду в інший. Наприклад, двійковий унітарний в двійковий позиційний. Такий перетворювач називається **дешифратором**. Важливо також згадати **пристрій двійково-десяткової корекції**, котрий перетворює звичайний двійковий код у десятково-двійковий, необхідний для відображення даних у звичній для людини десятковій системі числення на екранах, табло та інших засобах відображення;

- **комбіновані операційні елементи** – елементи, що реалізують кілька різнотипних операцій;

- **арифметико-логічний пристрій, АЛП** – операційний блок процесора, що представляє собою комбінований операційний елемент, котрий виконує операції над машинними словами, що містять дані.

2.3 Рівень машинної мови, або традиційно машинний рівень

Це рівень, до якого інтерпретуються програми рівня операційної системи або транслюються програми асемблерного рівня, якщо вони не використовують інтерфейс викликів функцій операційної системи. Програми цього рівня є потоком машинних слів, складених з наборів 0 і 1. **Інтерпретація може бути частковою** – тобто є деякі програми рівня операційної системи, що призначені для виконання прикладних обчислень, є заздалегідь підготовлені бібліотечні функції операційної системи, які виконують рутинні дії з керування апаратними ресурсами комп'ютера, є сама операційна система. І саме засоби операційної системи формують потік команд для машинного рівня забезпечуючи, за необхідністю, єдиний інтерфейс між прикладними програмами та апаратними ресурсами.

Інтерфейс – це спільний кордон між двома функціональними елементами, вимоги до якого визначаються стандартом, також це сукупність засобів, методів і правил взаємодії (управління, контролю і так далі) між елементами системи, що реалізовані у вигляді комунікаційних програмних чи апаратних модулів цієї системи чи комп'ютера.

Якщо розглядати комп'ютер на рівні машинної мови, то тут з'являються й інші нові поняття:

- процесор – це основний пристрій комп'ютера, що забезпечує обробку даних і зв'язок з іншими пристроями комп'ютера. Комп'ютер необхідно розуміти у більш широкому сенсі, як деякий обчислювальний вузол: персональний комп'ютер, мобільний телефон, маршрутизатор, промисловий контролер, система керування інтелектуальним побутовим чи промисловим пристроєм та інше. Сучасний процесор в основному являє собою надвелику інтегральну схему, що складається з елементів мікропрограмного рівня, причому інтегральна схема може об'єднувати і кілька процесорів, які у такому випадку називаються ядрами. Процесор може бути **основним** або периферійним, спеціалізованим на виконанні деякої задачі. До периферійних процесорів відносять **контролери шини**, **мосту**, **відео-, звукової чи мережної карти** та інші;

- потік традиційно машинного рівня – на цьому рівні термін означає потік машинних слів, що отримує процесор для обробки і повертає як результат;

- **пам'ять.** Існують різні види пам'яті як за призначенням, так і за способом реалізації, що буде розглянуто пізніше. Важливо розуміти, що пам'ять – це організована група регістрів, призначена для прийому, надання та зберігання машинних слів потоку (команд і даних) для процесора;

- **системна шина** здійснює прийом і передачу даних у вигляді потоку традиційно машинного рівня між вузлами комп'ютера;

- **порти введення-виведення** призначені для зв'язку комп'ютера чи іншого обчислювального вузла із зовнішнім світом, тобто периферійними пристроями. В загальному розумінні це може бути монітор, клавіатура, жорсткий диск, мережа. Тому з точки зору машинного рівня, порти мають регістри і виконують те ж саме, що і пам'ять за виключенням зберігання даних.

Традиційно машинний рівень є кінцевим рівнем, де представлене апаратне забезпечення комп'ютера, хоча і тут немає чіткої межі між ним та рівнем операційної системи. Далі, саме під час розгляду саме цього рівня, буде розглянута структура комп'ютера як системи, що об'єднує складні функціональні елементи.

2.4 Рівень операційної системи

Це програмний рівень, що відділяє апаратні рівні обчислювальної машини від програмних. До цього рівня транслюються програми вищого, асемблерного рівня.

Трансляція – це перетворення програми, представленої на одній з мов програмування, в програму на іншій мові і, в певному сенсі, рівносильну першої. Трансляція сама по собі не передбачає виконання коду. За напрямом переміщення коду по ієрархічній моделі розрізняють трансляцію та **детрансляцію**. Якщо вхідний трансльований код належить до більш високого рівня, вихідний до більш низького, то це трансляція, якщо навпаки – детрансляція. Трансляція – загальний термін, а для трансляції з асемблерного рівня до рівня операційної системи або нижче вживається і інший термін – **асемблювання**. Вихідний код, отриманий у результаті асемблювання, називають також **об'єктним**. Відповідно,

є і зворотня операція – **деасемблювання або дизасемблювання**. Операції детрансляції виконуються спеціалізованими програмами з метою відновлення вхідного коду за вихідним для подальшого аналізу фахівцями, наприклад, на предмет виявлення тіла нового вірусу. На рівні операційної системи існує організоване зберігання даних і виконуваного коду у вигляді файлів.

Варто зазначити, що у загальному розумінні **деінтерпретації програмного коду** не існує, оскільки сама інтерпретація пов'язана саме з його виконанням. Найближчою до деінтерпретації дією можна вважати процес записування макросів, наприклад, у MS Excel. Але й тут йде мова про запис дій людини-оператора, а не обчислювальної системи.

На рівні операційної системи розділяють поняття процес та потік.

Процес виконується під керуванням операційної системи і з точки зору одноядерного центрального процесору представлений як єдиний **потік команд** разом із самою операційною системою. Тобто, для процесора не має значення, обробляються у даний момент прикладні задачі чи системні. В багатоядерних системах обробка **потоків команд** організована складніше, але і вона визначається операційною системою. Багато авторів спеціальної літератури використовують термін «потік» у контексті процесу, нехтуючи уточненням, що це саме потік команд. Це іноді призводить до плутанини, оскільки існує окремий термін «потік», який має зовсім інше значення.

Потік, потік даних (англ. **stream**) – організоване сховище даних для обміну. Це може бути файл, через який два процеси у багатозадачній операційній системі обмінюються інформацією, канал передачі даних, порт введення-виведення.

Потік команд (англ. **thread**) – послідовність команд рівня операційної системи або традиційно машинного, який у багатозадачній операційній системі виконується послідовно. Інша назва потоку команд – **нитка**. Термін вживається порівняно рідко, хоча є більш точним перекладом з англомовних першоджерел. Варто зазначити, що у рамках одного процесу може виконуватись як одна, так і кілька ниток паралельно.

Процес – у нашому випадку достатньо розуміти цей термін як «програма, запущена на виконання». Хоча процес окрім програмного коду має власні дані, права доступу до апаратних та інформаційних ресурсів, пріоритетність, тощо. Більш докладно ці питання розглядаються у курсах з операційних систем та системного програмування.

Одне з призначень операційної системи – ізолювати програми вищих рівнів від особливостей апаратного забезпечення і звільнити розробника програмного забезпечення від необхідності програмування рутинних функцій керування апаратурою. Таким чином, рівень операційних систем надає вищим ієрархічним рівням єдиний інтерфейс доступу до пристроїв введення, виведення та іншої периферії. Інтерфейс може мати різне сервісне та програмне подання, спрямоване на полегшення його використання, але все зводиться до інтерфейсу системних викликів функцій введення, виведення, налаштування роботи апаратного

забезпечення. Як правило, функції оформлені у вигляді файлів, що входять до складу операційної системи.

Використання бібліотечних функцій програмами пов'язане з операцією зв'язування, яку виконує спеціальний компонент операційної системи – **лінкувальник**.

Зв'язування, лінкування (англ. **linking**) – формування виконуваного коду процесу з відасембльованого об'єктного коду прикладної програми, що може зберігатись у вигляді файлу та готового коду бібліотечної функції. Розрізняють **динамічне та статичне зв'язування**. Перше виконує лінкувальник операційної системи під час запуску програми. Друге – лінкувальник операційної системи чи окремий лінкувальник транслятора відразу після трансляції. Тобто, в останньому випадку отримаємо вихідний файл, який не потребує використання бібліотечних функцій операційної системи, оскільки вже включає їх у власний програмний код.

Розглядаючи рівень операційної системи, необхідно також згадати **використання віртуальних машин і процесорів**, як важливу на сьогодні частину системного програмного забезпечення. Для прикладу, візьмемо технологію Java. Програми, написані на Java компілюються у байт-код. Це код фактично традиційно машинного рівня, машинні слова якого призначені для Java-процесора. Java-процесор – складова операційної системи, яка дійсно є імітацією апаратного процесора і є інтерпретатором байт-кодів. Існують і апаратні реалізації процесорів, що підтримують байт-коди Java. Технологія спочатку впроваджувалась як одна зі складових концепції **мігруючих програм** – програм, що можуть розповсюджуватись через інтернет і виконуватись на комп'ютері-клієнті незалежно від типу і версії встановленої операційної системи. На сьогодні підтримуються й інші віртуальні процесори, або, як їх називають платформи: Qt, Python Platform, .NET та інші, в залежності від призначення обчислювального вузла.

2.5 Асемблерний рівень або рівень мови асемблера

На сьогодні не часто згадуваний, але важливий рівень ієрархічної архітектури комп'ютера. Тут не існує файлів, що містять виконуваний бінарний код, а є лише текстові, що включають зрозуміле для людини текстове описання програми на **мові асемблера**. Такі файли є вхідними для спеціальних програм – **асемблерів**, що виконують трансляцію у **об'єктний код** або код машинного рівня (за відсутністю у обчислювального вузла операційної системи). Сучасні компілятори, зокрема, компілятори C та C++ автоматично транслюють файли зі своїм вхідним кодом спочатку до мови асемблера, а потім, після можливої оптимізації коду, до рівня операційної системи.

Компіляція – це трансляція програми з проблемноорієнтованої мови на машинно-орієнтовану мову. Інакше кажучи, це трансляція програми рівня прикладних, проблемно-орієнтованих програм до рівня операційної системи. Власне, мова асемблера є машинно-орієнтованою. Тому програми на цій мові можуть бути написані більш оптимально з точки зору швидкодії або об'єму.

Не дивлячись на зрозумілу для сприйняття людиною форму подачі тексту програми, написання й відлагодження програм на мові асемблера – досить складний і витратний за часом процес. Тому сьогодні асемблером користуються лише у деяких випадках: при написанні драйверів, якщо для цього неможливо застосувати спеціалізовані засоби, для аналізу деасембльованого коду, для доводки програм.

Драйвер – це програма, за допомогою якої інші програми (зазвичай операційна система) отримують доступ до апаратного забезпечення деякого пристрою. Як правило, з операційними системами поставляються драйвери для ключових компонентів апаратного забезпечення, без яких система не зможе працювати.

Доводка програми – це переписування після проб і тестування на мові асемблера найбільш часто виконуваної частини програми з метою оптимізації коду. На сьогодні, з ростом продуктивності обчислювальної техніки та алгоритмів оптимізації початкових програмних кодів процес доводки може знадобитись досить рідко.

2.6 Рівень прикладної програми

На цьому рівні створюються програми на **мовах високого рівня** для прикладних та системних задач, які мають вирішуватись на ЕОМ. Програми тут в основному представлені у вигляді текстових файлів, які описують алгоритми та структури даних на мовах програмування високого рівня.

Системна задача – це задача, орієнтована на вирішення проблем операційної системи, у тому числі її взаємодії з людиною чи обладнанням. Написання системного програмного забезпечення здебільшого виконується також на рівні прикладної програми із, можливо, асемблерними вставками у високорівневий код. Останнє можна пояснити більшою придатністю до сприйняття людиною, а також більшою простотою мов високого рівня порівняно з асемблером.

2.7 Висновки

У процесі розвитку комп’ютер набув не лише сучасних фізичних форм реалізації. Удосконалення матеріалів, конструкцій та технічних рішень вкупі із ростом потреб людства у обробці і представленні даних призвели до розвитку перспектив, що змінили якості комп’ютера. З’явились більш складні абстракції і віртуальні рівні комп’ютера:

1. Цифровий логічний та традиційно машинний пов’язані з технічним оснащенням і вирішують питання автоматизації керування засобами процесора й комп’ютера під час вирішення системних чи прикладних задач.

2. Рівень операційної системи, котрий розділяє комп’ютер і користувача, надаючи останньому універсальний інтерфейс використання апаратних і програмних ресурсів комп’ютера, відносно незалежний від особливостей апаратної реалізації.

3. Асемблерний рівень, роль якого – представлення команд процесору і викликів функцій операційної системи у вигляді символьних команд, зрозумілих кваліфікованому фахівцю. На сьогодні для написання програм мова асемблера

(мова низького рівня) використовується рідко, хоча багато компіляторів, таких як C або C++, обов'язково транслюють вхідний код на мові прикладного рівня до рівня асемблера, а вже потім – до рівня операційної системи чи до традиційно машинного рівня.

4. Рівень прикладних програм, призначений виключно для програміста, де у зрозумілій для людини формі – мові високого рівня, орієнтованій, на відміну від мови асемблера, на людину, а не комп'ютер – описується вирішення прикладної чи системної задачі.

2.8 Питання для самоперевірки

1. Назвіть основні рівні архітектури комп'ютера. Які з них апаратні, а які програмні?
2. Що таке сигнал та потік сигналів?
3. Що таке мікропрограма?
4. Які терміни використовуються при розгляді мікропрограмного рівня комп'ютерів?
5. Навіщо використовується АЛП?
6. Навіщо використовується процесор?
7. Які бувають периферійні процесори?
8. Що таке системна шина?
9. Для чого використовують порти введення-виведення?
10. У чому різниця між трансляцією та інтерпретацією?
11. Потік традиційно машинного рівня, потік даних, нитка, процес – чим відрізняються ці поняття?
12. Що таке зв'язування і його які види використовуються?
13. Що таке асемблер?
14. Чому для вирішення системних задач частіше використовують мови високого рівня, ніж мови асемблера?

3 КЛАСИФІКАЦІЯ СУЧАСНИХ КОМП'ЮТЕРІВ

Класифікувати комп'ютери можливо за різними критеріями. Але з прагматичної точки зору, ЕОМ перш за все варто класифікувати за критерієм сфери їх використання. Тут можна виділити дві групи [20]:

1. Керуючі ЕОМ;
2. Універсальні ЕОМ.

3.1 Керуючі ЕОМ

Головна відмінність керуючих машин від універсальних ЕОМ полягає в особливості їх зв'язку з зовнішнім світом (керованим об'єктом). Дані надходять в машину безпосередньо від вимірювальних приладів або інших пристроїв, які фіксують характерні параметри об'єкта, сигнали управління також видаються машиною безпосередньо на об'єкт. Також керуючі машини повинні працювати в реальному масштабі часу, що пред'являє специфічні вимоги до їх програмного забезпечення та до апаратної архітектури машини.

До керуючих машин пред'являються більш високі вимоги по надійності, ніж до обчислювальних машин, але вимоги до точності знижені, оскільки вхідні дані все одно отримуються з певною похибкою. Представлення даних може бути обмежено машинним словом довжиною 16-32 двійкових розрядів, а не 32, 48, 64 розряду, як в універсальних ЕОМ [21].

Рисунок 3.1 – Промислові комп'ютери, зліва направо: малогабаритний ADVANTECH ITA-3630, портативний iROBO-4000, SIMATIC IPC547 для монтажу у стійку, панельний комп'ютер IEI PPC-F12B

До керуючих машин можна віднести як промислові керуючі комп'ютери, так звані **control computers**, так і промислові керуючі комп'ютери різних класів, так звані **comptrollers**. Сюди варто віднести також різні вбудовані системи, комп'ютерні засоби автомобілів, літаків, інших транспортних засобів.

Рисунок 3.2 – Промислові контролери, зліва направо: SIMATIC S7-300 з модулями вводу і виводу, DOS-сумісний ICP DAS i8000, панельний ICP DAS VP-23W1

Керуючими можна вважати також примітивні побутові вбудовані обчислювальні системи – мікропроцесорні плати керування мікрохвильовими печами, пральними машинами, дитячими іграшками, системи «розумний дім». Більшість з них, як власне і промислових контролерів використовують так звані однокристальні мікро-ЕОМ чи мікроконтролери. Оскільки це дійсно, хоч і з різноманітною вбудованою периферією, великі інтегральні схеми процесорів, в якості окремого класу ЕОМ їх розглядати не будемо.

3.2 Універсальні ЕОМ

Універсальні ЕОМ більш традиційні у загальному розумінні, можна розділити на [20]:

1. Персональні комп’ютери;

2. Комп’ютери масового обслуговування.

У свою чергу персональні комп’ютери розділяються на підгрупи:

1. Стаціонарні комп’ютери;

2. Переносні комп’ютери.

3.2.1 Персональні стаціонарні комп’ютери

Стаціонарні персональні комп’ютери останній час суттєво поступилися на ринку переносним, але використовуються як:

- офісні, які мають пристрої збереження та обробки даних, дисплей, звукові вихідні пристрої (колонки), клавіатуру, маніпулятор-мишу. Пристрої для збереження представлені вмонтованими у системний блок жорсткими дисками, DVD-ROM, зовнішніми жорсткими дисками та флеш-носіями;
- ігрові, в яких на відміну від офісних розширено можливості графічних і звукових контролерів. Ігрові комп’ютери є потужними системами, які за показниками продуктивності часто бувають потужнішими за офісні варіанти стаціонарних комп’ютерів. Можуть мати специфічну ігрову периферію – джойстик, руль, педалі;

Рисунок 3.3 – Офісний персональний комп’ютер

- робочі станції, які можуть, в залежності від призначення, мати значно більшу обчислювальну потужність, ніж офісні машини. Вони використовуються як автоматизовані робочі місця для управління у технологічних процесах разом чи замість промислових, для інженерних розрахунків, для автоматизованого проєктування. Частіше всього містить плату мережевого інтерфейсу, що дозволяє обмінюватися інформацією з сервером, іншими робочими станціями та іншими пристроями мережі. Останнім часом у якості робочих станцій або, як їх ще називають, комп’ютерів бізнес-призначення, можуть використовуватись так звані моноблоки – комп’ютери, де монітор і колонки конструктивно суміщені з системним блоком, окремі лише клавіатура та миша. Моноблоки займають менше місця і можуть працювати в умовах, відмінних від офісних – приміщеннях зі зниженою або підвищеною температурою, з підвищеною вологістю чи запиленістю, хоча вони за конструктивним виконанням не є промисловими;
- X-термінали, що представляють собою комбінацію бездискових робочих станцій і стандартних терміналів. Їм потрібен сервер, оскільки термінали працюють під керуванням розподіленої операційної системи.

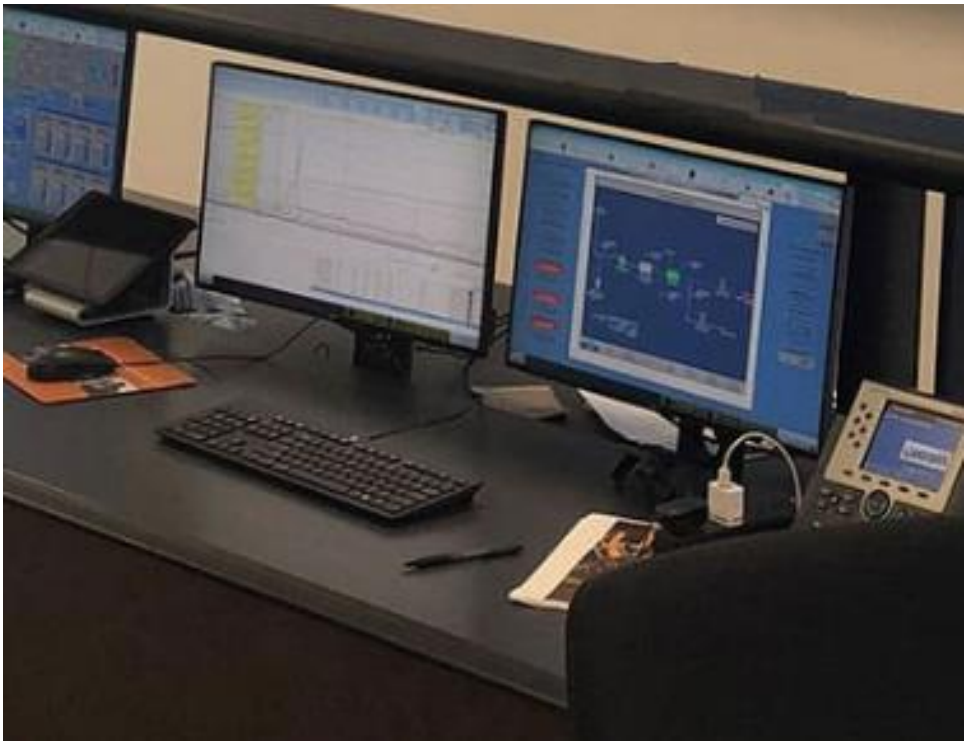

Рисунок 3.4 – Приклад робочої станції з трьома моніторами і додатковим обладнанням

3.2.2 Персональні переносні комп'ютери

До переносних та портативних персональних комп'ютерів віднесемо:

- ноутбуки, що можуть мати офісне, ігрове чи призначення в якості робочої станції і відрізняються тим, що всі їх компоненти розміщені в одному корпусі, що має розміри портфеля, зазвичай приблизно 14-27 дюймів за діагоналлю;
- нетбуки, які схожі на ноутбуки, але орієнтовані на роботу з Інтернетом. Тому вони мають порівняно нижчу продуктивність і більш компактні – приблизно 10-12 дюймів за діагоналлю. Зараз ця технологія вважається застарілою;
- планшети. Ці пристрої – дещо середнє між нетбуком та смартфоном. Мають технічні засоби для підключення до бездротової мережі, сенсорний екран діагоналлю близько 10 дюймів, призначені для переглядання відео та користування Інтернетом, тому їх продуктивність відповідає продуктивності нетбуків, що дозволило витіснити з ринку останні. Можна підключати зовнішню клавіатуру для планшетів;
- кишенькові переносні комп'ютери (застаріла технологія), смартфони та айфони. Основна їх перевага – можливість носити у кишені. КПК зараз майже не використовуються, а смартфони та айфони порівняно з ноутбуками менш продуктивні. Проте, останні цілком достатні для виходу в Інтернет і поєднують у собі інші повсякденно необхідні функції – мобільний зв'язок, а також можливість отримувати й обмінюватись фото та відеоінформацією. Така перевага, в свою чергу, дозволяє витіснити з ринку планшети.

3.2.3 Комп’ютери масового обслуговування

До цього класу комп’ютерів можна віднести:

1. Сервери;
2. Мейнфрейми;
3. Кластерні системи;
4. Суперкомп’ютери.

Сервери – це прикладні багатокористувацькі комерційні та бізнес-системи, що включають системи управління базами даних та обробки транзакцій, великі видавничі системи, мережеві додатки та системи обслуговування комунікацій, розробку програмного забезпечення та обробку зображень. Вони використовують модель обчислень «клієнт-сервер» і розподілену обробку даних. Сервер – це потужний мережевий комп’ютер, центр мережі, сховище даних, що використовує мережеву операційну систему, спеціалізоване системне та прикладне програмне забезпечення.

Рисунок 3.5 – Сервер Intel MFSYS25, що монтується у стійку

Мейнфрейми – це багатопроцесорні системи, що містять один або кілька центральних і периферійних процесорів із загальною пам’яттю, пов’язаних між собою високошвидкісними магістралями передачі даних.

Мейнфрейми найбільш потужні (не враховуючи суперкомп’ютери) обчислювальні системи загального призначення, що забезпечують безперервний цілодобовий режим експлуатації.

Стрімке зростання продуктивності персональних комп’ютерів і серверів створив тенденцію переходу з мейнфреймів на комп’ютери менш дорогих класів та організацію кластерних мереж. Ця тенденція отримала назву «розукрупнення» (downsizing). Але мейнфрейми все ще використовуються.

Рисунок 3.6 – Мейнфрейм

Кластерні комп'ютерні системи – саме ці засоби, покликані на заміну мейнфреймів. Це група, іноді досить велика, незалежних обчислювальних машин, що використовуються спільно і працюють як одна система для вирішення тих чи інших задач, наприклад, для підвищення продуктивності, забезпечення надійності, спрощення адміністрування тощо. Обчислювальний кластер потрібен для збільшення швидкості обрахунків за допомогою паралельних обчислень [26].

Кластери орієнтовані перш за все на роботу з базами даних у режимі високої доступності та здійснення паралельних обчислень на великій кількості комп'ютерів за аналогією з багатопроцесорним кластером. Як правило, кластери працюють під керуванням **розподілених операційних систем** або **мережевих у режимі розподілених**. Відмінність перших від других у тому, що користувач працює з мережевими ресурсами так само, як з локальними без виконання особливих дій, притаманних саме роботі з мережею.

Варто зазначити, що кластери є перш за все комп'ютерними системами і дещо виходять за рамки курсу «Архітектура комп'ютерів». Тому далі у цьому посібнику розглядатись не будуть.

Рисунок 3.7 – Лінукс-кластер Хемніцького технологічного університету

Суперкомп'ютери – це професійні машини з найбільш високою на сьогоднішні дні продуктивністю, вони використовуються в наукових лабораторіях і великому бізнесі. Такий пристрій являє собою комплекс комп'ютерних пристроїв, який може займати величезні приміщення.

Рисунок 3.8 – Суперкомп'ютер

Якщо ви щось чуєте про складне моделювання багатоаспектних процесів, наприклад, у генній інженерії, геологічних процесах, процесах у ядерній фізиці, прогнозування природних катастроф, то такий прогноз напевно був сформований за допомогою використання суперкомп'ютера.

3.3 Висновки

Поняття сучасного комп'ютера об'єднує досить широкий перелік пристроїв і засобів. Електронна обчислювальна техніка вирішує широкий спектр задач: автоматизація документообігу, обліку і статистики, прогнозування явищ і процесів різного ступеню складності, медіа та комунікації, підтримка прийняття рішень у виробництві, громадському житті, контроль і керування виробничим, транспортним, іншого роду обладнанням, технологічними операціями, процесами й підприємствами, обслуговування потреб людини у побуті, розвагах, повсякденному житті. Різноманіття задач, покладених на сучасний комп'ютер у широкому розумінні призвело до його розділення на певну класифікаційну структуру перш за все за призначенням. Саме призначення комп'ютерів призводить до різноманіття їх технічної реалізації, відмінностей у периферійних пристроях і операційних системах. Дійсно, операційні системи за своїми задачами і функціональними можливостями відрізняються для офісного комп'ютера, робочої станції, промислового контролера та кластерної системи. Однокристальні мікрокомп'ютери приладів можуть взагалі не мати операційної системи, лише керуюче програмне забезпечення. Тим не менш, спостерігається деякий загальний підхід до організації ЕОМ та її основної частини – системної плати. Цьому питанню присвячено наступний розділ.

3.4 Питання для самоперевірки

1. За сферою використання на які групи розділяються ЕОМ?
2. У чому полягають основні відмінності між керуючими і універсальними комп'ютерами?
3. Чим, на Вашу думку, відрізняється промисловий комп'ютер ADVANTECH ITA-3630 від офісного?
4. Чи завжди потрібен монітор для промислового комп'ютера?
5. У чому полягають відмінності між офісним комп'ютером, ігровим та робочою станцією?
6. У чому полягають особливості конструктиву і використання моноблоків?
7. Чим відрізняються X-термінали від інших персональних стаціонарних комп'ютерів?
8. Для чого можуть використовуватись ноутбуки?
9. На Вашу думку, чому в нетбуки вважаються застарілою технологією?
10. Чому смартфони та айфони поступово заміщують планшети?
11. Що таке сервер та для чого він використовується?
12. Що таке мейнфрейми та чому їх використання скорочується?
13. Що собою являють кластерні комп'ютерні системи?
14. Що таке суперкомп'ютери та де вони застосовуються?

4 СТРУКТУРА МАТЕРИНСЬКОЇ ПЛАТИ

4.1 Загальна структура системної плати

Необхідний набір зовнішніх пристроїв, що забезпечує можливість роботи користувача з комп'ютером, називається базовою **конфігурацією комп'ютера.**

Для персонального стаціонарного комп'ютера базова конфігурація включає:

- системний блок;
- блок живлення;
- жорсткий диск або кілька, для зберігання програм та даних;
- монітор;
- клавіатуру та мишу для керування комп'ютером.

Базова конфігурація інших типів ЕОМ може відрізнятись. Для прикладу, сервер чи промисловий контролер може не включати до свого складу монітор, клавіатуру, мишу. Контролер зазвичай замість жорсткого диску містить внутрішню енергонезалежну пам'ять чи зовнішню картку. Сервери, мейнфрейми, суперкомп'ютери містять **Raid-масиви** – пристрої, що складаються з групи жорстких дисків під керуванням загального контролера. Комп'ютери масового обслуговування включають групу терміналів, які складаються з моніторів, клавіатур, мишей. Блок чи система живлення присутня завжди у тій чи іншій формі. Системна плата у тій чи іншій формі також завжди присутня.

Найбільш загальна структура обчислювальної машини показана на рисунку 4.1 [18, 20].

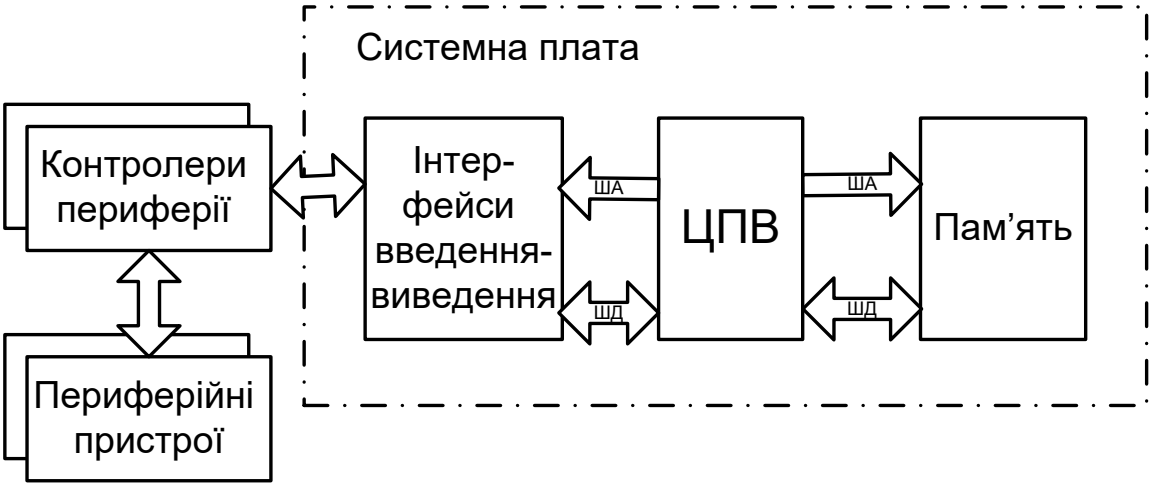

Рисунок 4.1 – Найбільш загальна структура обчислювальної машини

Як видно з рисунка 4.1, обчислювальна машина містить системну плату з центральним процесорним вузлом. **Системна плата**, іноді звана «материнською платою» (motherboard) – основна плата ЕОМ. Саме за допомогою елементів системної плати відбувається взаємодія між розрізненими пристроями, складовими комп'ютера.

До **центрального процесорного вузла** ЦПВ входять процесор або група (ядро) процесорів, обладнання тактового генератора, буфери узгодження зв'язку і синхронізації із **системною шиною**. Найчастіше процесор сучасного персонального чи керуючого комп'ютера є надвеликою інтегральною схемою, що має назву **мікропроцесор**. Використовуючи системну шину, ЦПВ читає з пам'яті команди, розшифровує їх та керує роботою всіх елементів комп'ютера. Рисунок 4.1 не розкриває всі особливості процесорного вузла. Більше того, оскільки структура загальна, то тут може бути і кілька процесорів.

Окрім процесора до складу ЦПВ можна віднести і **тактовий генератор** – окрему або вбудовану електронну схему, яка виробляє тактуючі імпульси, або тактові сигнали певної частоти. Тактовий генератор слугує для приведення у дію ЦПВ та для синхронізації його роботи з іншими складовими комп'ютера. Сам ЦПВ може працювати на більш високій тактовій частоті, ніж частота тактового генератора. Внутрішню частоту отримують зазвичай множенням зовнішньої за допомогою спеціальних схемотехнічних рішень всередині процесора.

Системна шина на системній платі включає дві складових: **однонаправлену шину адреси та двонаправлену шину даних**. Через системну шину ЦПВ взаємодіє з пам'яттю обчислювальної машини та портами введення-виведення.

Пам'ять **ЕОМ** служить для зберігання програм та даних поточних обчислень. Вона може бути єдиною для програм і даних. Таке рішення має назву **Прінгстонська, або фон-Нейманівська архітектура** [22]. Вважається, що сучасний універсальний комп'ютер має фон-Нейманівська архітектуру. І навпаки, можуть бути відокремлена пам'ять даних і відокремлена пам'ять програм. Це **Гарвардська архітектура** [18], яка притаманна однокристальним мікро-ЕОМ, які використовуються здебільшого для вбудованих керуючих комп'ютерних засобів.

Пам'ять ЕОМ, встановлена на системній платі, складається з окремих комірок, у кожній комірці зберігається одне машинне слово. Усі комірки пам'яті пронумеровані. Номер комірки називається її **адресою.** При читанні слова з комірки пам'яті за допомогою ЦПВ на шину адреси встановлюється її адреса. Потім, під впливом імпульсів тактового генератора, копія машинного слова електронною схемою пам'яті виставляється на шину даних і надходить до ЦПВ. Тобто зчитування передбачає не перенесення даних з пам'яті, а копіювання. Під час запису у пам'ять і адреса, і машинне слово даних одночасно виставляється з боку ЦПВ на шину адреси і шину даних відповідно.

Пам'ять ЕОМ можна розділити на постійну та оперативну. У постійній пам'яті зберігаються програми, що забезпечують роботу комп'ютера чи контролера після його увімкнення, та виконують початкову перевірку працездатності комп'ютера. Тут же зберігаються дані, які не змінюються у процесі експлуатації. Постійна пам'ять використовується лише у режимі читання інформації. Оперативна пам'ять навпаки, містить програми і дані, що змінюються у процесі роботи процесора і, відповідно, вона доступна за читанням та записом. Докладніше пам'ять ЕОМ буде розглянуто в окремих розділах.

Інтерфейси введення-виведення забезпечують обмін даними по системній шині між ЦПВ та оточенням, представленим контролерами периферії [18].

Контролери периферії можуть бути інтегровані на ту ж саму системну плату, а можуть бути реалізовані у вигляді окремих плат.

Контролери периферії, як правило, – це окремі міні-комп'ютери з власним ЦПВ, пам'яттю і портами вводу-виводу. Це можуть бути мережеві, відео чи звукові карти, контролер жорсткого диску, контролер переривань, а для промислових контролерів – модуль введення чи виведення, що підключається до материнської плати через крос-плату, шинний роз'єм з кріпленням або монтажну корзину. Доступ до портів введення-виведення, а також до регістрів налагодження контролерів периферії зазвичай організовано за тими ж принципами, що й до пам'яті. Дані портів та регістрів можуть змінюватись без участі ЦПВ, оскільки тут є й інші актори: процесори контролерів, зовнішні комунікації й периферія.

Периферійні пристрої можуть бути інтегровані у материнську плату (календар і годинник реального часу), вмонтовуватись у системний блок (жорсткий диск, вбудовані динаміки), або розміщуватись окремо (монітор, клавіатура, принтер). Можуть бути зовнішні мережні комунікації з іншими комп'ютерами, контролерами, мережними пристроями, інтелектуальним обладнанням. Кожен пристрій, що підключається до шини, повинен забезпечуватися логічною схемою (адаптером), що адаптує пристрій до правил роботи з шиною.

Периферійні пристрої посилають ЦПВ запити на власне обслуговування. Ці запити називаються **апаратними перериваннями або BIOS Hardware Interrupts**. Є апаратні переривання від таймера, клавіатури, дисків. Зазвичай апаратні переривання є асинхронними і незалежними відносно роботи ЦПВ подіями окрім переривання помилки через ділення на нуль. Момент ділення на нуль можна визначити заздалегідь, оскільки він залежить від виконуваної програми. Не варто плутати апаратні переривання з **програмними перериваннями**. Програмні переривання насправді не є перериваннями у загальному сенсі, це заздалегідь визначені функції операційної системи, які можуть бути використані програмою шляхом їх виклику.

Програмні переривання, що використовуються для звернення до системних пристроїв забезпечуються **базовими функціями BIOS** (сервісами), які називаються **ROM BIOS Services**.

Апаратні переривання обслуговує **контролер переривань**. Він приймає запит на переривання від зовнішніх пристроїв, визначає рівень пріоритету цього запиту і видає сигнал переривання в ЦПВ. ЦПВ, одержавши цей сигнал, припиняє виконання поточної програми і переходить до виконання спеціальної програми обслуговування того переривання, яке запросило зовнішній пристрій. Після завершення програми обслуговування відновлюється виконання перерваної програми [27]. Контролер переривань має заздалегідь завантажені **вектори переривань** – адреси початку згаданих вище програм, що мають назву **обробники переривань**.

Розглянута на рисунку 4.1 структурна схема системної плати має шинну архітектуру. Тут на системній платі розташовано контролер прямого доступу до пам'яті, або **контролер DMA**. Такий контролер сам керує пересиланням блоків даних від пристрою безпосередньо у пам'ять, не залучаючи до цього процесора.

Блоки даних зазвичай набагато більші за розрядність процесора. Наприклад, їх довжина може бути 4 КБайт [27].

Отже, маємо режим **DMA**, **Direct Memory Access**, **прямий доступ до пам'яті** – це режим обміну даними між пам'яттю і пристроєм введення/виведення без участі ЦПВ. Варто зазначити, що для DMA не всі архітектури використовують контролер DMA, наприклад у сучасних архітектурах персональних комп'ютерів ця задача вирішується інакше.

Використання цього режиму значно прискорює пересилання даних, тому що виключає пересилання даних між пристроєм введення/виводу (наприклад, це жорсткий диск) та оперативною пам'яттю через процесор. Обмін даними здійснюється на запит зовнішнього пристрою на прямий доступ до пам'яті.

4.2 Шинно-мостова архітектура системної плати персонального комп'ютера

У завдання чипсетів для 80286/386 входили ув'язка шини процесора з відносно нескладним контролером пам'яті і підключення до цього зв'язування шини (E)ISA, на якій розташовувалися всі пристрої. Поступово стала ускладнюватися підсистема пам'яті – з'явився кеш на системній платі, а потім до нього додався вбудований кеш процесора. Для процесорів класу 486 продуктивності шини (E)ISA виявилося вже недостатньо, і з'явилися нові шини, зокрема шини VLB, вона ж VESA. Згодом з'явилася шина PCI, для якої довелося будувати міст від системної шини. Спочатку її називали «прибудованою» (mezzanine bus), але незабаром вона надовго стала центральною шиною, навколо якої установлювалися всі інші елементи. Традиційно на схемах шину PCI зображують посередині, як екватор. Процесор і пам'ять (разом з кеш-пам'яттю) зображують вище – «північ», а шину ISA і всі пристрої, що підключаються до PCI і ISA, зображують нижче – «південніше екватора». Відповідні частини чіпсета отримали вкорінені назви північних (north) і південних (south) [28].

Таким чином, у шинно-мостовій архітектурі є центральна магістральна шина, до якої інші компоненти підключаються через мости. У ролі центральної магістралі спочатку виступала шина (E)ISA, потім її змінила шина PCI. Шина PCI в ролі головної магістралі втрималася недовго: відеокартам з 3D-акселератором її пропускної здатності, що розділяється між всіма пристроями, виявилося недостатньо.

Тоді і з'явився порт AGP як виділений потужний інтерфейс між графічним акселератором і пам'яттю (а також процесором). При цьому завдання північного мосту ускладнилися: контролеру пам'яті доводиться працювати вже на три фронти - йому посилають запити процесор(и), майстри шини PCI (і ISA, але теж через PCI) і порт AGP. Пропускна здатність AGP в режимі 2x/4x/8x становить 533/1066/2133 Мбайт/с, так що шина PCI по продуктивності стала вже другорядною. Однак у шинно-мостовій архітектурі вона зберігає свою роль магістралі підключення всіх периферійних пристроїв (крім графічних). В якості потужного прикладу шинно-мостової архітектури можна розглядати чіпсет AMD-760, де

реалізовано первинну шину PCI на 64 біт і 66 МГц, що є «екватором», і вторинну шину для підключення периферії [28].

Розглянемо більш докладно шинно-мостову архітектуру системної плати. Вона визначається набором мікросхем (**chipset**). Це одна або кілька мікросхем, таймери, система управління спеціально розроблена для взаємодії з мікропроцесором. Вони містять у собі контролери переривань, прямого доступу до пам'яті. Зазвичай в одну з мікросхем набору входять також годинник реального часу з CMOS-пам'яттю і іноді контролер клавіатури, однак ці блоки можуть бути виконані окремо [23].

Головними структурними елементами системної плати є так звані північний і південний мости. Існує певна плутанина між шинно-мостовою і хабовою архітектурою, тому наведений рисунок 4.2 дещо скориговано порівняно з першоджерелом [23] у бік шинно-мостової архітектури [28].

Північний міст, Northbridge служить для швидкісного зв'язку між процесором, оперативною пам'яттю і відеоадаптером, підключеним до високошвидкісної шини (PCI-E або AGP).

Південний міст, Southbridge призначений для зв'язку з контролерами портів і периферійних пристроїв. Також на південному мосту знаходиться BIOS і контролери пристроїв введення-виведення (I/O, Input/Output devices).

Мости з'єднані між собою внутрішньої шиною, яка забезпечує зв'язок процесора з периферійними пристроями.

Загальна структурна схема системної плати представлена на рисунку 4.2 і з боку північного мосту містить позначення:

CPU – центральний процесорний вузол;

Clock Generator – тактовий генератор;

AGP (Accelerated Graphics Port) – прискорений графічний порт, спеціалізована 32-розрядна системна шина для відео карти;

Memory Slots – слоти для встановлення плат RAM (Random Access Memory), або **ОЗП**, оперативних запам'ятовуючих пристроїв. В ряді моделей, зокрема для ноутбуків, таких слотів може бути лише два. ОЗП слугує для оперативного розміщення машинного коду операційної системи, програм і їх даних. Тому персональні, а насправді, і універсальні, і більшість управляючих комп'ютерів можна вважати машинами Прінгстонської архітектури. З іншого боку, є BIOS, де знаходиться окремий програмний код, який завантажує операційну систему до ОЗП. З цієї точки зору згадані різновиди комп'ютерної техніки можна віднести і до Гарвардської архітектури. Більше того, якщо у операційної системи використовується чисто сторінкова, одновимірна модель віртуальної пам'яті процесу, то це Прінгстонська модель управління обчисленнями, а якщо сегментна, де код відокремлено від змінних – Гарвардська. Таким чином, строго віднести сучасні комп'ютери до моделі фон-Неймана не можна, тут все буде залежати від точки і мети аналізу обчислювальної системи.

Щодо шини PCI. **PCI (Peripheral component interconnect, з'єднання зовнішніх компонент)** – для шинно-мостової архітектури це взаємозв'язок між північним та південним мостом, а також шина введення-виведення для

підключення периферійних пристроїв до материнської плати. Шина замінила собою застарілі MCA, ISA, VESA. Це надало ряд переваг при конструюванні комп'ютера:

- найбільші виробники апаратного забезпечення персональних комп'ютерів підтримують і просувають стандарт PCI. Шина універсальна і використовується не тільки для відеокарт, але і для інших пристроїв;
- стандартна тактова частота, що полегшує задачі виробників контролерів;
- стандартний роз'єм;
- синхронізація доступу. Шина захищає зв'язки між процесором і пам'яттю і дозволяє багатьом пристроям використовувати одну локальну шину;
- повна підтримка Plug-and-Play.

Шина PCI має наступні технічні характеристики:

- розрядність - 32/32 (розширений варіант - 64/64);
- тактова частота - до 33 МГц (PCI 2.1 - до 66 МГц);
- пропускна здатність - до 132 Мб/с (264 Мб/с для 32/32 на 66 МГц і 528 Мб/с для 64/64 на 66 МГц);
- підтримка автоконфігурації.

Роз'ємів шини на одному сегменті може бути до чотирьох. Вони з'єднуються один з одним за допомогою мостів (bridge). Сегменти можуть об'єднуватися в різні топології (дерево, зірка і т.п.). Зараз найпопулярніша шина, яка використовується також на комп'ютерах, відмінних від IBM-сумісних. Всі роз'єми і карти до них діляться на такі, що підтримують рівні сигналів 5 В, 3,3 В і універсальні. Окрім наведеного, варто зазначити, що шина PCI використовується як канал зв'язку між північним і південним мостами.

Застаріла альтернатива PCI – **шина PCMCIA** (Personal Computer Memory Card International Association, асоціація виробників плат пам'яті персональних комп'ютерів) – зовнішня шина комп'ютерів класу NoteBook. Інша назва модуля PCMCIA – PC Card. Гранично проста, розрядність – 16/26 (адресний простір – 64 Мб), підтримує автоконфігурацію, можливе підключення і відключення пристроїв в процесі роботи комп'ютера. Конструктив – мініатюрний 68-контактний роз'єм. Контакти живлення зроблені більш довгими, що дозволяє вставляти і виймати карту при включеному живленні комп'ютера [24].

Більш сучасний варіант PCMCIA – **PCI Express.** Відрізняється більш високою швидкодією (до 128 ГБ/с для PCI Express 6.0) і підтримкою USB 2.0 [25].

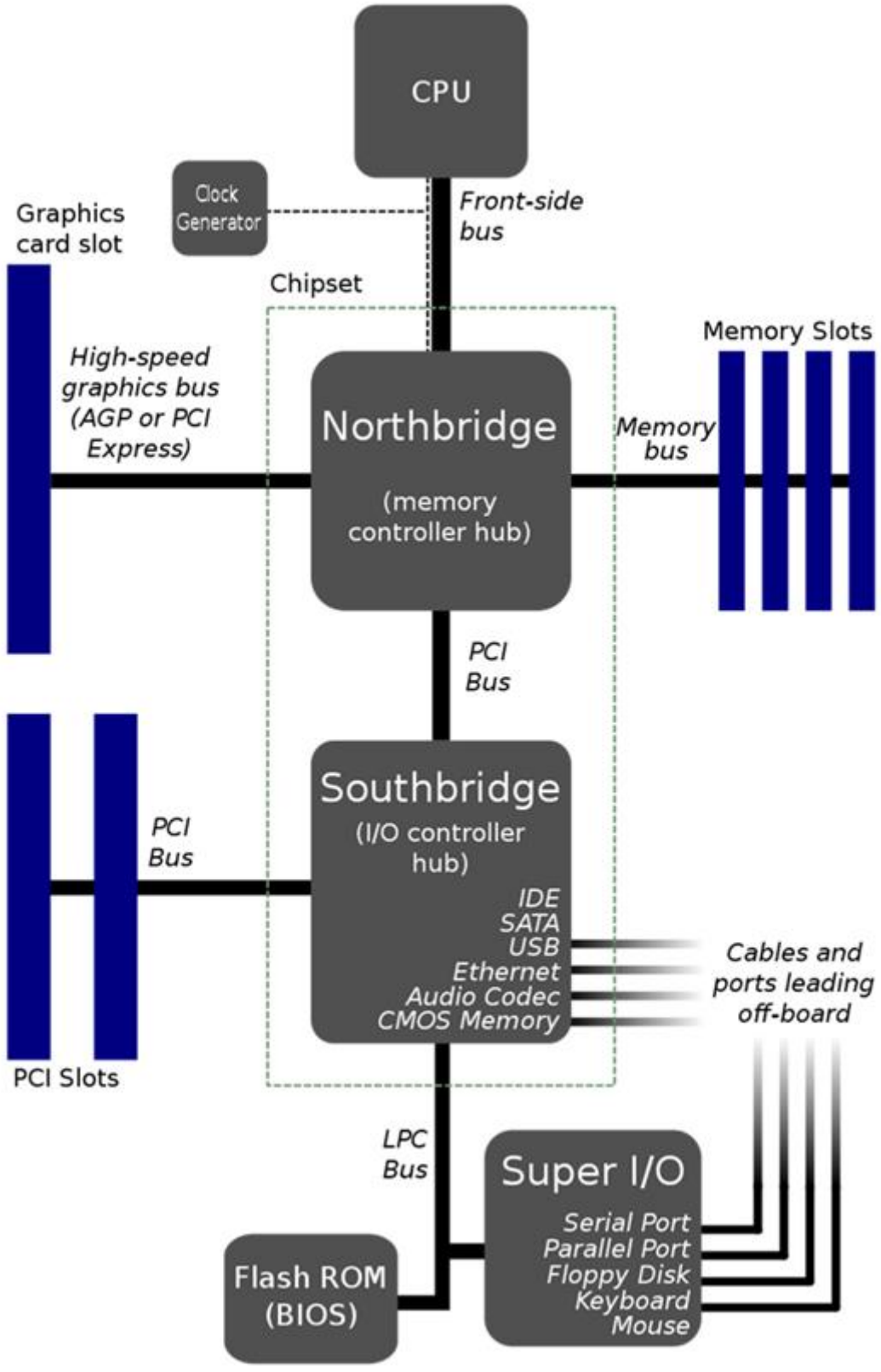

Рисунок 4.2 – Загальна структурна схема системної плати персонального комп’ютера з шинно-мостовою архітектурою

Південний міст об’єднує важливі порти для підключення додаткових модулів розширення функціональних можливостей комп’ютера. Для цього служать роз’єми (слоти розширення), встановлені на системній платі комп’ютера. До південного мосту відносять наступні компоненти:

- **IDE (Integrated Drive Electronics) або ATA (Advanced Technology Attachment)** – інтерфейс підключення накопичувачів (жорстких дисків і оптичних дисководів) до комп’ютера;
- **SATA** – більш сучасна версія IDE;
- **USB (Universal Serial Bus)** – універсальна послідовна шина, послідовний інтерфейс передачі даних для периферійних пристроїв в обчислювальній техніці;
- **BIOS** на рисунку 4.2 позначена ще як **Flash ROM** – це ROM (Read Only Memory), або ПЗП, постійний запам’ятовуючий пристрій. Містить базову систему введення-виведення, тобто набір мікропрограм, які реалізують інтерфейс для роботи з апаратурою комп’ютера і підключеними до нього пристроями.

Система BIOS виконує у комп’ютері три основні функції:

– здійснює ініціалізацію та початкове тестування апаратних засобів, тобто перевіряє роботу вузлів комп’ютера під час його включення, виконує процедуру тестування POST (Power On Self Test);

– відшукує операційну систему, завантажує їх у оперативну пам’ять і передає їй управління комп’ютером;

– після завантаження операційної системи BIOS надає прикладним програмам доступ до окремих блоків комп’ютера.

Система BIOS пов’язана з елементом **CMOS (complementary metal-oxide-semiconductor, комплементарна структура метал-оксид-напівпровідник)** – мікросхема, де зберігаються зміни, зроблені в налаштуваннях BIOS. Це постійна перепрограмована пам’ять. У системі BIOS є програма Setup, яка може змінювати вміст пам’яті CMOS залежно від конфігурації комп’ютера. У мікросхемі CMOS реалізовано, у тому числі, годинник реального часу **RTS** (Real Time Clock). Він працює і при вимкненому з мережі комп’ютері від спеціальної батареї. Годинник дозволяє стежити за поточним часом, користувач комп’ютера завжди може дізнатися час, число, місяць, рік, скористатися програмами, які обмежать час використання комп’ютера для дітей. Комп’ютер може нагадати його господареві про необхідність розпочати заздалегідь заплановані на певний час дії, включити в певний час електронну техніку, або вимкнути її тощо.

Південний міст для зв’язку з BIOS та Super I/O використовує шину **LPC. LPC (low pin count)** – шина, яка використовується в IBM PC-сумісних персональних комп’ютерах для підключення пристроїв, що не вимагають великої пропускної здатності до ЦПВ. Зазначимо, що для сучасних персональних (але не промислових) комп’ютерів **блок портів Super I/O (супер введення-виведення)** є дещо застарілим, оскільки:

- для клавіатури і миші використовується **порт USB**;

- **Paralel Port**, він же LPT-порт, був призначений для принтерів, але принтери на сьогодні мають інтерфейс USB;
- **Serial Port**, він же **COM-порт**, на сьогодні також не використовується, оскільки був призначений для телефонного модему або миші. У випадку, коли COM-порт потрібен, наприклад, для зв'язку з промисловим контролером, використовують зовнішній перехідник USB-COM;
- **floppy**-диски вийшли з ужитку.

Звичайно, представлений варіант архітектури системної плати не є вичерпним і наведений тут для демонстрації можливостей типового структурного рішення і можливостей інтеграції з периферією.

4.3 Хабова архітектура системної плати на базі архітектури Pentium 4 та AMD 64

З введенням високошвидкісних режимів UltraDMA (ATA/66, ATA/100 і ATA/133) зв'язок двоканального контролера IDE з пам'яттю через шину PCI став сильно навантажувати цю шину. Крім того, з'явилися високошвидкісні інтерфейси, такі як Gigabit Ethernet і USB 2.0. Відповіддю став перехід на хабів архітектуру чипсета. Хаби – це спеціалізовані мікросхеми, що забезпечують передачу даних між своїми зовнішніми інтерфейсами, а саме: інтерфейсами процесорів, модулів пам'яті, шин розширення, периферійні інтерфейси ATA, SATA, USB, Ethernet. Чипсет будується з пари основних хабів (північного і південного), пов'язаних між собою високопродуктивним каналом [28].

Для прикладу розглянемо систему на базі Pentium 4. Компанія Intel випустила в 2002 році набір мікросхем Chipset 850, в який входять:

– концентратор контролерів пам'яті **MCH** (Memory Controller Hub) типу Intel 82850;

– концентратор контролерів пристроїв введення-виведення **ICH2** (I/O Controller Hub) типу Intel 82801BA;

– контролер мікрокоду **FWH** (FirmWare Hub) типу Intel 82802AB.

Контролер-концентратор пам'яті **MCH** називають північним хабом, контролер-концентратор **ICH2** – південним хабом. Чипсети для Pentium 4 вдосконалювалися, змінилася система шин та тип пам'яті, але типова архітектура залишилася колишньою. Основною особливістю цієї архітектури є використання контролерів-концентраторів, з'єднаних за принципом точка – точка.

На боці **MCH** використано шину **FSB**, що забезпечує обмін зі швидкістю 3,2 Гбайт/с, з частотою передачі даних 400 МГц. При тактовій частоті каналу 100 МГц за рахунок чотирьох каналів забезпечується частота обміну з оперативною пам'яттю, еквівалентна 400 МГц. Це у 3 рази вище, ніж для системних плат, що працюють на частоті 133 МГц. До контролера MCH підключається також універсальний роз'єм AGP4X (PCI Express), що використовується для зв'язку з графічним адаптером при швидкості передачі більше 1 Гбайт/с.

Контролер **ICH2** служить для підключення різних зовнішніх пристроїв: 2 контролери по 2 порти USB; порт Ethernet; шина PCI на 5 слотів; 6 каналів аудіоданих; BIOS на флеш-пам'яті ємністю 4 Мбіт; порти PS/2 для підключення клавіатури й миші, на сьогодні не використовуються; 1 COM-порт; 2 слоти ULTRA ATA/66/100 (Serial ATA, SATA) для обміну із жорстким диском на швидкості 66 або 100 Мбайт/с.

Очевидно, що з основних структурних змін у хабовій архітектурі від Intel маємо перенесення шини PCI на південний хаб, відмову від інтерфейсу PCI на рівні зв'язку між мікросхемами чипсету, використання більш потужної двох або чотирьохканальної шини FSB для обміну з оперативною пам'яттю. Також варто зазначити, що мікросхеми хабів часто на схемах іменуються також і мостами, що вносить певну плутанину. Насправді треба дивитись саме на структурну схему материнської плати, а не на мікросхему.

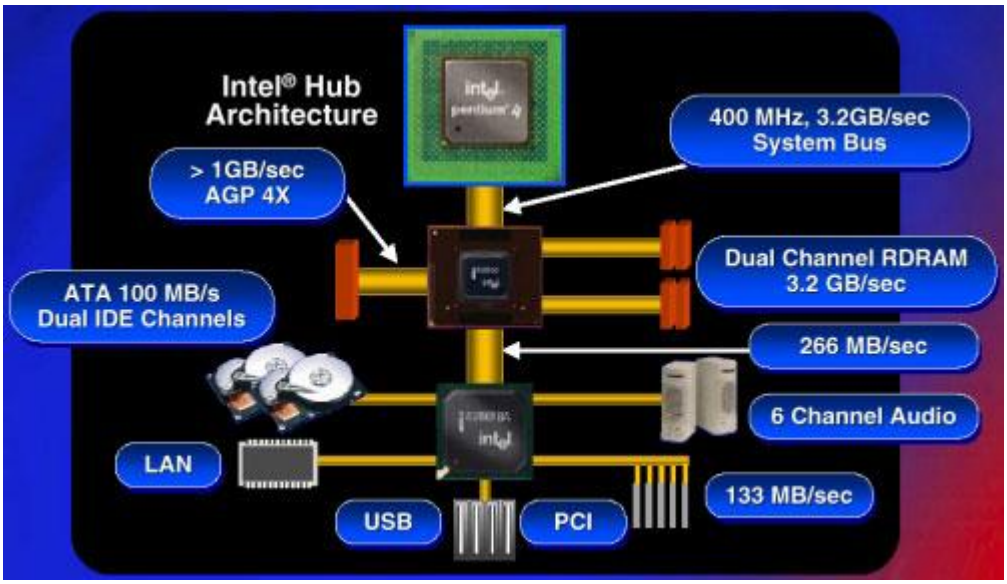

Рисунок 4.3 – Архітектура систем Chipset 850

Особливість архітектури 64-розрядних процесорів AMD 64, яка використовується в процесорах **Athlon 64** і **Sempron 64** дозволяє працювати як з 64-бітними додатками, так і з 32-бітними без втрати продуктивності. На додачу, Athlon 64 використовують технологію **Cool'n'Quiet**, яка дозволяє знижувати тактову частоту і напругу на процесорі в залежності від розв'язуваних задач, наприклад, під час набору тексту. Приклад хабової архітектури AMD наведено на рисунку 4.4 [28].

На відміну від архітектури Intel, хабова архітектура від AMD використовує пряме підключення до оперативної пам'яті по окремій шині, що підвищує продуктивність системи без застосування більш вартісних технологій для побудови оперативної пам'яті. Разом з тим тут застосовано технологію **HyperTransport™**. Це високопродуктивним двоспрямований інтерфейс типу "точка-точка", розроблений за участі AMD і призначений для зв'язку пристроїв південного і північного хабів на швидкості до 12,8 Гб/с.

Північний хаб має назву **AMD HyperTransport™ Tunnel**.
Південний хаб – **AMD HyperTransport™ I/O Hub.**

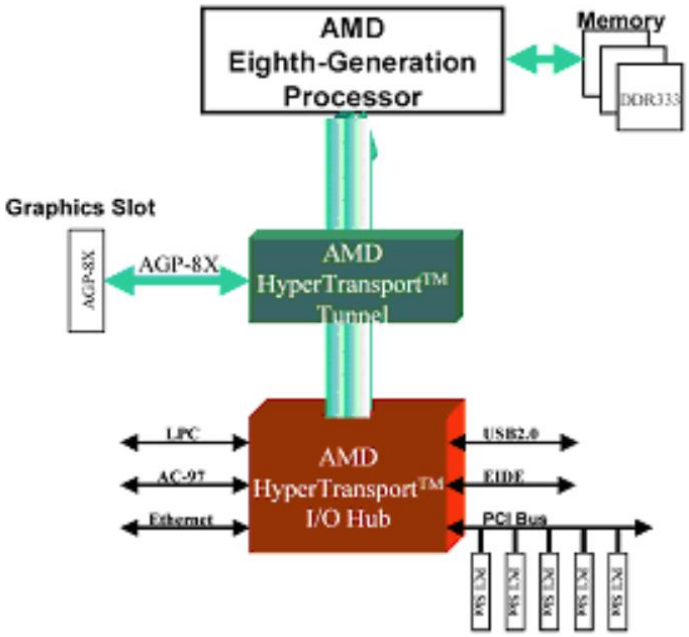

Рисунок 4.4 – Архітектура AMD 64

У розгляді шинно-мостової та хабової архітектури не було розглянуто такі важливі компоненти як системний таймер, контролер DMA та контролер переривань (мікросхема APIC). У обох випадках всі ці пристрої належать південному мосту чи хабу.

4.4 Висновки

Цифрові ЕОМ з програмно змінюваною логікою мають дещо різні архітектури, які, тим не менш, включають такі обов'язкові компоненти як постійна пам'ять, оперативна пам'ять, центральний процесор, тактовий генератор, системна шина, інтерфейси введення та виведення даних, контролери периферії, частина з яких для ЕОМ різних призначень складає обов'язкові набори. До цих наборів входять системні таймери, контролери переривань, можуть входити контролери прямого доступу до пам'яті, інтерфейси та порти послідовного чи паралельного введення-виведення із різних середовищ.

Розвиток архітектури комп'ютера йшов від шинної до шинно-мостової, а згодом і хабової архітектури. Це не значить, що з появою кожного нового рішення попереднє відмирало. Так, на сьогодні, саме шинна архітектура використовується у багатьох промислових контролерах, однокристальних мікро-ЕОМ, інтелектуальних приладах промислового та побутового призначення. Те ж саме можна сказати й про шинно-мостове рішення: воно порівняно дешеве і цілком придатне для контролерів інтернету речей, приладів різного призначення, малопотужних комп'ютерів – там, де продуктивність цієї архітектури достатня для вирішення поставлених задач.

На сьогодні основною архітектурою системної плати персонального комп'ютера є хабова. При цьому хабова архітектура ввібрала у себе як ідеї шинного рішення (якщо поглянути на шини PCI та USB південного хабу) так і мостове. В обох архітектурах є дві функціонально наближені мікросхеми чипсету, які узгоджують різноманітні інтерфейси до одного високопродуктивного: для мостів це PCI, для хабів – швидкісний спеціалізований міжхабовий інтерфейс.

4.5 Питання для самоперевірки

1. Які пристрої включає базова конфігурація персонального комп'ютера?
2. Для чого потрібна системна плата?
3. Для чого потрібні процесор та тактовий генератор?
4. З яких підшин складається системна шина?
5. Чим відрізняється фон-Нейманівська архітектура комп'ютера від Гарвардської?
6. Для чого потрібна постійна та оперативна пам'ять комп'ютера?
7. Що таке BIOS Hardware Interrupts?
8. Для чого використовуються ROM BIOS Services?
9. Як працює контролер переривань?
10. Як працює контролер DMA?
11. Які компоненти включає в себе системна плата шинно-мостової архітектури?
12. Чим відрізняється хабова архітектура системної плати від шинно-мостової?
13. Які компоненти шинно-мостової та хабової архітектури можна вважати на сьогодні застарілими?
14. Чим відрізняється хабова архітектура з технологією HyperTransport™ від тієї, що використовує Chipset 850?
15. Що таке контролер APIC?
16. Що таке SATA?
17. Навіщо потрібен порт AGP?
18. Що об'єднує між собою північний хаб?
19. Що об'єднує між собою південний хаб?
20. Що таке Raid-масив?
21. Чим відрізняються апаратні переривання від програмних переривань?
22. Що таке вектор переривання?
23. Що таке RAM?
24. Яку роль відіграють шини PCI різних модифікацій у сучасній архітектурі системної плати?

5 АРХІТЕКТУРА ПРОЦЕСОРА

5.1 Класифікація процесорів за технологією обробки команд

Існує дуже велика кількість процесорів, різних за поколіннями, архітектурою, функціоналом і сферами застосування. Їх можна класифікувати, у тому числі, за принципом використання технологій обробки інструкцій. При цьому варто пам'ятати, що технології обробки тим чи іншим чином впливають як на архітектуру, так і на призначення того чи іншого процесора. За вказаним критерієм можна виділити наступні різновиди процесорів [22]:

- **CISC (Complex Instruction Set Computer)** – «комп'ютер з повним набором команд», тип процесорної архітектури, в першу чергу, з нефіксованою тривалістю часу виконання команд, а також з кодуванням арифметичних дій в одній команді і невеликою кількістю регістрів, спеціалізованих на строго визначених функціях. У CISC процесорах одна команда може бути замінена аналогічною, або групою команд, що виконають таку ж дію. Звідси випливає висока продуктивність завдяки тому, що кілька команд можуть бути замінені однією аналогічною, але вища ціна порівняно з RISC-процесорами через більшу складність архітектури;

- **RISC (Reduced Instruction Set Computer)** – «комп'ютер зі скороченим набором команд», архітектурний процесор, в якому швидкодію збільшено за рахунок спрощення інструкцій: їх декодування стає більш простим, а час виконання – меншим. Перші RISC-процесори не мали навіть інструкцій множення і ділення, а також не підтримували роботу з числами з плаваючою крапкою. Ця архітектура має фіксовану тривалість команд, а також меншу кількість схожих інструкцій. Це зменшує ціну процесора та енергоспоживання, що критично для мобільного сегменту. Деякі виробники комп'ютерної техніки вважають, що RISC-процесори продуктивніше CISC. Це дискусійне питання – з одного боку, RISC- процесори у 5..10 разів швидші за CISC, але замість однієї машинної команди CISC необхідно, через обмеженість набору інструкцій, виконати 4…6 RISC-команд. У будь-якому випадку тут все визначається характером вирішуваної задачі, ефективністю використання доступної архітектури процесора та особливостями реалізації алгоритму програми;

- **MISC (Minimal Instruction Set Computer)** – «комп'ютер з мінімальним набором команд», подальший розвиток RISC-технології, вирізняється ще більш простою архітектурою, спрямованою на зменшення вартості та енергоспоживання процесора. Використовується в IoT-сегменті і недорогих системах, наприклад, у роутерах. **IoT (Internet of Things)** – Інтернет речей, концепція обчислювальної мережі фізичних предметів («речей»), оснащених вбудованими технологіями одне з одним або із зовнішнім середовищем. Ця концепція може застосовуватись не лише з MISC-процесорами, а й з іншими різновидами в залежності від призначення та функціоналу пристрою;

- **VLIW (Very Long Instruction Word)** – «дуже довга машинна команда», архітектура процесорів з декількома обчислювальними пристроями.

Характеризується тим, процесор містить і виконує паралельно чи псевдопаралельно кілька операцій.

Всі означені вище різновиди процесорів використовують певні структурні (архітектурні) рішення та їх комбінації. Розглянемо ці рішення докладніше.

5.2 Базова архітектура процесора

Для опису принципу роботи процесора наведемо зразок його базової архітектури, показаний на рисунку 5.1. Така архітектура ще називається скалярною.

Скалярний процесор – це найпростіший клас мікропроцесорів. Скалярний процесор дозволяє отримати один елемент даних за одну інструкцію (SISD, Single Instruction Single Data), типовими елементами даних можуть бути цілі або числа з плаваючою комою [18, 22].

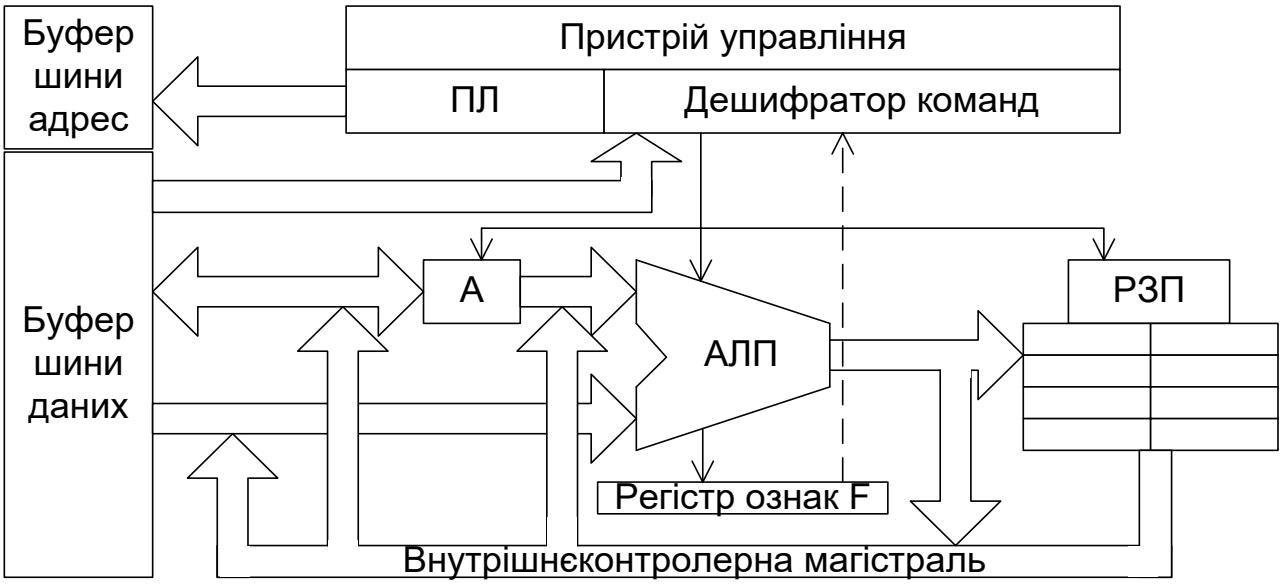

Рисунок 5.1 – Базова архітектура процесора

На рисунку 5.1 внутрішнєконтролерна магістраль показана у вигляді окремих зв'язків (широкі стрілки), хоча частіше на схемах її показують у вигляді загальної шини. Вузькі стрілки показують напрям керуючих впливів.

Розглянемо роботу процесора на прикладі виконання команди додавання. Найважливішим вузлом процесора є **пристрій управління**, що об'єднує в собі **програмний лічильник** ПЛ і **дешифратор команд**. Під впливом тактового генератора виставляється адреса з програмного лічильника ПЛ до буфера шини адрес, за якою з пам'яті у буфер шини даних з пам'яті програм зчитується перше слово машинної команди.

Генератор тактових імпульсів (генератор тактової частоти, тактовий генератор) призначений для синхронізації різних процесів в цифрових пристроях – процесорах ЕОМ, електронному годиннику, таймерах та інших.

Програмний лічильник ПЛ під впливом одиничного імпульсу від тактового генератора збільшується на одиницю, а перше слово машинної команди

потрапляє до **дешифратора команд** де воно порівнюється з еталонними мікрокодами, у результаті чого відбувається налаштування обладнання: налаштовується арифметико-логічний пристрій АЛП на виконання операції додавання, розпізнається тип адресації для роботи з наступним машинним словом. Тим часом програмний лічильник виставляє наступну адресу слова команди, на яку припадає перший аргумент. Цей аргумент через буфер шини даних надходить до дешифратора команд і відбувається визначення, в якому з **оперативних регістрів загального призначення** РЗП, чи, можливо в зверхоперативному регістрі А, який називається **акумулятором**, знаходиться перший аргумент і згодом буде розміщено результат операції. Тим часом знову збільшений на одиницю лічильник ПЛ виставляє наступну адресу команди у буфер шини адреси і система отримує другий аргумент команди. Далі дешифратор команди визначає, де знаходиться другий аргумент: регістрі РЗП, акумуляторі А, чи у зовнішній пам'яті даних. В останньому випадку, використовуючи додаткові такти машинного циклу, у **буфер шини адрес** виставляється **адреса комірки пам'яті даних**, за якою у **буфер шини даних** потрапляє другий аргумент. З буфера шини даних, чи з регістра загального призначення другий аргумент потрапляє до АЛП і виконується, власне, операція (у нашому випадку додавання). Результат потрапляє до місця зберігання першого аргументу, а лічильник ПЛ збільшується на 1 для читання наступної команди. Під час виконання операції можуть бути виставлені **ознаки у регістрі ознак F**. У нашому випадку це може бути у тому числі й ознака переповнення, коли результат виявився більшим, ніж може вмістити машинне слово процесора, а також ознака нуля. Останній випадок настане, коли за умови операції додавання сталося переповнення (і є відповідна ознака), але таке, що всі комірки машинного слова дорівнюють нулю.

Машинне слово традиційно машинного рівня – це довжина даного у бітах така, що дане може бути отримане по шині даних, або записане у регістр, або над ним може бути виконана мікрооперація за один такт тактового генератора. Розмірність машинного слова у бітах становить 8^n, де n=1..m в залежності від розрядності шини даних чи регістра процесора у байтах. Не варто плутати машинне слово і тип даних «слово», тобто WORD. Останнє – беззнаковий цілочисельний тип даних високорівневих мов програмування довжиною у два байти.

Зазначимо, що розглянута архітектура лише показує принципи роботи простого процесора, але не розкриває особливості якої-небудь конкретної моделі. Так, акумулятор А у серії x86 віднесено до складу РЗП, а для архітектури сімейства PIC замість акумулятора використовується робочий регістр W, до якого немає прямого звертання як до аргументу машинної команди. Процесори сімейств x80, x86, x51 використовують особливий регістр SP, вказівник стека, а ті ж серії PIC10, 12, 16 вказівника стека не мають, як не мають і РЗП. Точніше, РЗП для них – вся внутрішня (розміщена на тому ж кристалі) пам'ять даних (процесори є однокристальними мікро-ЕОМ Гарвардської архітектури). Сімейства x51 (також однокристальні мікро-ЕОМ Гарвардської архітектури) навпаки, включають ще чотири змінних набори по 8 РЗП. Процесори можуть відрізнятись і розмірністю машинного слова і наявністю додаткових спеціальних регістрів, наприклад, для

непрямої адресації, але в основному їх базова архітектура відповідає показаній на рисунку 5.1.

5.3 Блокова архітектура процесора

Основна особливість блокової архітектури – здатність виконувати команди, на виконання яких витрачається часу більше, ніж на їх зчитування по системній шині і дешифрацію. Рисунок 5.2 показує особливості роботи процесора з блоковою архітектурою [29] або процесорами з блоковими АЛП [22].

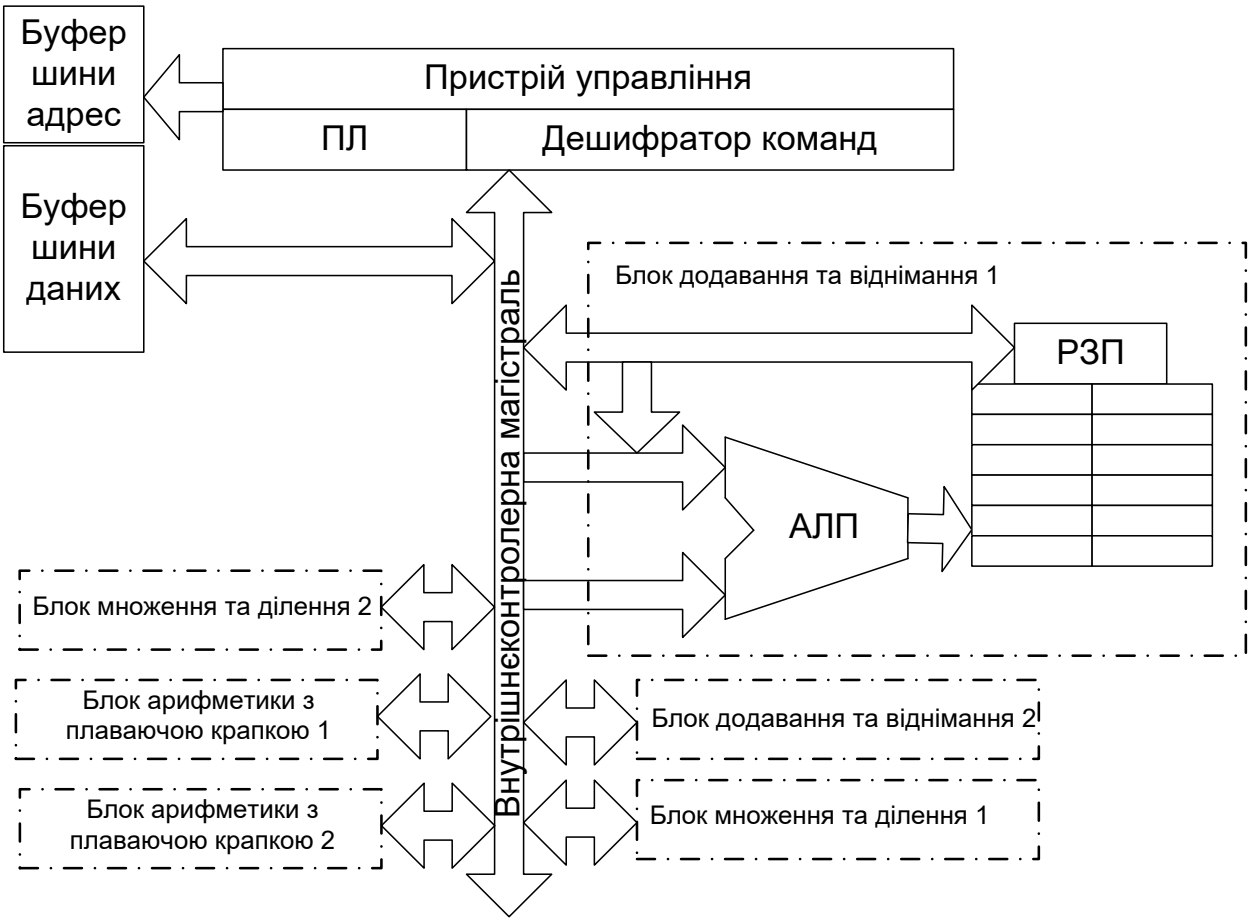

Рисунок 5.2 – Блокова архітектура процесора

Як показано на рисунку 5.2, є кілька окремих наборів АЛП та РЗП (акумулятор та інші додаткові регістри зараз для пояснень не суттєві). Всі блоки спеціалізовані на виконання притаманних тільки їм функцій, хоча може бути і дублювання, як на рисунку. Як і в базовій архітектурі, пристрій управління зчитує та дешифрує команду, але налагоджує для використання і розміщує аргументи не будь-де, а у вільний блок, спеціалізований на виконанні саме таких операцій. Потім пристрій управління, не чекаючи завершення операції, зчитує наступну команду, і так далі. Доведено, що у середньому на процесорі блокової архітектури паралельно можуть обробляти команди до 30% обчислювальних блоків. Архітектура виправдана для використання на великих ЕОМ масового обслуговування для складних обчислень, таких як моделювання процесів у ядерній фізиці.

Звичайно, середня продуктивність процесору буде знижуватись при використанні у алгоритмі команд умовного переходу і послідовність машинних команд повинна бути такою, щоб паралельно вести кілька відносно незалежних розрахунків, тобто результат виконання n-ї команди не повинен використовуватись для n+1-ї. Останнє накладає додаткові вимоги на складання алгоритмів програм і проєктування засобів управління процесами в операційних системах. Тому блочні архітектури з великою кількістю блоків не є поширеними.

Тим не менш, архітектура процесора на два блоки знайшла своє місце і для персональних комп'ютерів. Були розроблені співпроцесори для арифметики з плаваючою крапкою, які хоча і виконувались на окремому кристалі, але знаходились під управлінням головного. Це співпроцесор i8087 для процесора i8086, i80287 для i80286, i80387 для i80386 та їх аналоги. Таке рішення дістало назву рознесеної архітектури [22]. Більшість процесорів покоління x486 наявності співпроцесора не потребували, оскільки мали у своєму складі два блоки: блок для цілочисельних розрахунків та блок арифметики з плаваючою крапкою. Архітектура Pentium також підтримує блоковий підхід, хоча ні x486, ні Pentium не є процесорами блокової архітектури.

5.4 Матрична архітектура процесора

Принцип матричної організації обчислень – паралельні обчислення на кількох наборах АЛП та РЗП за однією і тією ж програмою. Рисунок 5.3 демонструє роботу матричного процесора. Це явно виражена Гарвардська модель процесора, причому пам'ять даних організована так, що кожний блок АЛП-РЗП працює із власним сегментом даних [29]. Рішення розраховане на роботу з параметричними циклами, тобто циклами, де заздалегідь відома кількість кроків. Що стосується непараметричних обчислень або умовних переходів по гілкам алгоритму в залежності від результатів обчислень – така архітектура не дуже вдала, оскільки є декілька наборів даних, результати обробки яких будуть відрізнятися. Тому і оцінка умови переходу для кожного результату обчислень може бути різною. Щоб вийти з такої ситуації, зазвичай вважають один з блоків (перший) ведучим і умовний перехід у алгоритмі здійснюють за результатами обчислень ведучого блоку.

Матрична архітектура виправдана у системах, де потрібно обробляти велику кількість даних, але обчислення порівняно прості. Як приклад, це може бути задача обробки графіки, моделювання та прогнозування погоди, коли надходять великі обсяги даних метеоспостережень. Таке рішення існувало на великих ЕОМ 60-80 рр. (наприклад, обчислювальна система SOLOMON), але на сьогодні не використовується.

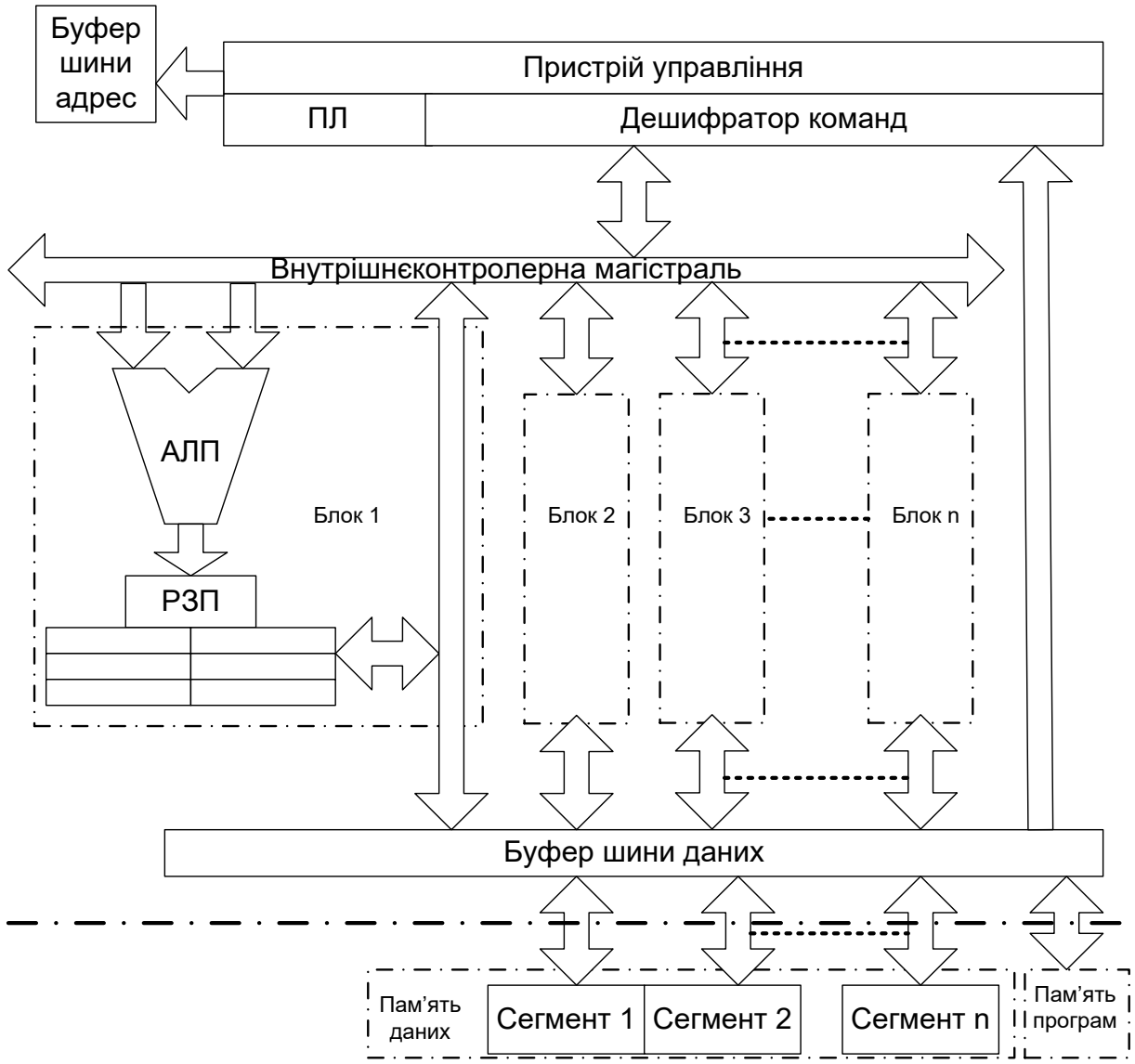

Рисунок 5.3 – Матрична архітектура процесора

Окрім поняття матричної архітектури процесора існує ще **матрична комп'ютерна система** [22]. Це дещо інше – замість матриці АЛП використовується матриця процесорів під керуванням єдиного контролеру масиву за аналогією з пристроєм управління окремого процесору. Така система є багатопроцесорною і має назву **SIMD-процесора (Single Instruction-stream Multiple Data-stream, один потік команд з кількома потоками даних)**. Аналогічно матричному, процесори SIMD-процесора виконують одну послідовність команд стосовно різних наборів даних. Першим у світі таким процесором був ILLIAC IV (університет Іллінойс) [29]. Більш докладно багатопроцесорні системи розглянуті нижче.

5.5 Векторна архітектура процесора

Векторні процесори з точки зору програміста за принципом дії подібні матричним. Але тут є лише один АЛП, який працює з групою РЗП. Іншими словами, на один АЛП подаються дані з різних РЗП, але з регістрів одного й того

самого розташування, причому над кожним з цих даних по черзі АЛП виконує одну й ту саму операцію [29]. Такі регістри називаються векторами. Наприклад, якщо в окремому блоці РЗП є регістри A, B, C, то і для групи РЗП існують вектори A, B, C. Відповідно, АЛП по черзі виконає дію над кожним елементом вектору A, B чи C. Група РЗП у даному випадку називається векторним РЗП. А загальна архітектура, яка зовні нагадує SISD, але у результаті працює як матрична, називається векторною. Вектор даних РЗП також називають чергою даних.

5.6 Магістральні процесори

Розглянемо архітектуру **магістрального або конвеєрного процесора**. Цей тип процесорів будь-яку команду виконують поетапно за рахунок використання буферу конвеєризації команд.

Для пояснення принципу роботи магістрального процесора покажемо не всю структурну схему, а лише принцип роботи п'ятиступінчастого **конвеєра команд** деякого умовного процесора.

Перша ступінь (див. рисунок 5.4а) (блок С1) викликає команду з пам'яті і розміщує її у буфер, де вона зберігається доти, доки не буде потрібно. Друга ступінь (блок С2) декодує цю команду, визначаючи її тип і тип операндів. Третя ступінь (блок С3) визначає місцезнаходження операндів і викликає їх з регістрів або пам'яті. Четверта ступінь (блок С4) виконує команду, зазвичай проводячи операнди через тракт даних. І нарешті, блок С5 записує результат назад у потрібний регістр [29].

На рисунку 5.4б показано роботу конвеєра у часі. Під час циклу 1 блок С1 обробляє команду 1, викликаючи її з пам'яті. Під час циклу 2 блок С2 декодує команду 1, тоді як блок С1 викликає з пам'яті команду 2. Під час циклу 3 блок С3 викликає операнди для команди 1 блок С2 декодує команду 2, а блок С1 викликає команду 3 і так далі [29].

Таким чином магістральний процесор реалізує принцип конвеєрної обробки команд. На рисунку 5.4 на конвеєрі одночасно може знаходитись до 5 команд, значить, за рахунок псевдопаралелізму тривалість машинного циклу по виконанню машинної команди скорочується у 5 разів порівняно з базовою (SISD) архітектурою без підвищення швидкості циклів.

Функціонально магістральна архітектура є ще й векторною. Дійсно, АЛП повинна обробляти вектор даних РЗП, оскільки на конвеєрі знаходиться одночасно кілька команд із власними даними.

Виникає питання: як буде працювати конвеєр, якщо у програмі зустрінеться команда умовного переходу? Процесор не може виконувати наступну команду, поки не виконається команда умовного переходу і його продуктивність падає до продуктивності базової архітектури [29]. Насправді, навіть в не дуже вдалих алгоритмах таких команд не багато, близько 10%, тому конвеєризація все одно ефективна.

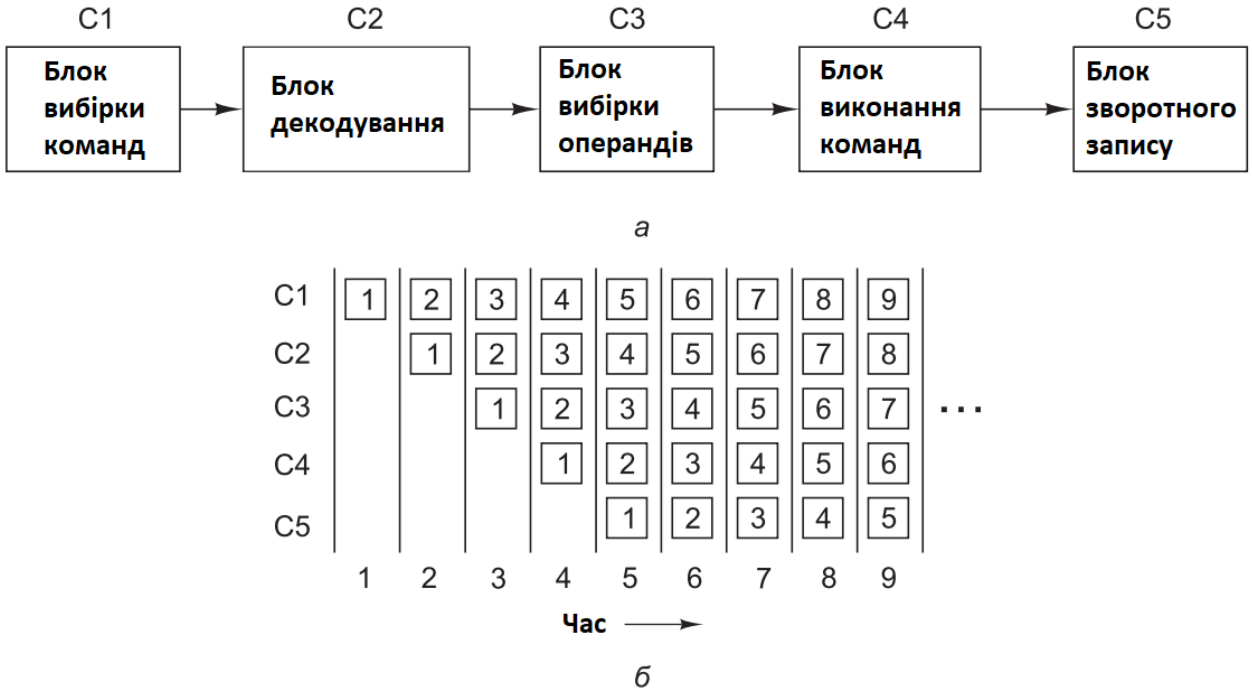

Рисунок 5.4 – П’ятиступінчастий конвеєр (а); стан кожного ступеня залежно кількості пройдених циклів (б). Показано 9 циклів

Всі сучасні процесори, які використовують конвеєри, для прискорення обробки команд умовного переходу мають ще й механізм спекулятивності.

Спекулятивний магістральний процесор – це процесор, який не чекає завершення команди умовного переходу, а обирає одну з гілок алгоритму випадково, завжди одну й ту саму, або виходячи з попереднього досвіду. Процесор оперує двійковою системою числення, переходи на машинному рівні мають лише 2 варіанти, а тому 50% випадків вибору у команді умовного переходу мають бути правильними і продуктивність не впаде. Якщо ж за результатами завершення команди умовного переходу було зроблено неправильний вибір, вміст конвеєра очищається, зміст програмного лічильника ПЛ і РЗП відновлюється до моменту виконання команди умовного переходу, а потім здійснюється умовний перехід на потрібну гілку алгоритму [29].

Досвід спекулятивного процесора – це використання лічильників з обмеженою пам’яттю, на попередні 2…3 результати виконання умовного переходу. Вони підраховують кількість переходів за умовою «Так», 1 або «Ні», 0. Вибір напрямку у такому випадку йде на користь лічильника з більшим значенням. Такий принцип врахування досвіду ефективний для керування умовними переходами у циклічних алгоритмах, оскільки тут циклічно перевіряється одна й та ж умова і лише при виході з тіла циклу буде обрано іншу гілку алгоритму.

Для процесорів персональних комп’ютерів черга команд з’явилася ще у процесора i8086 [30], де був для цього передбачений буфер на 6 8-розрядних регістрів, де можна було помістити 2..3 команди, але справжнім процесором з магістральною архітектурою можна вважати i80386. Тут 6-байтна черга команд

замінена 2-ступінчастим конвеєром: 16-байтна черга команд і окрема від неї черга інструкцій. Пристрій вибірки в міру звільнення черги команд вибирає з пам'яті нові команди і поміщає їх в чергу, а дешифратор команд із звільненням у черзі інструкцій вибирає з черги команд нові команди, перетворює їх в 112-бітний внутрішній формат і поміщає в чергу інструкцій. Пристрій управління процесора вибирає вже дешифровані команди з черги інструкцій і виконує їх. Зазвичай по завершенню виконання однієї команди негайно починається виконання наступної, яка вже вибрана з пам'яті і дешифрована. Однак тут не реалізовано механізм спекулятивності, як і в процесорі i8086 [31].

Процесор i80486 є подальшим розвитком i80386, і в ньому, на відміну від попереднього, з'явився вбудований блок математичного співпроцесора для операцій з плаваючою крапкою (окрім моделі 486SX). Тобто i80486 включає в себе як векторну, так і блокову архітектуру. Інші нововведення в архітектуру 80486 [32]:

- введені так звані пакетні цикли, передають одне машинне слово за один такт шини (до цього слово передавалось за два такти).
- в архітектурі процесора застосоване швидкісне RISC-ядро, яке дозволяє виконувати найчастіші команди за один такт;
- передбачено внутрішні множники тактової частоти (на 2, 2.5 або 3);
- подовжено конвеєр команд до 32 байт.

Таким чином, виходячи з визначення скалярного процесора, i80486 таким не є. Це процесор блоково-конвеєрної архітектури.

5.7 Суперскалярні процесори

Суперскалярна організація процесора означає, що на кожному етапі обробляються відразу кілька потоків інструкцій паралельно – від вибірки з кешу інструкцій до повного завершення чи відставки у разі невдалої спекулятивної операції. Подібне рішення, коли використовується 2 паралельних конвеєри вперше з'явилося в Intel Pentium. На рисунку 5.5 показана саме така архітектура. Вона використовує загальний блок вибірки команд, який розподіляє команди між конвеєрами [29].

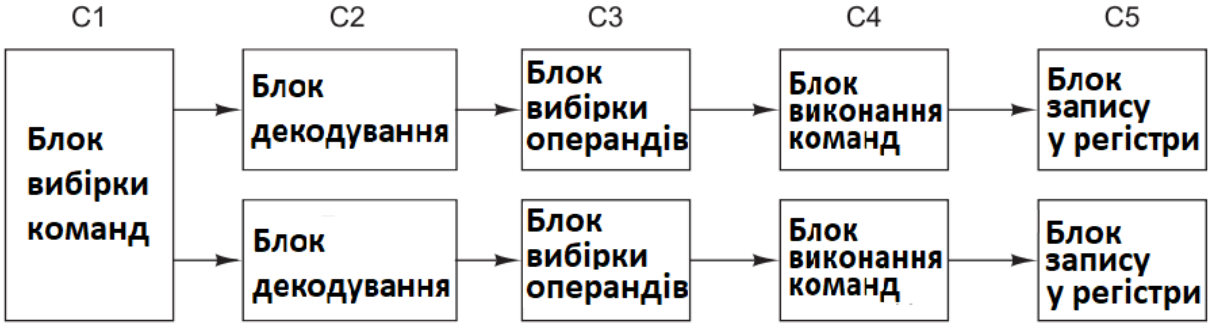

Рисунок 5.5 – Здвоєний п'ятиступінчастий конвеєр процесора Intel Pentium

Блоки вибірки операндів при використанні архітектури, показаної на рисунку 5.5, отримують дані або з векторів РЗП, або з **кеш-пам'яті**. Головний конвеєр (u-конвеєр) тут виконує будь-які команди, допоміжний, v-конвеєр – лише прості цілочисельні команди та одну з плаваючою крапкою [29].

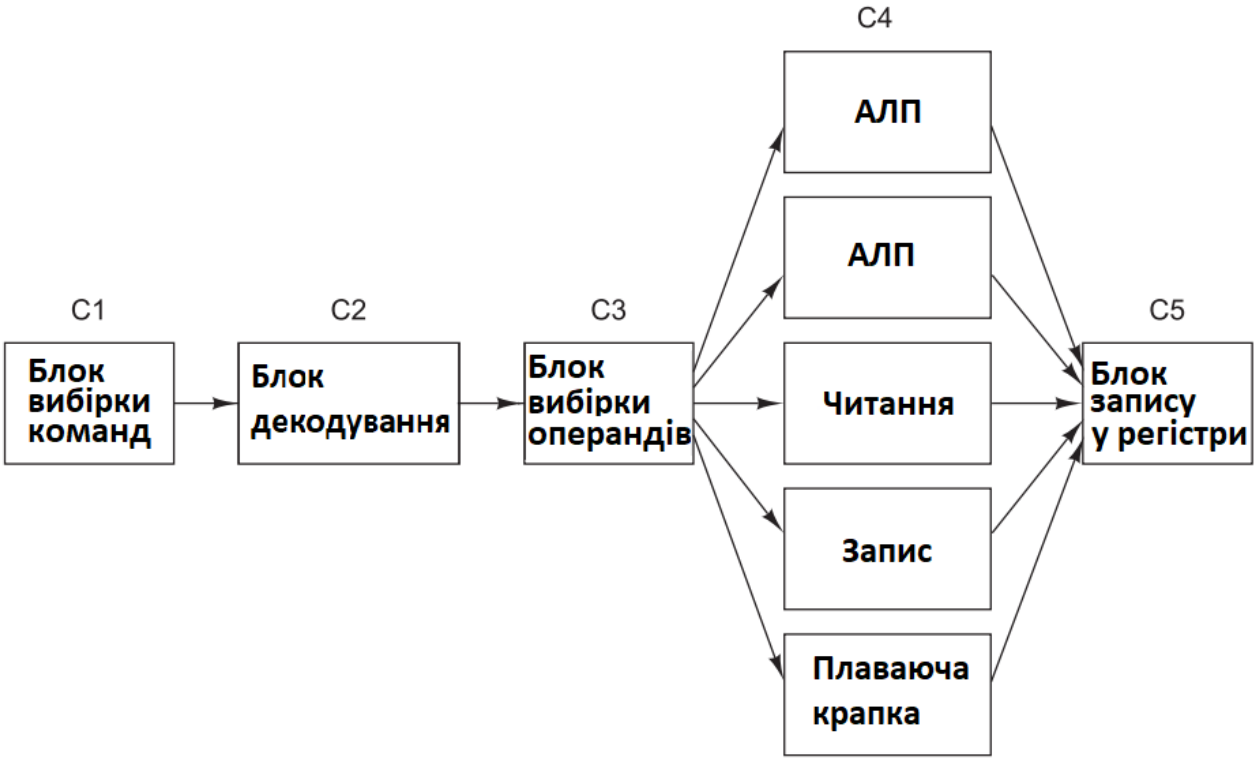

Рисунок 5.6 – Суперскалярний процесор з п'ятьма функціональними блоками

Кеш-пам'ять – надоперативна пам'ять, вбудована у системну плату чи сам процесор, яка забезпечує більшу продуктивність обчислювальної системи порівняно із оперативною пам'яттю. Докладніше розглянута у розділах, присвячених саме пам'яті.

Перехід до більшої кількості конвеєрів вважається технічно недоцільним, тому вже процесор Intel Core використовує іншу архітектуру, яка вважається на сьогодні суперскалярною [29]. Натомість архітектуру, показану на рисунку 5.5 у деяких джерелах називають суперконвеєрною [22], а у деяких – суперскалярною [29].

Як видно з рисунка 5.6, тут використано один конвеєр, але у комбінації з ним застосована блокова архітектура. Таке рішення притаманне процесорам Pentium II та Pentium III. А їх конкурент AMD Athlon використовує три конвеєри. Більш пізня модель AMD Duron використовує і суперскалярність, і суперконвеєризацію [22].

Також варто зазначити, що з точки зору технології обробки команд, суперскалярні процесори можуть використовувати і CISC, і RISC-технології.

5.8 Багатоядерні процесори

Багатоядерний процесор – це центральний процесор, що містить два або більше обчислювальних ядер в одному корпусі. З цієї точки зору більшість сучасних персональних комп'ютерів і ноутбуків є багатоядерними системами. Для персональних комп'ютерів випускаються 2, 3, 4, 6 та 12-ядерні процесори. Існує кілька альтернативних термінів для багатоядерних процесорів. Їх називають також однокристальними мультипроцесорами (існують і мультипроцесори, складені з окремих процесорів, але принцип побудови той же самий). Є кілька процесорних ядер або процесорів, об'єднаних загальною шиною чи кільцем (кільце це теж різновид шини). Ці ядра мають доступ да загальної пам'яті і, часто але не завжди, до власної локальної [29]. На рисунку 5.7 в якості прикладу наведена загальна архітектура процесора Core i7, де кеш-пам'ять і є згаданою загальною та локальною пам'яттю.

Деякі мультипроцесори називають мультискалярними. Це несиметричні багатопроцесорні системи, де окрім загальних процесорів є виділений планувальник – особливий процесор, що розподіляє задачі [22]. Це не є універсальним рішенням, оскільки планувальником процесор може призначити асиметрична операційна система із числа звичайних ядер. Натомість, симетрична система підтримує конкуренцію між ядрами.

Мультикомп'ютери – обчислювальні системи без загальної пам'яті. Використовуються, коли очікувана кількість ядер чи процесорів перевищує 256. Один з таких мультикомп'ютерів – IBM Blue Gene/P, має близько 250 000 процесорів [29].

Окрім структурних, існують ще й віртуальні процесорні ядра. Технологія Hyper-Threading в процесорах виробництва Intel змушує комп'ютер працювати з двоядерним процесором як з чотириядерним [33].

Деякі додатки не потребують багато ресурсів. Це можуть бути прості офісні програми або програми для роботи з Інтернетом. У такому випадку Hyper-Threading розділяє кожне фізичне процесорне ядро на дві віртуальні частини. Відповідно, і конвеєри, і вектори регістрів ядра також будуть розділені. Застосування будуть виконуватись швидше. І навпаки, якщо додаток ресурсомісткий, пов'язаний з обробкою графіки, програмуванням, іграми, то розділення на два віртуальних ядра не виконується, воно працюватиме як єдиний пристрій.

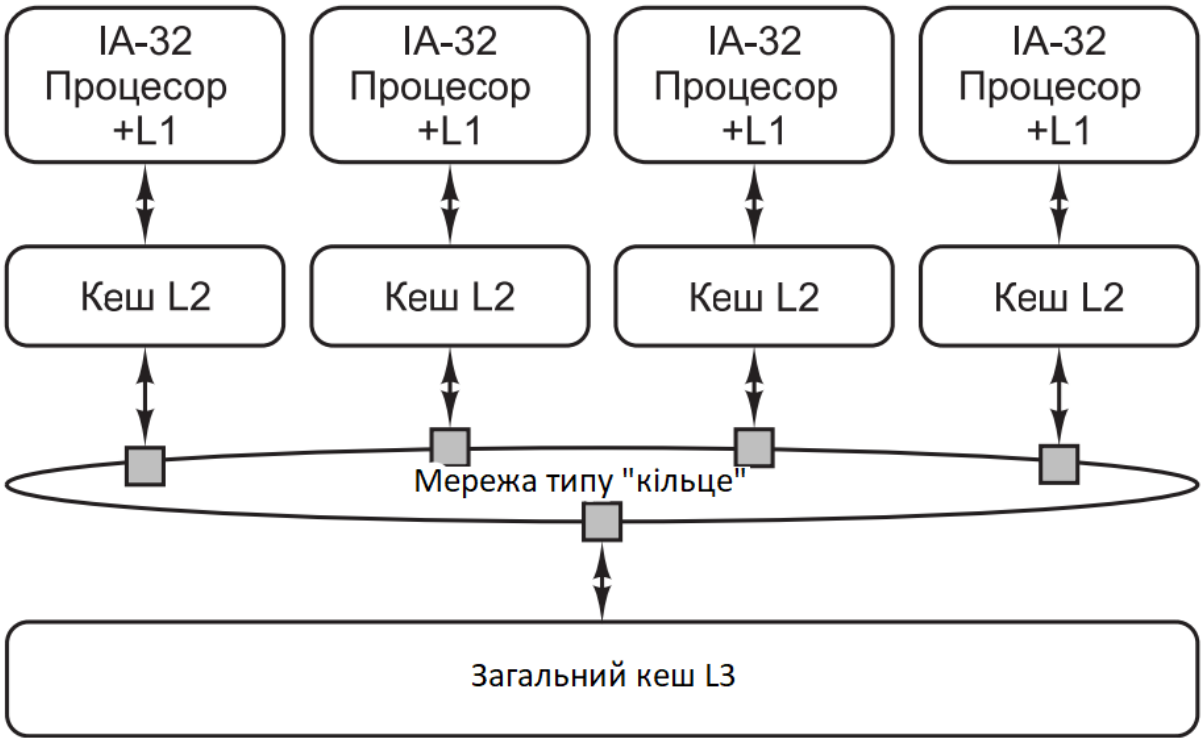

Рисунок 5.7 – Однокристальний мультипроцесор (багатоядерний процесор) Core i7

Починаючи з класу Core i7, процесори обов'язково оснащені Hyper-Threading. У них 4 фізичних ядра і 8 віртуальних. Тобто, на одному процесорі одночасно можуть працювати до 8 обчислювальних потоків. Менш дорогі, але також потужні процесори Intel класу Core i5 складаються з чотирьох ядер, але Hyper Threading там не працює, тому Core i5 працюють з 4 потоками обчислень. Процесори середньої на сьогоднішній день продуктивності Core i3 переважно мають два ядра з Hyper-Threading. Тому Core i3 має чотири обчислювальних потоки. Двоядерні, менш потужні процесори – це процесори лінії Core 2 (Dual-core), Pentium D і Celeron. Нижче наведена порівняльна таблиця деяких багатоядерних систем.

Варто зазначити, що для ПК є і більш дорогі та продуктивні процесори, наприклад, серія Core i9. Насправді, на ринку зараз є багато різноманітних рішень. Наприклад, застаріла на сьогодні технологія Core 2 реалізована не лише у двоядерних, але і чотириядерних процесорах. Тому вибір процесора – досить складна задача, пов'язана з вивченням і порівнянням характеристик різних моделей і поколінь процесорів.

Таблиця 4.1 – Варіанти реалізації фізичних і віртуальних ядер для процесорів різних моделей

Процесор	Кількість ядер	Кількість потоків	Застосування
Atom	1-2	1-4	Застаріла технологія. Малопотужні комп'ютери і нетбуки. Орієнтовані на мінімальне енергоспоживання. Продуктивність у них мінімальна. На сьогодні не випускається
Celeron	2	2	Застаріла технологія. Найдешевші процесори для настільних ПК і ноутбуків. Продуктивність достатня для офісних завдань. Вже не випускається.
Pentium D	2	2	Застаріла технологія. Аналог Celeron. Pentium оснащуються більшим за об'ємом кешем і мають підвищену продуктивність в порівнянні з Celeron. Не випускається
Core i3	2	4	Два продуктивних ядра, кожне з яких поділено на два віртуальних «процесора» (Hyper-Threading). Продуктивність достатня для офісних завдань та домашнього користування (ігри, мультимедіа)
Core i5	4	4	Повноцінні чотириядерні, досить дорогі процесори. Продуктивність достатня для більшості сучасних завдань для ПК.
Core i7	4-6	8-12	Потужні, але дорогі процесори Intel. Як правило, рідко виявляються швидше Core i5, і лише в деяких програмах.
Core i9	8–24	16–32+	Флагманські процесори, створені для максимальних навантажень. Часто потребують кращого охолодження. Створені для важких задач: 3D-рендеринг, відеомонтаж 4K/8K, професійні програми, робота з великими даними, ігри.

5.9 Висновки

Як показано у викладеному підрозділі, розвиток архітектури процесорів йде поетапно, від базового (скалярного, SISD) процесора до сучасних мультискалярних та мультикомп'ютерних рішень. Розвиток архітектури йшов із підвищенням складності й різноманітності задач, які ставляться обчислювальним системам. З іншого боку, відставання розвитку технологій виробництва великих інтегральних схем стримував і стримує зараз цей розвиток. Так, у чистому

вигляді блокова й матрична архітектури процесора є за своєю сутністю досить затратними, орієнтованими на певні специфічні класи задач. Зокрема, для блокової архітектури збільшення кількості операційних блоків процесора у 10 разів призводить лише до трикратного зросту продуктивності, а зріст продуктивності матричного процесора зумовлений наявністю готових до обробки сегментів даних – якщо даних для заповнення всіх сегментів не достатньо, то й зріст підсумковий продуктивності менший. Тим не менш, ідеї, реалізовані у блоковій та матричний архітектурах, дали свій синергетичний ефект. Тому вони були повторені у пізніших рішеннях.

Окремо варто звернути увагу на базові технології обробки інструкцій процесора, які прямо впливають на внутрішні схемотехнічні рішення. Справа у тому, що існують різні сегменти застосування програмованих цифрових обчислювальних систем. Так, перехід свого часу від CISC до RISC-технологій дозволив вийти на ринок такому виробнику як Microchip з серією дешевих і продуктивних процесорів PIC. Це відразу здешевило конкурентні CISC-процесори x51 від Intel, Motorola, Sygnal. І PIC, і x51, і більш пізні x196, які є, перш за все, скалярними процесорами. Вони займали і займають на сьогодні свою нішу на ринку: промислові контролери, інтелектуальні прилади різного призначення, IoT. Переваги RISC над CISC у цьому сегменті з точки зору споживача на сьогодні не очевидні, але виробники, зі свого боку, запропонували ще й MISC-технологію для ще більш вузької і специфічної ніші. Це свідчить про жорстку конкуренцію у сфері простих спеціалізованих обчислювальних засобів, а також про очікувані перспективи розвитку цього напрямку.

Щодо VLIW, то ця технологія обробки притаманна більш пізнім та більш складним варіантам архітектури. Так, спочатку з'явились ідеї векторів даних, конвеєризації операцій, спекулятивності. Перші спроби VLIW-обробки з'являються в i80386 та i80486. І хоча, наприклад, i80486 використовує RISC-ядро, надалі, для більш сучасних і потужних процесорів, більше значення для підвищення продуктивності починають відігравати не технології обробки окремих інструкцій, а структурні й функціональні рішення, які можуть включати як RISC, так і CISC-підходи.

Суперскалярні, а згодом і багатоядерні архітектури фактично об'єднали у собі властивості скалярних, блокових, матричних і магістральних архітектур. За умови належного керування цим апаратним ресурсом з боку операційної системи, людство отримало досить потужний і універсальний засіб для здійснення як локальних так і розподілених у мережі обчислень. При цьому сучасний процесор універсального комп'ютера є результатом певної тривалої еволюції як технологій на рівні схемотехніки, так і на рівні методів обробки даних і команд.

5.10 Питання для самоперевірки

1. Які різновиди архітектур за принципом використання технологій обробки інструкцій Ви можете назвати?

2. У чому полягають відмінності RISC-архітектури процесора від CISC-архітектури?

3. Що таке IoT?

4. Що собою являє архітектура VLIW процесора?

5. Що таке скалярний процесор?

6. Які функціональні вузли є у пристрою управління процесором?

7. Що таке машинне слово?

8. Що таке F-регістр?

9. Які особливості в структурній схемі, роботі і застосуванні блокового процесора?

10. Які особливості в структурній схемі, роботі і застосуванні матричного процесора?

11. Чим векторний процесор відрізняється від матричного?

12. Який принцип роботи спекулятивного магістрального процесора?

13. Які є варіанти реалізації за структурою і принципом роботи для суперскалярних процесорів?

14. Що собою являють багатоядерні процесори і які варіанти реалізації їх існують?

15. Що собою являє технологія Hyper Threading?

16. Які процесори використовують VLIW-технологію?

17. Які процесори використовують RISC-технологію?

18. Які процесори використовують скалярну технологію?

21. Які спільні риси притаманні матричній та векторній архітектурі процесорів?

22. У чому відмінність матричної від блокової архітектури?

23. Що спільного є у магістральній та суперскалярній архітектурі процесорів?

24. У чому полягають відмінності магістральної від суперскалярної архітектури процесорів?

25. Що спільного є у блоковій та суперскалярній архітектурі процесорів, таких як Pentium II і III?

26. У чому полягають відмінності суперконвеєрної від суперскалярної архітектури процесорів?

27. Для чого використовується MISC-архітектура?

28. Які багатоядерні системи Ви знаєте?

29. Про які суперкомп’ютери Ви знаєте?

30. Які CISC-процесори використовуються на сьогоднішній день?

6 КОМАНДИ ТРАДИЦІЙНО МАШИННОГО РІВНЯ ЕОМ

6.1 Загальні принципи побудови команд традиційно машинного рівня

У загальному випадку команда машинного рівня, що надходить у потоці команд з системної шини, може мати наступні складові [29]:

1. Код команди – двійкова комбінація, необхідна дешифратору команд для налагодження арифметико-логічного пристрою перед виконанням операції.
2. Тип адресації – визначає, яким чином процесор повинен обробляти аргументи команди, щоб виконати очікувану операцію над даними.
3. Перший аргумент – містить інформацію про дане, яке повинне бути інтерпретоване відповідно до використаного типу адресації.
4. Другий аргумент – аналогічний першому.
5. Інформація про місце зберігання результату.

Більшість процесорів не використовують як окрему складову місце зберігання результату для основних команд, вважаючи, що місце зберігання результату – перший аргумент. Тим не менш, є ряд команд, що виконують читання регістрів зі стеку, де місце зберігання результату використовується, такі як:

```
POP RCX; зчитати до 64-розрядного регістру даних C дані зі стеку
POP EAX; зчитати до 32-розрядного акумулятору A дані зі стеку
POP BL; зчитати до 16-розрядного регістру BL пари BX дані зі стеку
POP E; зчитати до 8-розрядного регістру E дані зі стеку
```

та їм подібні [34]. Далі будуть використовуватись команди мови асемблера, оскільки вони більш придатні для сприйняття, ніж безпосередньо команди машинного рівня. Приклад наведено для 64, 32, 16 та 8-розрядного процесора відповідно. Тут ні першого, ні другого аргументів немає, але вказане місце зберігання результату машинної операції.

Варто звернути увагу і на формати команд сімейства процесорів PIC [35], де явно вказується, де буде зберігатись результат виконання операції – на місці зберігання першого чи другого аргументів, наприклад:

```
ADDWF f,d; додати регістр f до робочого регістру W. Якщо d=0, то
          ; результат зберегти у W, якщо d=1 – то у f
```

Таким чином, багато команд у сімействі процесорів PIC, хоча і не містять посилань на перший аргумент, проте надають можливість вибору місця збереження результату.

Демонстрація команди ADDWF показує ще одну особливість машинної мови для PIC – тут позбулися від першого аргументу в ряді команд, використовуючи для обчислень робочий регістр W – аналог акумулятору, причому згадок про W регістр у самих машинних командах немає.

Втім, і інші машинні мови, такі як мови для процесорів сімейств x64, x86, x51 [36] не використовують перший аргумент (якщо вважати, що перший аргумент –

це акумулятор) у так званих одномісних операціях, тобто таких, де дія над даним виконується безпосередньо у місці його зберігання. Це може бути зсув, інверсія, декремент або подібна операція.

Команди, що взагалі не мають операндів, використовуються в усіх відомих сімействах процесорів. Це такі команди як, наприклад [34]:

```
NOP; немає операції
RET; повернення з процедури
MOVSW; Записати в комірку за адресою ES:DI
        ; слово з комірки з адресою DS:SI, де ES та DS – адреси
        ; сегментів даних в архітектурі x286, DI та SI – адреси комірок
        ; пам'яті у цих сегментах
```

Скоротити такі елементи команди машинного рівня як тип адресації та код операції теоретично можливо, але на сьогодні недоцільно. Тому такі варіанти команд надалі розглядатися не будуть. У загальному випадку, під час проєктування процесора і його системи команд повинні застосовуватись наступні принципи [29]:

1. Принцип достатності – АЛП процесора повинен розпізнавати та виконувати весь необхідний набір команд для вирішення задач, покладених на процесор, при цьому множина команд має бути достатньою, щоб продуктивність обчислень не падала нижче очікуваної.

2. Принцип мінімалізму – надлишкові машинні команди ускладнюють апаратну реалізацію АЛП і сприяють зменшенню співвідношення продуктивність/вартість. Тому не варто реалізовувати апаратну підтримку команд, які на даній архітектурі будуть мало використані.

3. Принцип універсальності – мають враховуватись всі вимоги та різноманітність задач, що будуть вирішуватись на комп'ютері чи пристрої, під який проєктується процесор.

4. Принцип кратності – всі машинні команди повинні мати довжину, кратну розрядності регістрів процесора, а в ідеалі ще й таку, що співпадає з розрядністю шини даних материнської плати, щоб зчитуватись за один такт системної шини. Ця довжина виражена деяким числом 2^n.

5. Принцип розширення системи команд – не завжди достатня кількість команд кратна 2^n. Але оскільки витримується принцип кратності – не всі можливі двійкові комбінації машинного слова коду операції будуть використані, що призводить до нераціонального використання апаратних ресурсів. З іншого боку, не всі операції потребують повного формату команди. Наприклад, команда NOP або RET не має аргументів і не повністю використовують системну шину. У такому випадку місце першого аргументу можна використати для розширення коду операції NOP або RET, щоб використати розширений код, наприклад, для умовного переходу чи двійково-десяткової корекції числа в акумуляторі. Тоді, з точки зору програміста, команда умовного переходу чи корекції існує, а з точки зору системної шини і процесора – це та ж сама команда NOP чи RET, тому загальна кількість команд і число можуть 2^n бути зменшені, що у ряді випадків скорочує кількість тактів системної шини для під час зчитування команди.

6.2 Типи адресації в командах традиційно машинного рівня

Перед тим, як перейти до розгляду команд традиційно машинного рівня доцільно навести типи адресації, що існують для цих команд [29]:

1. Безпосередня адресація. В якості аргументу команди використовується саме дане, розміщене, таким чином, безпосередньо у коді програми у вигляді константи. Звичайно, у більшості архітектур така адресація зустрічається у комбінації з іншим видом, оскільки сама константа не має сенсу без сумісного використання із змінною.

2. Пряма адресація. В якості аргументу команди використовується адреса комірки пам'яті, де розміщене дане, яке у цьому випадку є змінною.

3. Непряма адресація. В якості аргументу команди використовується адреса регістру загального призначення процесору. У свою чергу, в регістрі зберігається адреса комірки пам'яті, де розміщене дане. Таке ускладнення необхідне для обробки даних, зібраних у масиви. Забезпечена можливість циклічного повтору одних і тих же обчислень над кожною змінною масиву. На кожному кроці циклу адреса звернення, що зберігається у регістрі процесора, збільшується або зменшується.

4. Регістрова адресація. Іноді вважається підвидом прямої. Але тут в якості аргументу використовується не адреса комірки пам'яті, а номер регістру загального призначення. Розміщення даних у регістрах загального призначення дозволяє процесору більш швидко виконати команду, оскільки не буде витрат часу на роботу із системною шиною материнської плати. Для процесорів сімейства PIC такий вид адресації не виділяється, оскільки там йде робота із внутрішньою оперативною пам'яттю як з РЗП, а окремих РЗП немає [35].

5. Стекова (безадресна, неявна) адресація. Вважається підвидом непрямої адресації. Операції зі стековою адресацією називають також безадресними операціями. Команда зі стековою адресацією не потребує аргументу. Адреса аргументу міститься у спеціальному регістрі – вказівнику стека. Вказівник стека завжди вказує на так звану вершину стека – останнє розміщене дане у стеці. Стек – ділянка оперативної пам'яті з одностороннім доступом, або, як ще її називають, пам'ять магазинного типу. Порядок одностороннього доступу до пам'яті називається LIFO (last input – first output, останній увійшов – перший вийшов). Якщо дане розміщується у стеці, покажчик стека збільшується на 1, якщо виймається – зменшується на 1.

Окрім вказаних типів адресації існують і інші види, такі як адресація зі зміщенням, відносна, індексна, сторінкова, блокова [22]. Це не основні типи адресацій, які використовуються обмежено для деяких видів процесорів. Тут використовуються додаткові дії, пов'язані з апаратним обчисленням прямих чи непрямих адрес для прискорення роботи з рядками, блоками, сторінками даних. Окремо у цьому посібнику вони не розглядаються.

Зазначимо, що багато машинних команд, ті що використовують два аргументи, містять два типи адресації – по одному на аргумент.

6.3 Групи команд традиційно машинного рівня

Для процесорів сучасних персональних комп'ютерів існує класифікація машинних команд [35, 37] (і команд мови асемблера), пов'язана з розвитком технологій:

- команди загального призначення;
- операції роботи з рядками;
- системні команди;
- команди співпроцесора (x87 FPU);
- команди управління станом співпроцесора і SIMD;
- команди технології MMX, SSE, SSE2…SSE4;
- команди розширення 3DNow!, DNow! Ext;
- команди архітектури x86-64.

Розглянемо кожну групу окремо.

6.3.1 Команди загального призначення

Ця група команд є універсальною і може використовуватись процесорами різних архітектур від скалярної до суперскалярної. На прикладі команд загального призначення розглянемо ще один вид класифікації команд за розмірністю і функціями [29]:

1. Команди передачі даних. Перш за все це команди читання і записування. При цьому обмін даними між оперативною пам'яттю, акумулятором та регістрами загального призначення. Можна сюди віднести і команди обміну даними між стеком та РЗП. В сучасних архітектурах процесорів для персональних комп'ютерів використовують також команди перетворення байту у слово, слова у подвійне слово, обмін місцями тетрад у регістрі чи байт у слові і подібні операції.

2. Двомісні операції. Це команди, що потребують наявності двох операндів. Команди передачі даних можуть бути як з одним, так і з двома операндами, але вони виділені у попередню групу. Тут маємо на увазі арифметичні та логічні операції: додавання, віднімання, множення, ділення, логічні І, АБО, ВИКЛЮЧАЮЧЕ АБО.

Іноді двомісною може бути інверсія, якщо вона не реалізована апаратно. В останньому випадку для інверсії, наприклад, 8-розрядного X необхідно виконати наступну дію: X <– (FFh – X).

3. Одномісні операції. Це команди, що для виконання потребують наявності одного операнду: інверсія, інкремент, декремент, двійково-десяткова корекція, логічні та циклічні зсуви вправо та вліво.

Інкремент – збільшення змінної на 1.

Декремент – зменшення змінної на 1.

Двійково-десяткова корекція – це операція, що використовується для відображення даних засобами людино-машинних інтерфейсів, такими як монітори, принтери, цифрові індикатори.

Циклічний зсув вмісту регістру передбачає, що на місце нового розряду, що з'явився, копіюється розряд, який випав за межі регістру у результаті зсуву. Така операція може використовуватись, наприклад, при порозрядному маскуванні

змінної за методом ТА при завадостійкому кодуванні чи обробці сигналів від датчиків з виходом ТАК-НІ.

Арифметичний зсув уліво еквівалентний операції множення на 2 з тією різницею, що у більшості архітектур він виконується швидше. Подвійний зсув, відповідно, замінює множення на 4. Арифметичний зсув вправо, відповідно, еквівалентний діленню на 2 без остачі. Якщо дільник чи множник не кратний 2, то зсуви для множення і ділення застосовувати не можна. В такому випадку, якщо процесор не підтримує команди множення чи ділення (наприклад, йдеться про програмування дешевої однокристальної мікро-ЕОМ для датчика, простого приладу, речі) застосовують алгоритми з додаванням або відніманням.

4. Безадресні операції. Це операції перш за все роботи зі стеком, зокрема операції повернення. Вони не містять аргументів, але при цьому з вершини стека дане, що являє собою адресу продовження машинного коду після виклику процедури, переноситься до лічильника команд. Команди читання даних між стеком та РЗП також можна віднести не до п. 1, а до безадресних операцій, хоча тут є аргумент і скомбіновано стекову адресацію з регістровою. Процесори серій PIC10, 12, 16, 18 містять апаратний стек глибиною від 2 до 8, побудований на 13-бітних регістрах. Ці процесори команди обміну зі стеком не використовують. Також до безадресних можна віднести команду NOP – немає операції. Цю команду використовують, коли у програмному коді необхідно зарезервувати місце під майбутню вставку або для того, щоб витратити процесорний час, тобто організувати затримку, оскільки на читання NOP з пам’яті програм та дешифрацію час все одно витрачається. Це не кращий метод організації затримок, оскільки у програмі буде припущення щодо продуктивності процесору, але іноді він виправданий.

5. Операції умовного переходу. Ці операції змінюють вміст лічильника команд або залишають таким, як був в залежності від деяких ознак результату виконання попередньої машинної операції. Звичайно, після виконання операції, незмінений лічильник команд збільшується на одиницю, щоб перейти до наступної операції. Ознаки результату виконання попередньої операції зберігаються у спеціальному регістрі – регістрі ознак або регістрі прапорів. В більшості архітектур його позначають літерою F. Операції умовного переходу бувають: перехід по нулю чи ненульовому результату, перехід по переповненню або непереповненню регістра акумулятора, перехід за парністю чи непарністю результату обчислень та інші подібні операції. Сюди ж можна віднести і операцію безумовного переходу. У будь-якому випадку, команди умовного переходу в якості аргументу містять адресу переходу, що завантажується до програмного лічильника.

6. Операції виклику процедур. На машинному рівні операції виклику безумовно змінюють зміст лічильника команд. Особливості їх роботи будуть розглянуті окремо.

В основному, команди загального призначення розглянуто. До огляду не ввійшли команди, специфічні для архітектур x86 та x64, що з’являлися протягом еволюції процесорів та разом з появою нових структурних компонентів: двійково-десяткові арифметичні команди; команди роботи з бітами (хоча такі з’явились

спочатку для однокристальних мікро-ЕОМ) та окремими байтами у 32 та 64-розрядних процесорах; та команди роботи з рядками символів та послідовностями байтів.

6.3.2 Операції роботи з рядками та послідовностями байтів

Це операції, специфічні для суперскалярних архітектур. Їх ще називають ланцюговими операціями.

Рядок – це послідовність байтів з відомим алгоритмом визначення її довжини. У мовах програмування прописують процедуру визначення довжини рядків, а також обмеження щодо байтів, які можуть містити в рядках, а також щодо форми інтерпретації рядків. Наприклад, у мові С рядком вважають послідовність байтів, що обмежена байтом зі значенням 0.

Рядкові операції працюють з одним елементом рядка: байтом, словом, подвійним словом (мова йде про типи даних, а не про машинні слова). Для того, щоб команда маніпулювала послідовністю наведених елементів, використовуються команди (наведено мовою асембера) [34]:

REP – повторювати, поки ECX не дорівнюватиме 0;

REPE/REPZ – повторювати, поки елементи двох рядків співпадають (пошук неспівпадаючої пари байт) /повторювати, поки ознака ZF нульова. Рядкова команда виконується до тих пір, поки вміст ECX не стане 0;

REPNE/REPNZ – антипод парі REPE/REPZ, тобто повторювати, поки є неспівпадіння двох рядків (пошук співпадаючої пари байт)/повторювати, поки ознака ZF не нульова.

Рядкові команди вважають, що рядок-джерело знаходиться за адресою DS: ESI, а рядок-приймач за адресою ES: EDI.

Усі рядкові команди можна поділяються на шість груп [34]:

1. Команди пересилки рядка байтів, слів, подвійних слів.
2. Команди порівняння рядка байтів, слів, подвійних слів.
3. Команди пошуку рядка байтів, слів, подвійних слів.
4. Команди читання зі рядка байтів, слів, подвійних слів.
5. Команди запису до рядка байтів, слів, подвійних слів.
6. Команди читання/запису з/до порту введення/виведення рядка байтів, слів, подвійних слів.

Приклад копіювання масиву байтів з використанням інструкції REP MOVSB:

```
; Копіювання N байтів із [ESI] до [EDI] за допомогою REP MOVSB
; Передумови:
;   ECX = кількість байтів для копіювання
;   ESI = джерело
;   EDI = призначення
;   DF = 0 (встановимо CLD, щоб індекси зростали)
; Результат:
;   Скопійовано ECX байтів; ESI та EDI зміщені на ECX
```

```
    copy_bytes:
  cld             ; DF=0: інкремент ESI/EDI після кожного байта
  rep movsb       ; повторювати MOVSB, поки ECX != 0
  ret
```

Приклад порівняння двох рядків/масивів байтів до першої розбіжності з використанням інструкції REPE CMPSB:

```
; Порівняння двох масивів байтів, поки вони збігаються.
;   ESI = адреса першого масиву
;   EDI = адреса другого масиву
;   ECX = довжина для порівняння
; Вихід:
;   ZF=1 якщо всі ECX байтів збіглися; ZF=0 якщо знайдено першу розбіжність.
;  ESI/EDI вказують на позицію ПІСЛЯ перевіреного байту (або на перший збій +
;  1).
;   ECX показує, скільки байтів лишилось неперевіреними.

compare_bytes_equal_prefix:
  cld
  repe cmpsb      ; повторювати, поки ECX != 0 і ZF=1 (тобто байти однакові)
  ; Після виходу:
  ; - Якщо ECX стало 0 => усі перевірені байти рівні (ZF=1).
  ; - Якщо перервалося через ZF=0 => знайдено першу відмінність.
  ;   Поточні (ESI-1) і (EDI-1) були порівняні байти.
  ret
```

6.3.3 Системні команди

Це операції апаратної підтримки операційної системи та віртуальної пам’яті, а також керування апаратними засобами та системою переривань [34]. Тут є такі команди як зупинка процесора, блокування системної шини, вхід/вихід з привілейованого режиму, узгодження привілеїв та прав доступу процесів до апаратних ресурсів, перевірка сегментів процесів у віртуальній пам’яті, робота з таблицями дескрипторів (тобто з таблицями вказівників), зі словами-характеристиками поточного стану обчислювальної системи, звертання до лічильника продуктивності системи та міток реального часу і подібні операції, вивчення і застосування яких потребують окремої навчальної дисципліни.

Приклад з використанням системних команд:

```
      hlt   ; Зупинка процесора

; Вхід у привілейований режим як у стан CPU (ring0).
; Перехід у привілейований режим як у стан CPU (ring0)
; через PE-біт у CR0 (звичайні команди x86):
      mov eax, cr0
```

```
        or eax, 1
        mov cr0, eax

; Завантаження GDT/IDT (системні таблиці дескрипторів)
        lgdt [gdt_desc]
        lidt [idt_desc]

; Виклик переривання та повернення
        int 0x80      ; (для Linux історично, у user space → системний виклик
                      ; у kernel — software interrupt)
        iret          ; повернення з обробника переривання
                      ; (тільки у відповідному контексті ISR)

        ; Атомарність через LOCK
        lock add dword [mem], 1

        ; Таймер TSC
        rdtsc         ; результат у EDX:EAX
```

6.3.4 Команди співпроцесора (x87 FPU)

Не важливо, чи є співпроцесор окремим процесором, чи входить до складу суперскалярного процесора як представник технічного рішення з блоковою архітектурою, його робота орієнтована перш за все на операції з плаваючою крапкою. При цьому, є багато операцій, за призначенням аналогічних операціям загального призначення. Сюди відносять [34]:

1. Команди передачі даних. Окрім звичайної передачі дійсних чисел тут є команди роботи з двійково-десятковими числами, читання/запису цілих чисел (тобто перетворення чисел між дійсними і цілими форматами), умовне присвоєння за рівністю, нерівністю, умовою «більше», «менше» та подібні команди.

2. Базові арифметичні команди. В основному це двохмісні, але є і одномісні операції. Перш за все це різні варіанти (у тому числі за участю цілого числа) додавання, віднімання, множення, ділення, отримання остачі. Щодо одномісних операцій, то сюди слід віднести отримання модуля числа, зміна знаку, округлення до цілого, множення на ступінь двійки (тобто арифметичний зсув вліво), отримання квадратного кореня, розкладання дійсного числа на мантису та ступінь числа 10.

3. Команди порівняння. Це команди порівняння аргументу команди зі змінною, що зберігається у стеці співпроцесора з виштовхуванням цією змінної зі стека або без. Аргументом може бути вміст регістра співпроцесора або комірка пам’яті. При цьому змінна може бути як дійсною, так і цілою.

4. Команди трансцендентних операцій. Це операції обчислення тригонометричних функцій, а також такі, що призначені для обчислення ступеню числа 2 та логарифмів з довільною основою.

5. Команди завантаження констант. За умови використання програмістом математичних модулів бібліотек операційної системи, до співпроцесора буде завантажено деякі константи, з досить високим ступенем точності представлення:

- число +1.0;
- число +0.0 (з точки зору математичного співпроцесора воно відмінне від -0.0);
- число π;
- число $\boldsymbol{log_2 2}$;
- число $\boldsymbol{log_2 10}$;
- число $\boldsymbol{log_{10} 2}$.

Приклад використання інструкцій співпроцесора:

```
; x87_fpu_demo.asm – демонстрація команд x87 FPU (Linux, 32-bit)
; Показує:
; - базові стекові операції: FLD, FSTP, FADD, FMUL, FDIV
; - спеціальні інструкції: FSQRT, FSIN
; - зміну контрольного слова FPU: FNSTCW, FLDCW (режим округлення)
; Збірка: nasm -f elf32 x87_fpu_demo.asm -o x87_fpu_demo.o
; Лінкування: ld -m elf_i386 x87_fpu_demo.o -o x87_fpu_demo

BITS 32

global _start

section .data
   ; Вхідні значення (double, IEEE-754)
   a      dq 3.25          ; a
   b      dq 2.0           ; b
   c      dq 10.0          ; c
   d      dq 4.0           ; d
   angle  dq 1.0           ; радіани (приблизно 57.2958° — це 1 рад)

   ; Контрольні слова FPU
   cw_orig dw 0            ; оригінальне контрольне слово
   cw_trunc dw 0           ; модифіковане (RC=11b → truncate)

section .bss
   results  resq 6         ; місце для 6 результатів (double)
   ; results[0] = (a*b + c) / d   (звичайне округлення)
   ; results[1] = sqrt(a)
   ; results[2] = sin(angle)
   ; results[3] = (a*b + c) / d   (після зміни округлення на truncate)
   ; results[4] = a + b
   ; results[5] = a / b
```

```
section .text

_start:
   ; 1) Ініціалізація FPU (необов'язково на сучасних системах, але корисно для чистоти)
   finit                    ; ініціалізує x87 (FNINIT без очікування)

   ; 2) Зчитати поточне контрольне слово
   fnstcw [cw_orig]         ; зберегти контрольне слово у пам'ять

   ; 3) Обчислити expr = (a*b + c) / d зі стандартним режимом округлення (round-to-nearest)
   ; Стек x87: працює за принципом LIFO (ST(0) — вершина)
   ; Завантажимо a, b, c, d й послідовно виконаємо операції
   fld qword [a]            ; ST0 = a
   fld qword [b]            ; ST0 = b, ST1 = a
   fmul st0, st1            ; ST0 = b*a
   fld qword [c]            ; ST0 = c, ST1 = (a*b)
   fadd st0, st1            ; ST0 = c + (a*b)
   fld qword [d]            ; ST0 = d, ST1 = c+(a*b)
   fdiv st1, st0            ; ST1 = (c+(a*b)) / d, ST0 = d
   fstp qword [results + 0*8] ; зберегти ST0? Ні — ми хочемо ST1.
                            ; Коректно: витягнемо d (ST0) і потім ST0 стане нашим значенням:
   ; Пояснення: після fdiv st1,st0:
   ;   ST1 = expr, ST0 = d. Тому спершу приберемо d:
   fstp st0                 ; поп (видалити поточний ST0 = d)
   fstp qword [results + 0*8] ; тепер ST0 = expr, і ми зберігаємо expr у results[0]

   ; 4) Корисні спеціальні інструкції: FSQRT, FSIN
   ; sqrt(a)
   fld qword [a]            ; ST0 = a
   fsqrt                    ; ST0 = sqrt(a)
   fstp qword [results + 1*8] ; results[1] = sqrt(a)

   ; sin(angle)
   ; Зауваження: FSIN працює коректно для |angle| <= ~2^63 (але викликає редукцію аргументу).
   ; Для дуже великих значень можуть бути втрати точності / винятки.
   fld qword [angle]        ; ST0 = angle
   fsin                     ; ST0 = sin(angle)
   fstp qword [results + 2*8] ; results[2] = sin(angle)
```

```
; 5) Змінимо режим округлення на truncate (RC=11b) і повторимо вираз
; Формат контрольного слова: RC — біти 10..11
; Зчитаємо поточне CW, модифікуємо, завантажимо назад
mov ax, [cw_orig]
or ax, 0x0C00            ; встановити RC=11b (truncate toward zero)
mov [cw_trunc], ax
fldcw [cw_trunc]         ; завантажити модифіковане контрольне слово

; Повторно обчислити expr = (a*b + c) / d (тепер з truncate)
fld qword [a]            ; ST0 = a
fld qword [b]            ; ST0 = b, ST1 = a
fmul st0, st1            ; ST0 = a*b
fld qword [c]            ; ST0 = c, ST1 = a*b
fadd st0, st1            ; ST0 = c + a*b
fld qword [d]            ; ST0 = d, ST1 = c+a*b
fdiv st1, st0            ; ST1 = (c+a*b) / d, ST0 = d
fstp st0                 ; прибрати d
fstp qword [results + 3*8] ; results[3] = expr (truncate mode)

; 6) Додаткові приклади: додавання та ділення
fld qword [a]            ; ST0 = a
fld qword [b]            ; ST0 = b, ST1 = a
faddp st1, st0           ; ST1 = a + b, pop ST0 → вершина стає сумою
fstp qword [results + 4*8] ; results[4] = a + b

fld qword [a]            ; ST0 = a
fld qword [b]            ; ST0 = b, ST1 = a
fdivp st1, st0           ; ST1 = a / b, pop ST0
fstp qword [results + 5*8] ; results[5] = a / b

; 7) Відновити початкове контрольне слово (бажана практика)
fldcw [cw_orig]

; 8) Акуратне завершення процесу
mov eax, 1               ; sys_exit
xor ebx, ebx             ; код 0
int 0x80
```

6.3.5 Команди керування станом співпроцесора і SIMD

Команди, з тих, що стосуються саме співпроцесора, призначені для управління з боку процесора. Вони орієнтовані на налагодження співпроцесора [34]:

FSTSW dest; Запис слова стану в конфігураційний регістр dest

FLDCW src; Завантаження керуючого слова з dest

На обробку збоїв співпроцесора:

```
FCLEX ; Скидання винятків (помилок). Також скидаються біти ES і B
FSTENV src; Збереження стану співпроцесора в пам’ять за адресою
          ; src (після збою)
FLDENV src; Відновлення стану співпроцесора з пам’яті за адресою src
```

Також є команди очікування звершення обробки команди співпроцесора й інші спеціалізовані команди.

6.3.6 Команди керування стеком співпроцесора

Взагалі ці команди за дією дещо відрізняються від класичних команд роботи зі стеком. Регістри стеку співпроцесора обмежені вісьмома комірками для чисел з плаваючою крапкою. І доступ може бути не лише до вершини, але, для певних потреб, й за назвою регістру: ST0 (це вершина), ST1, ST2 і так до ST7. Команди читання зі стеку також працюють і з його вершиною, і з регістрами за назвою, але, в залежності від відсутності чи присутності індексу «p» у символьному позначенні команди, це може бути лише копіювання змісту вершини або дійсно читання зі стеку [38]. Приклади деяких команд роботи зі стеком:

```
FLD mem ; завантажити у стек 32 або 64-розрядне число mem
        ;з плаваючою крапкою
FLD ST(n) ; завантажити у стек число з іншої комірки стеку
FILD mem ; завантажити у стек 32 або 64-розрядне ціле число mem
FBLD mem ; завантажити у стек двійково-десяткове число mem

FST[p] mem; копіювати [вивантажити] з ST0 32 або
         ; 64-розрядне ціле число mem
FST[p] STn ; копіювати [вивантажити] з ST0 у регістр стеку STn
```

6.3.7 Команди технології MMX

MMX (Multimedia Extensions – мультимедійні розширення) – комерційна назва додаткового набору інструкцій, що виконують характерні для процесів кодування/декодування потокових аудіо/відео даних дії за одну машинну інструкцію. Різновид технології SIMD [39].

У Pentium MMX додано нові команди обробки і чотири нові типи даних. За одну операцію команда MMX обробляє 64-розрядне двійкове слово (так зване квадраслово, або QWord). Нові типи даних утворюються від упаковки у квадраслово байтів (по 8), слів (по 4) або подвійних слів (по 2). Четвертий тип являє собою саме квадраслово. Одна елементарна MMX-операція має справу або з одним квадрасловом, що схоже на звичайну операцію великої розрядності, або з двома подвійними словами, чотирма словами або вісьмома байтами, причому виконання відбувається одночасно і кожен елемент даних обробляється незалежно від інших, що власне, і є проявом SIMD. Подібні групові операції переважають під час обробки зображення і звуку.

Набір MMX-команд складається з команд пересилання даних, упаковки/розпакування, складання/віднімання, множення, зсуву, порівняння і порозрядних логічних операцій [39].

Для обробки даних і зберігання проміжних результатів в Pentium MMX використовуються вісім 64-розрядних регістрів MM0..MM7, які фізично поєднані зі стеком регістрів математичного співпроцесора [39]. Тому одночасна робота режиму математичного співпроцесора і режиму MMX неможлива.

Команди технології SSE, SSE2..SSE4 (англ. Streaming SIMD Extensions, потокове SIMD-розширення процесора) – це набір інструкцій, розроблений Pentium III як подальший розвиток технології MMX, без використання регістрів математичного співпроцесора [40].

Команди технології 3DNow!, 3DNow! Ext. Це аналог технологій SSE, але від фірми AMD, а не Intel.

Команди архітектури x86-64. Архітектура x86-64 підтримує весь наведений вище набір команд архітектури x86 (включаючи розширення MMX, SSE, SSE2, 3DNow!, окрім SSE3), розширюючи деякі інструкції для використання 64-бітових операндів. Додатково вводяться дві нові інструкції: пересилання 4-байтної змінної до 8-байтої зі знаковим розширенням та швидке завантаження адреси системних структур даних [40].

6.4 Висновки

Наведена у розділі класифікація команд не може вважатись вичерпною, але вона достатньо репрезентативна. Система команд на машинному рівні розвивалась одночасно із розвитком варіантів архітектури процесора за певними законами розвитку і у відповідь на нові задачі, що ставились до нових архітектур процесора і технологій, які створювались наново.

На сьогодні можливо виділити певні закономірності у архітектурі команд традиційно машинного рівня:

1. Всі команди потоку команд мають обмежений перелік функціональних складових, який визначається призначенням процесора, застосованою технологією обробки команд, архітектурою процесора і є компромісом між рядом суперечливих вимог до проєктованого процесора.

2. Під час звертання до даних у загальному випадку процесори використовують п’ять основних типів адресації: безпосередню, пряму, непряму, регістрову, стекову а також розширені типи, що базуються на вказаних основних.

3. Будь-які процесори включають у себе перелік команд загального призначення, який має певну внутрішню класифікацію в залежності від функціонального призначення команд, а також кількості та особливостей використання аргументів.

4. Поява співпроцесора та подальший розвиток технологій обчислень на основі співпроцесора призвели до фактичного повторення базової архітектури команд процесора у частині пересилки та обробки даних, але вже з плаваючою крапкою. Додатково з’явилась підтримка трансцендентних операцій та використання точних констант, але це зумовлено специфікою оброблюваних даних.

5. Співпроцесор, як елемент блокової архітектури, став потребувати особливої системи команд для управління ним самим з боку головного процесора, а також його стеком. Суперскалярна та багатоядерна архітектура не мають окремого співпроцесора, а лише використовують технологію на одному з конвеєрів чи ядер. Тим не менш, згадана система команд використовуються і у SIMD-технологіях.

6. Сталий розвиток багатозадачних операційних систем призвів до їх апаратної підтримки на рівні машинних команд з точки зору обслуговування привілейованих режимів, прав доступу до ресурсів, керування енергоспоживанням. Ці, здавалося б, не обов'язкові засоби знайшли своє місце навіть в простих SISD-архітектурах однокристальних мікроконтролерів.

7. Поява потреби у обробці даних потоків мультимедіа призвела до прогресу SIMD-технологій у суперскалярних та багатоядерних архітектурах. Технології MMX, SSE, 3DNow! та їх більш сучасні варіації врешті-решт ввійшли як складові у сучасну архітектуру x86-64 разом із технологіями FPU, керування станами співпроцесора та підтримки операційної системи. Таким чином, архітектура команд x86-64 на сьогодні перш за все є узагальнюючою реплікою системи машинних команд сучасного багатоядерного суперскалярного процесора.

6.5 Питання для самоперевірки

1. Які складові можуть входити до команди традиційно машинного рівня?
2. Які методи скорочення довжини машинної команди використовують у різних сімействах процесорів?
3. Чому у розділі під час розгляду команд машинного рівня використані команди рівня асемблера?
4. Які принципи використовуються під час проєктування процесора і його системи команд?
5. Які є основні типи адресації?
6. Яке призначення команд передачі даних?
7. Які бувають одномісні й двомісні операції? У чому їх відмінності?
8. Чим відрізняються арифметичні й циклічні зсуви?
9. Які операції пов'язані зі стеком та вказівником стеку?
10. Які операції змінюють вміст лічильника команд?
11. Які групи команд за технологіями обслуговують рядки, вектори та потоки даних?
12. У чому є спільне та відмінності технології FPU від базових машинних команд загального призначення?
13. Навіщо використовуються управління станом співпроцесора і SIMD?
14. Для чого використовується операція NOP?
15. Яка технологія SSE не підтримується архітектурою x86-64?

7 ВИКОРИСТАННЯ МОВНИХ КОНСТРУКЦІЙ НА ТРАДИЦІЙНО МАШИННОМУ РІВНІ

7.1 Використання процедур

Процедури, як один з найважливіших інструментів традиційно машинного рівня, що пов'язаний з операціями виклику, потребує окремого огляду.

Процедура, підпрограма, або функція – на традиційно машинному рівні все це є ділянкою машинного коду, орієнтованого на вирішення певної задачі. Цей код називається тілом функції і його розміщено в певному місці пам'яті програм. Основний програмний потік може звертатись до тіла функції багаторазово, використовуючи команди виклику [29].

Слід відрізняти процедури і функції від макросів. **Макроси** не є поняттям традиційно машинного рівня. Це деяка іменована ділянка програмного коду асемблерного або рівня прикладних програм [29]. Ім'я макросу – це його заголовок, тіло – власне, поіменований програмний код. Виклик макросу – це команда транслятору або компілятору під час свого виконання розмістити у вихідному об'єктному коді відтрансльоване чи відкомпільоване тіло макросу, а не команду виклику окремо розміщеного тіла. Скільки в разів у тексті програми був виклик макросу за його іменем, стільки разів в об'єктному коді буде розміщене тіло макросу. Такий підхід призводить до збільшення витрат пам'яті програм, але може бути виправданим, оскільки дещо пришвидшує продуктивність за рахунок відсутності команд виклику і повернення із процедури.

Оскільки виклик процедури явно змінює зміст програмного лічильника, виникає задача зберігання адреси точки повернення із процедури, тобто адреси команди, наступної після команди виклику. Зрозуміло, що ця адреса для різних точок виклику буде відрізнятись.

Взагалі відомо 4 способи зберігання адреси повернення [29]:

1. Зберігання адреси повернення у заголовку процедури, в якості першого слова процедури. Це найбільш невигідний варіант, оскільки вимагає чистої фон-Нейманівської архітектури обчислювальної машини, від чого на рівні операційної системи намагаються відійти, та не припускає рекурсії.

2. Зберігання адреси повернення у фіксованій комірці пам'яті чи групі комірок. Варіант підходить і для фон-Нейманівської, і для Гарвардської архітектури. Але тут виключається рекурсія та вкладені виклики процедур. На сьогодні серії процесорів PIC12…PIC18 використовують так званий апаратний стек адрес повернення – це група від 3 до 10 в залежності від моделі фіксованих комірок пам'яті. Це дозволяє обмежено використовувати вкладені виклики процедур та рекурсію, але останнє – лише на більш потужному PIC18 з 10 регістрами.

3. Зберігання адреси повернення у регістрі чи групі регістрів. Гібрид способу 1 і 2 дозволяє вкладені та рекурсивні виклики, але розміщення адреси повернення повинне визначатись самою процедурою, а фактично, програмістом чи засобами трансляції, досить інтелектуальними для вирішення такої задачі.

4. Зберігання адреси повернення у стеці. Тут кількість вкладень та рекурсивних викликів процедур обмежується розмірами стека, який може бути значним (кілобайти, сотні кілобайт) і визначається на рівні асемблера програмістом чи його інструментарієм з урахуванням обмежень традиційно машинного рівня чи операційної системи.

Дамо визначення терміну **рекурсія**. Це коли один раз викликана процедура викликає саму себе за допомогою команди виклику, розміщеної у власному тілі. Є також **ланцюгова рекурсія**, тобто коли перша процедура викликає другу, а друга першу і так багато разів. У ланцюговій рекурсії може бути задіяно і більше (три, чотири і так далі) процедур [29]. Щоб програма могла для досягнення мети використовувати рекурсивні методи повинно виконуватись дві умови:

1. Рекурсія має бути такою, що сходиться, тобто здача вирішується за кінцеву кількість кроків.

2. Розміри стеку повинні дозволяти зберігати адреси повернень для всіх рекурсивних викликів процедури.

Звичайно, коли використовується спосіб 1 для організації зберігання адреси повернення, перший же рекурсивний виклик зіпсує адресу повернення з процедури, що зберігається у заголовку процедури.

Очевидно, що використання інструменту рекурсії пов’язане з командами виклику на традиційно машинному рівні та командами повернення. Останні процесор має виконати стільки разів, скільки було використано рекурсивний виклик, хоча результат роботи процедури уже досягнуто. З цієї причини використання команд умовного переходу в межах процедури, там де не потрібні додаткові повернення є більш продуктивним інструментом, ніж рекурсія. Як наслідок, рекурсія при написанні програм не дуже популярна.

7.2 Використання співпроцедур

З процедурами пов’язаний один з інструментів організації паралельних обчислень – використання співпроцедур.

Співпроцедури – це група процедур, які виконуються сумісно, фрагментально в міру надходження нових даних чи проходження стадій обробки цих даних. Співпроцедури викликають одна одну, при цьому продовжуючи роботу не з початку тіла, а відразу за операцією виклику [29]. Різницю між звичайною сумісною роботою процедур та співпроцедур показано на рисунку 7.1

Для організації роботи співпроцедур використовується наступний метод. Безпосередньо перед виконанням команди повернення із співпроцедри (крок 1 на рисунку 7.2) у деякий тимчасовий регістр чи комірку пам’яті зчитується вершина стеку, тобто дій.

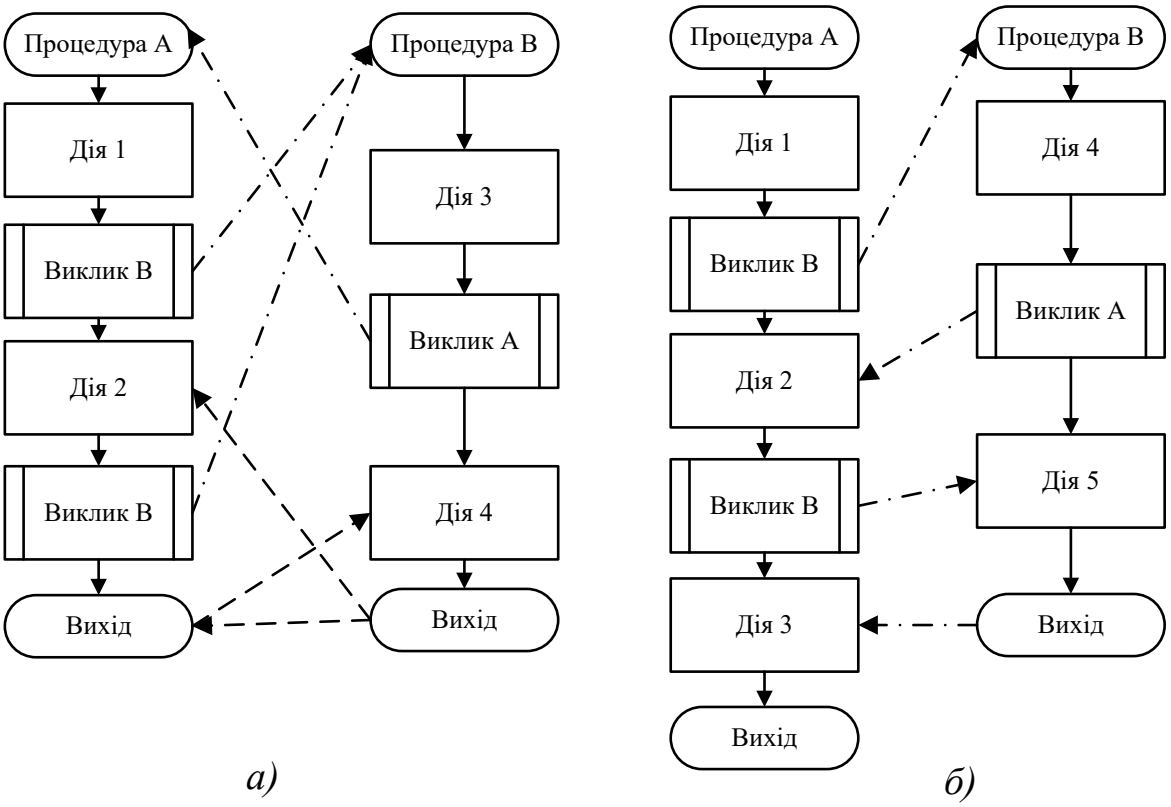

Рисунок 7.1 – Взаємний виклик процедур: а) звичайний; б) як співпроцедур. Штрих-пунктир – операція виклику, пунктир – повернення

Для організації взаємодії співпроцедур використовується тимчасовий регістр обміну даними для адреси повернення з процедури.

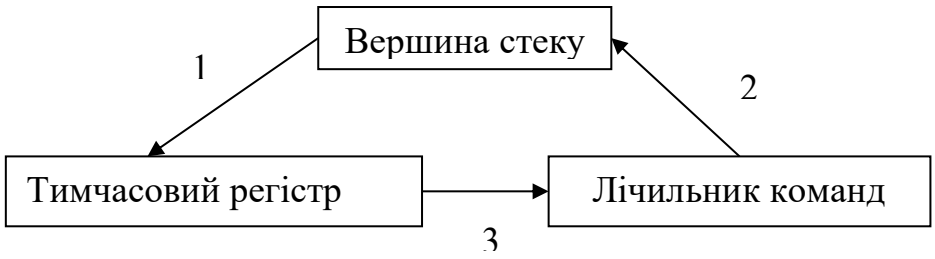

Рисунок 7.2 – Організація роботи співпроцедур

Перший раз процедура А викличе процедуру В звичайним чином. Далі виклики співпроцедур передбачають використання не операції виклику, а наступні кроки:

1. Перемістити стару адресу повернення з процедури із вершини стека у тимчасовий регістр.
2. Завантажити вміст лічильника команди у вершину стека.
3. Помістити вміст тимчасового регістра у лічильник команд.

Останній виклик співпроцедури (на рисунку 7.2 процедури А) не передбачає виконання кроку 2 або додаткову операцію по вилучення однієї адреси повернення зі стеку.

Таким чином, при використанні співпроцедур стек не нарощується і для їх взаємних викликів, які у даному випадку будуть називатись відновленнями, лічильнику команд буде надаватися потрібна адреса, тобто адреса команди після дій з відновлення.

7.3 Робота з циклами

Цикли як одна з конструкцій програм складаються із заголовку, де задано і оброблюються умови продовження циклу чи виходу з нього і тіла, де, власне, виконуються корисні для вирішення поточної задачі обчислення.

Цикли можна організувати різними методами в залежності від потреб, тому навіть на традиційно машинному рівні є кілька критеріїв їх класифікації [36-38]:

1. За порядком перевірки умов виконання є цикли з передумовою і післяумовою. Цикл з післяумовою завжди виконується не менше одного разу на відміну від циклу з передумовою.

2. Продовження за умовою перевірки результату. Цикл може продовжувати виконувати своє тіло або припинити це за умовою «істинно» чи «хибно».

3. За критерієм визначеності кількості кроків цикли розділяють на параметричні та непараметричні. Перші зазвичай використовують для роботи з масивами даних, другі – для обчислень рядів, біномів та інших алгоритмів, що повинні дати результат потрібної точності за невизначену, але кінцеву кількість кроків. Тобто непараметричні цикли можуть застосовуватись для тих же задач, що і рекурсія.

Приклад непараметричного циклу з передумовою:

```
; while (i < n) { sum += a[i]; i++; }
; Вхід:
;   ESI = адреса масиву a (байти)
;   ECX = n (довжина)
; Вихід:
;   EAX = sum
; Реалізація: i в EDX

sum_while:
   xor     eax, eax          ; sum = 0
   xor     edx, edx          ; i = 0

while_check:
   cmp     edx, ecx          ; i < n ?
   jae     while_end         ; якщо i >= n => вихід
```

```
    movzx   ebx, byte [esi+edx] ; завантажити a[i] у EBX (розширення до 32 біта)
    add     eax, ebx          ; sum += a[i]
    inc     edx               ; i++

    jmp     while_check       ; назад до перевірки
while_end:
    ret
``
```

У найбільш поширеній для вивчення здобувачами ступеню бакалавра мові програмування С в якості заголовка для параметричних циклів з передумовою використовують оператор for, для непараметричних – while та пару do…while для циклів з перед та післяумовою відповідно [41]. При цьому всі ці конструкції передбачають продовження циклічних обчислень за умовою «істинно». Обмеження не є строгими, оскільки можливо використати for для організації непараметричних і, у свою чергу, while та do…while для параметричних обчислень, хоча часто це нераціонально. Також не варто забувати і про використання у тілі циклу операторів break та continue. Перший дозволяє організувати додатковий непараметричний вихід навіть з параметричного циклу, наприклад, для запобігання помилок чи виключень під час роботи програми. Другий – пропустити поточний крок циклу без подальших обчислень у його тілі.

У мові С є також інструмент переходу за мітками goto…label, що дозволяє найбільш точно конфігурувати і структуру, і порядок перевірки, і умову продовження роботи циклу. Але цей метод не рекомендується до використання у обчислювальних машинах, де є рівень операційної системи. Причини такого обмеження пов’язані з організацією віртуальної пам’яті, що буде розглянуто у подальших лекціях.

7.4 Обробка переривань прцесора

На відміну від операцій виклику, необхідність у обробці переривань виникає асинхронно до основного потоку обчислень і явно порушує послідовність алгоритмів. Для обробки переривань необхідно безумовно змінити зміст лічильника команд, але виконує це не сам по собі центральний процесор обчислювальної системи, а у взаємодії із окремим пристроєм – **контролером переривань**. Контролер переривань виявляє події від пристроїв консолі, жорстокого диску, системного таймера на материнській платі та формує процесору сигнал запиту на переривання через виділені для цієї мети провідники, після чого надсилає по системній шині адресу переходу на процедуру-обробник переривання. Ця адреса називається **вектор переривання** і зберігається у так званій **таблиці векторів переривань** в пам’яті контролера переривань [29]. Для повернення з обробника переривання використовується безадресна операція повернення з переривання. Однокристальні мікро-ЕОМ серій PIC, AVR,

x51 та їм подібні містять контролер переривань в одному кристалі з центральним процесором [35, 36]. У загальному випадку, кожен обробник переривань повинен в обов'язковому порядку виконати наступні дії:

а) заборонити будь-які переривання процесора;

б) зберегти, зазвичай у стеку, поточні значення всіх регістрів процесора;

в) виконати корисну роботу, тобто очікувану обробку переривання;

г) відновити дані, що містилися у регістрах процесора на момент надходження запиту на переривання;

д) дозволити переривання процесора та повернутись до виконання основного обчислювального потоку.

Насправді, існують різні підходи до використання переривань. Тому вказана вище процедура може й не витримуватись, але це ускладнить роботу і знизить надійність програмного забезпечення.

Приклад встановлення і використання обробника переривань від системного таймера IRQ0 → INT 08h

```
; ------------------------------------------------------------------
; Встановлення нового обробника переривання INT 08h
; (системний таймер)
; ------------------------------------------------------------------

org 100h

start:
   cli              ; Забороняємо переривання, щоб безпечно змінити вектор
   xor ax, ax
   mov es, ax       ; ES = 0000h – сегмент таблиці векторів переривань

   mov ax, offset new_int08
   mov [es:08h*4], ax  ; Заносимо offset нового обробника
   mov ax, cs
   mov [es:08h*4+2], ax ; Заносимо segment нового обробника
   sti              ; Дозволяємо переривання

   mov ax, 4C00h
   int 21h          ; Завершення програми

; ------------------------------------------------------------------
; Новий обробник переривання INT 08h
; ------------------------------------------------------------------

new_int08:
   cli              ; (а) Заборонити переривання, щоб уникнути вкладених IRQ
```

```
    push ax             ; (b) Зберігаємо всі регістри
    push bx
    push cx
    push dx
    push si
    push di
    push ds
    push es

    mov ax, cs
    mov ds, ax          ; Встановлюємо сегменти (для безпеки)

    ; (c) Корисна робота — наприклад, інкремент лічильника
    inc byte ptr [tick_count]

    ; ---------------------------------------------------------------
    ; Надсилаємо контролеру переривань 8259A команду End Of Interrupt
    ; ---------------------------------------------------------------
    mov al, 20h
    out 20h, al         ; Підтвердження PIC, що переривання оброблено

    ; (d) Відновлюємо регістри
    pop es
    pop ds
    pop di
    pop si
    pop dx
    pop cx
    pop bx
    pop ax

    sti                 ; (e) Дозволяємо переривання
    iret                ; Повернення з обробника переривання

tick_count db 0         ; Глобальний змінний лічильник
```

7.5 Обробка системних викликів

Існує певна термінологічна плутанина між поняттями «переривання» і «системний виклик». Деякі автори вводять навіть термін «програмне переривання», для якого передбачено машинну та асемблерну команду:

INT n; де n – номер функції операційної системи DOS (застаріла, але обмежено використовується) або BIOS

Насправді, це ніякий не виклик переривання. Це взагалі не категорія традиційно машинного рівня. Це виклик сервісної функції за номером n операційної системи. На відміну від переривання, системний виклик – внутрішня синхронна подія, тобто її появу можна передбачити. Переривання також іноді може бути синхронною подією, коли йде мова про порушення прав доступу до ресурсів чи діленні на нуль, але це нештатний режим, якого слід запобігати.

Таким чином, системний виклик – це звернення до операційної системи виконати деякий рутинний машинний код, який міститься у бібліотечних ресурсах операційної системи і пов'язаний, найчастіше, з усвідомленим доступом до консолі, жорсткого диску, до таймера і таке інше.

Для прикладу розглянемо код виведення рядка "Hello!" за допомогою функції DOS 21h [37]:

```
format MZ                ; Формат EXE-файлу.
entry code_seg:start     ; Точка входу у віртуальну пам'ять програми.

segment data_seg         ; Початок сегменту даних (про двовимірну
                         ; модель віртуальної пам'яті й
                         ; сегменти йтиметься у розділі 9).
msg db 'Hello!$'         ; Рядок "Hello!".

stack 200h               ; Розмір стеку складає 200 машинних слів.

segment code_seg         ;Початок сегменту коду (пам'яті програми).

start:                   ; Мітка початку виконання.
  mov ax, data_seg       ; Взяти адресу сегменту даних.
  mov ds, ax             ; і розмістити її у спеціалізований регістр DS
 mov dx, msg             ; Помістити вказівник на рядок "Hello!" у
                         ; спеціалізований регістр DX.

 mov ah, 9h              ; Викликати DOS-функцію 9h для виведення
 int 21h                 ; строки з адреси DS:DX на екран.

 mov ah, 4ch             ; Викликати DOS-функцію 4ch для
 int 21h                 ; завершення програми
```

У наведеному лістингу використовується системний виклик для виведення рядка, початкова адреса якої попередньо завантажена у регістри DS:DX, на екран. Регістри DS:DX та акумулятор мають спеціалізоване призначення для налаштування роботи функції 21h. Та ж сама функція використана і для завершення роботи програми, але для цього в регістр AX розміщено код 4ch, а не 9h, а пара DS:DX не використовується.

Є багато інших системних викликів DOS, BIOS, Linux, Windows, які реалізовані не лише засобами асемблера. Важливо пам'ятати, що це не справжні переривання (IRQ), а лише сервісні функції операційних систем чи базової системи введення-виведення, які можуть використовуватись у складі коду обробників переривань, а також у складі коду інших системних чи прикладних програм.

7.6 Висновки

Хоча у даному розділі йшла мова про мовні конструкції традиційно машинного рівня ЕОМ, для їх демонстрації були задіяні засоби, притаманні й іншим рівням. Так, терміни «процедура», «підпрограма» та «функція» мають одну й ту саму сутність і на асемблерному, і на традиційно машинному, і на рівні операційної системи. З іншого боку, поняття «макроси» має сенс на асемблерному та рівні прикладної програми. Виходячи із сутності та призначення процедури у основному алгоритмі обчислювальної системи, відомі різні способи зберігання адреси повернення із її тіла до основного алгоритму, але основним на сьогодні є зберігання адреси повернення у стековій пам'яті. Спосіб вважається основним для ЕОМ через можливість багаторівневого вкладення викликів процедур у тіла інших процедур та можливості рекурсії. Технологія програмування зі співпроцедурами відома, але з існуванням механізму переривань вона не є вживаним способом організації паралельних обчислень.

У розділі показано, що технологія рекурсії не є переважаючою порівняно з використанням циклів. Це пояснюється додатковими витратами часу на вивантаження рекурсивних адрес повернення з кроків рекурсії зі стеку та, власне, читанням і дешифрацією рекурсивних команд повернення.

Під час розгляду технології циклів застосовано синтаксис мови С рівня прикладних програм через більші можливості з точки зору демонстрації. Показано властивості циклів та призначення з точки зору практичного застосування.

Технологія переривань, як така, порушує одну з базових властивостей алгоритму програми або системи – строгу послідовність дій, оскільки вимагає негайної реакції на деяку зовнішню, асинхронну по відношенню до обчислювальної системи, подію. Ця технологія використовує особливий апаратний засіб – контролер переривань з таблицею векторів переривань, а також програмні обробники переривань, які є фактично процедурами. Використання переривань дозволяє на сьогодні створювати багатозадачні операційні системи.

Системні виклики або ж програмні переривання, хоча і можуть звертатися до апаратних переривань, перериваннями насправді не є. Це лише заздалегідь підготовлені сервісні функції операційної системи, представлені у вигляді процедур у вигляді бінарного коду. Цей код розташовано у файлах бібліотек операційної системи або, у деяких випадках, у її монолітному чи гібридному ядрі в пам'яті.

7.7 Питання для самоперевірки

1. Що таке процедура?
2. У чому відмінність процедури від макросу?
3. Які способи зберігання адреси повернення з процедури існують?
4. Що таке рекурсія?
5. Що таке співпроцедури?
6. З чого складається цикл як конструкція мови програмування?
7. На які види класифікуються цикли за порядком перевірки умов виконання?
8. Які два результати перевірки умови продовження виконання тіла циклу можуть бути на традиційно машинному рівні?
9. Як класифікуються цикли за критерієм визначеності кількості кроків?
10. Для якого типу циклів за критерієм визначеності кількості кроків використовується оператор for?
11. Для якого типу циклів за критерієм визначеності кількості кроків використовується оператор while?
12. Для якого типу циклів за критерієм визначеності кількості кроків використовується оператор do…while?
13. За порядком перевірки умов виконання яким є цикл for?
14. За порядком перевірки умов виконання яким є цикл do…while?
15. Як у циклах використовують оператори break та continue?
16. Який пристрій в обчислювальній системі здійснює запит на зміну вмісту лічильника команд під час запиту на переривання?
17. Що зберігає таблиця векторів переривань?
18. Які події здатен виявити контролер переривань?
19. Яким є загальний алгоритм обробника переривань?
20. Що таке системний виклик?

8 ВІДОБРАЖЕННЯ ДАНИХ НА ТРАДИЦІЙНО МАШИННОМУ РІВНІ

8.1 Прості типи даних

8.1.1 Тип даних для відображення цілих чисел

Це найбільш часто використовуваний тип даних. Розмірність цілочисельних типів складає 1 байт, 2 байти, 4 байти, 8 байт і існує тип для розміщення 16-байтного числа. У будь якому разі, цілий тип дозволяє розміщувати 2^n чисел, де n – розмірність комірки пам'яті у бітах для зберігання цілого числа. Цілі числа розділяють на знакові та беззнакові. Діапазон значень беззнакових складає від 0 до 2^n-1 включно, тоді як діапазон значень знакових – від -2^{n-1} до $+2^{n-1}$-1. Це пояснюється тим, що старший розряд знакового числа віддається під знак, і 1 у старшому бітові комірки пам'яті змінної означає саме «мінус». Діапазон не зменшується, а зміщується наполовину у від'ємний відрізок осі абсцис (осі координат X). При цьому число -0 – досить нормальне з точки зору традиційно машинного рівня, але при програмуванні воно здатне принести деякі неприємності. Наприклад, мова Visual Basic для багатьох застосувань не підтримує беззнакові цілі. У такому випадку можливо, наприклад, що переданий двохбайтний «-0», який з точки зору скрипта Visual Basic є нормальним нулем, застосуванням чи приладом із використанням беззнакових змінних буде сприйнятий як число «32 768». Таким чином, при написанні програм варто враховувати, з якими пристроями і програмами передбачено взаємодію. У наведеному прикладі для представлення змінної варто було б взяти її по модулю.

Різні компілятори прикладного рівня програмування можуть використовувати дещо різні діапазони цілочисельних даних із, здавалося б, однаковими назвами. Так, для С за стандартом ANSI дане типу int розміщується у 4 байта, а для мови Паскаль – у 2. Два байти передбачено для даного типу int також і в компіляторах С для деяких промислових контролерів та однокристальних мікро-ЕОМ. У будь якому випадку, програміст перед початком використання будь якого інструменту програмування має ознайомитись із особливостями типів даних, як їх сприймає компілятор. У таблиці 8.1 наведено сучасні типи цілих даних мови С ANSI [41].

Таблиця 8.1– Типи цілочисельних даних мови С

Тип даних, формат об'явлення для С	Діапазон значень та пояснення	Формат виведення для С
char, signed char	Цілочисельний, найменший з можливих адресованих типів. Містить 8 біт. Може приймати значення з діапазону -128 ... + 127	%d, %i, %x, %o

Продовження таблиці 8.1

Тип даних, формат об'явлення для С	Діапазон значень та пояснення	Формат виведення для С
unsigned char	Того ж розміру що й char, але без знаку. Діапазон 0 ... 255.	%d, %i, %x, %o
short, short int, signed short, signed short int	Тип короткого цілого числа зі знаком. Може містити числа з діапазону -32768 ... 32767. Розмірність - 2 байти.	%d, %i
unsigned short unsigned short int	Такий же, як short, але беззнаковий. Діапазон: 0 ... +65535	%u
int, signed, signed int	У компіляторах для 32-розрядних, та обчислювальних платформ більшої розрядності має розмір 4 байти і діапазон -2 147 483 648… +2 147 483 647, однак на 16- і 8-бітних платформах має розмір, як правило, 2 байти в діапазоні значень -32768...+32767, що часто призводить до несумісності неакуратно написаного коду	%d, %i
unsigned, unsigned int	Такий же як int, але беззнаковий. Діапазон: 0…+4 294 967 295	%u
long, long int, signed long, signed long int,	Тип довгого цілого числа зі знаком. Може містити числа в діапазоні -2 147 483 648…+2 147 483 647. Таким чином, це принаймні 32 біта (4 байт).	%li, %ld
unsigned long, unsigned long int	Такий же як long, але беззнаковий. Діапазон: 0…+4 294 967 295.	%lu
long long, long long int, signed long long, signed long long int	Тип довгого довгого (подвійного довгого) цілого числа зі знаком. Може містити числа в діапазоні -9 223 372 036 854 775 808… +9 223 372 036 854 775 807. Таким чином, це 64 біти. Тип затверджено в стандарті С99, а не в ANSI.	%lli
unsigned long long, unsigned long long int	Схожий на long long, але беззнаковий. Діапазон: 0…18 446 744 073 709 551 615.	%llu

8.1.2 Тип даних для відображення символів

Це вже показаний вище тип char, але тут з кожним можливим значенням змінної асоційовано деякий графічний символ цифри, букви, іншого позначення чи недрукованого символу. Графічні символи зібрані у асоційовану таблицю ASCII. ASCII (англ. American standard code for information interchange) – була розроблена і стандартизована в США, в 1963 році. На сьогодні для

мережевих платформ і персональних комп'ютерів більш широко використовується тип даних wchar_t, також цілочисельний, але орієнтований виключно на представлення символів [41]. Враховуючи, що його розмірність становить 2 байти, можливості представлення інформації людині з урахуванням нових спецсимволів та національних алфавітів значно розширились. Тип wchar_t асоційований з таблицею символів UNICODE.

8.1.3 Логічний тип даних

По суті, це також цілочисельний тип даних, у якому допустимі лише два значення. З цими значеннями пов'язано дві інформаційні ознаки «істинно» та «хибно». Важливо відмітити, що логічні типи різних компіляторів, а також для різних програмно-апаратних платформ, можуть відрізнятись. Так стандартний тип bool займає 1 байт, хоча у 32-розрядних платформах він приведений до 4 байт. Тип BOOL – строго 4 байти. Є також тип _Bool. Особливість логічного типу у тому, що «хибно» тут для більшості компіляторів нуль, а «істинно» – не нуль. Тобто більше або менше нуля. При цьому діапазон значень для «істинно» bool 1…255 або -127…-0 та 1…127, у той час як для BOOL: 1…+4 294 967 295 або -2 147 483 648…-0 та 1…2 147 483 647 [42]. _Bool аналогічний bool, але з урахуванням ознаки переповнення, тобто тут 256, -128 та 128 також «істинно». Таким чином, при використанні бібліотечних функцій під час програмування бажано використовувати саме той логічний тип, який зумовлений для вхідних чи вихідних даних цієї функції.

8.1.4 Вказівники

Цей цілочисельний беззнаковий тип даних використовується для непрямої адресації пам'яті програм чи даних. Його розмірність співпадає із розрядністю обчислювальної системи і може складати 8, 16, 32, 64 біт. Вважаючи, що сучасні процесори 32 та 64-розрядних обчислювальних машин апаратно підтримують двовимірну віртуальну модель пам'яті, вказівники дозволяють звертатись до, відповідно, $2^{32}\times2^{32}$ та $2^{64}\times2^{64}$ машинних слів пам'яті по 32 та 64 біти [29].

8.1.5 Порожній тип даних

На традиційно машинному рівні такий тип використовується перш за все для допоміжних дій з перетворення типів та форматів з використанням операцій передачі даних та деяких двомісних операцій у тому числі співпроцесора. У мовах програмування високого рівня порожній тип, крім того, використовується для: визначення функцій, які не повертають значення; для вказання порожнього списку аргументів функції; як базовий тип для вказівників [43].

У будь-якому випадку, коли йде мова про перетворення типів та покажчики, котрі на машинному рівні всі безтипові, необхідно мати чітке уявлення про формати типів даних, а також архітектуру обчислювальних засобів. Це необхідно, щоб коректно використовувати операції з порожнім типом при вирішені означеного кола задач.

8.1.6 Типи даних для відображення чисел з плаваючою крапкою

Внутрішнє відображення дійсного числа у пам'яті комп'ютера відрізняється від цілого числа. Число з плаваючою крапкою має експоненційну форму:

$$\pm mE \pm p,$$

де m – мантиса (ціле або дробове число з десятковою крапкою), p – порядок (ціле число). Для того щоб перевести число з експоненційної форми до звичайного представлення з фіксованою крапкою, необхідно мантису помножити на десять в ступені порядок, наприклад:

$$-6.42E + 2 = -6{,}42 \times 10^2.$$

Діапазони значень речових типів показано у таблиці 7.2:

Таблиця 7.2 – Діапазон значень чисел з плаваючою крапкою [41]

Тип даних	Діапазон значень	Розмір (байт)
float	±3.4E-38 ... 3.4E + 38	4
double	±1.7E-308 ... 1.7E + 308	8
long double	±3.4E-4932 ... 3.4E + 4932	10

Довжина мантиси визначає точність числа, а довжина порядку – його діапазон.

Дані типу float займають 4 байти, з яких 1 двійковий розряд відводиться під знак, 8 розрядів – під порядок і 23 – під мантису. Оскільки старша цифра мантиси у двійковій системи числення завжди дорівнює 1, вона не зберігається [41].

Дані типу double займають 8 байт, в них під порядок – 11 розрядів, під мантису – 52 розряди відповідно, 1 двійковий розряд відводиться під знак [41]. Тип double на сьогодні вважається основним для 32- та 64-розрядних систем і таким, що забезпечує основні вимоги до точності обчислень. Тому для обчислень не має сенсу використовувати тип float. З іншого боку, тип float використовується для малопотужних обчислювальних систем, контролерів, інтелектуальних речей – там, де потрібна швидка реакція, але не надто висока точність, наприклад, для задач автоматичного регулювання, ідентифікації, первісної обробки даних від давачів та засобів вимірювання і таке інше.

Специфікатор типу long перед ім'ям типу double вказує, що під величину відводиться 10 байт. З цих 10 байт: 64 біти займає мантиса, 1 біт – знак, 15 біт – порядок числа [41].

8.2 Складні типи даних

8.2.1 Масиви даних

Це складний тип даних, що може поєднувати групу даних одного типу. Цим типом може бути будь-який з простих або складних типів. Для простоти викладення, тут в якості прикладів, будемо розглядати цілочисельний тип.

На традиційно машинному рівні масив може бути представленим двома методами: за допомогою інформаційного вектору та методом маргінального індексування [29].

Інформаційний вектор – простий спосіб збереження даних одного типу, при якому дані розміщуються на ділянці пам’яті послідовно, одне за одним і без розривів. При цьому для звернення до елементів масиву здійснюється за допомогою команд із непрямою адресацією. Одновимірний масив, який і називається вектором, використовується для обчислень найчастіше у тому числі і через простоту організації.

Розглянемо інформаційний вектор для двовимірного масиву, тобто двовимірної матриці. Тут елементи на ділянці пам’яті будуть розміщені, як показано на рисунку 8.1:

Рисунок 8.1 – Зберігання двовимірної матриці 3×4 у пам’яті обчислювальної системи

Варто зазначити що термін «рядок» або «стовпчик» з точки зору зберігання масиву у пам’яті даних досить умовні, оскільки представлення масиву в інформаційному векторі все одно одновимірне, як видно з рисунка 8.1. Ми могли б вважати, що у пам’яті послідовно розміщено в одному векторі не кілька векторів рядків, а кілька векторів стовпчиків. Це все одно, але важливо, щоб при розробці однієї програми координати усіх масивів трактувались однаково. Тобто слід вважати, що всі масиви у пам’яті програми розміщені порядково або всі постовпцево. І, оскільки терміни «рядок» або «стовпчик» – поняття усього лише відображення, виводити двовимірний масив на екран порядково, постовпцево, у вигляді прямої матриці чи транспонованої – це лише питання алгоритму відображення.

Розглянемо взаємозв’язок масивів високорівневого програмного коду та їх представлення на машинному рівні. Нехай є двовимірний масив цілих чисел:

int A[5,6].

Тоді для копіювання до змінної x до i-го елементу j-го стовпця масиву A на мові C достатньо зробити запис:

int x = A[i, j].

При цьому, вважаючи, що інформаційний вектор складається з підвекторів рядків, та при визначенні вказівника

int * a = A;

на традиційно машинному рівні наведений код буде еквівалентний наступному запису на мові C:

int x = (a+i*5+j)*.

Запис int * a = A еквівалентний int * a = A[0,0] і означає отримання вказівника на перший елемент інформаційного вектору, який одночасно є першим, а враховуючи вимоги мов програмування, елементом нульового стовпчика нульового рядка масиву A. Тобто, на машинному рівні для матриці m×n для звернення до її i-го елементу j-го стовбця необхідно до регістру, де зберігається початкова (базова) адреса матриці, додати зміщення s, що обчислюється за формулою:

s = i*m+j,

та скористатися командою з непрямою адресацією із застосуванням вказаного регістру.

Аналогічно, якщо мова йде про тривимірний масив m×n×q, де інформаційний вектор складається із послідовно розміщених підвекторів двовимірних масивів, а ці, у свою чергу – із одновимірних, то тут зміщення s:

s = i*m*n+j*n + k,

де k – координата елемента масиву B[m,n,k], при повних базисних координатах (i;j;k).

Тобто, для обчислення зміщення s для звернення шляхом непрямої адресації до елементу N-мірного інформаційного вектору у пам’яті програм справедлива залежність:

$$\boldsymbol{s = a_1 \prod_{i=1}^{N-1} b_i + a_2 \prod_{i=2}^{N-1} b_i + \cdots + a_N}$$

де $\boldsymbol{a_1},\ \boldsymbol{a_2}, \ldots \boldsymbol{a_N}$ – координати елемента у масиві, $\boldsymbol{b_1},\ \boldsymbol{b_2}, \ldots \boldsymbol{b_{N-1}}$ – довжини базисів масиву, тобто «ширина», «довжина», «глибина» і наступні координатні базиси.

Зазвичай, навіть тривимірні масиви для обчислень застосовуються нечасто, здебільшого при роботі з графікою. Крім того для більш складної організації даних використовують інші методи.

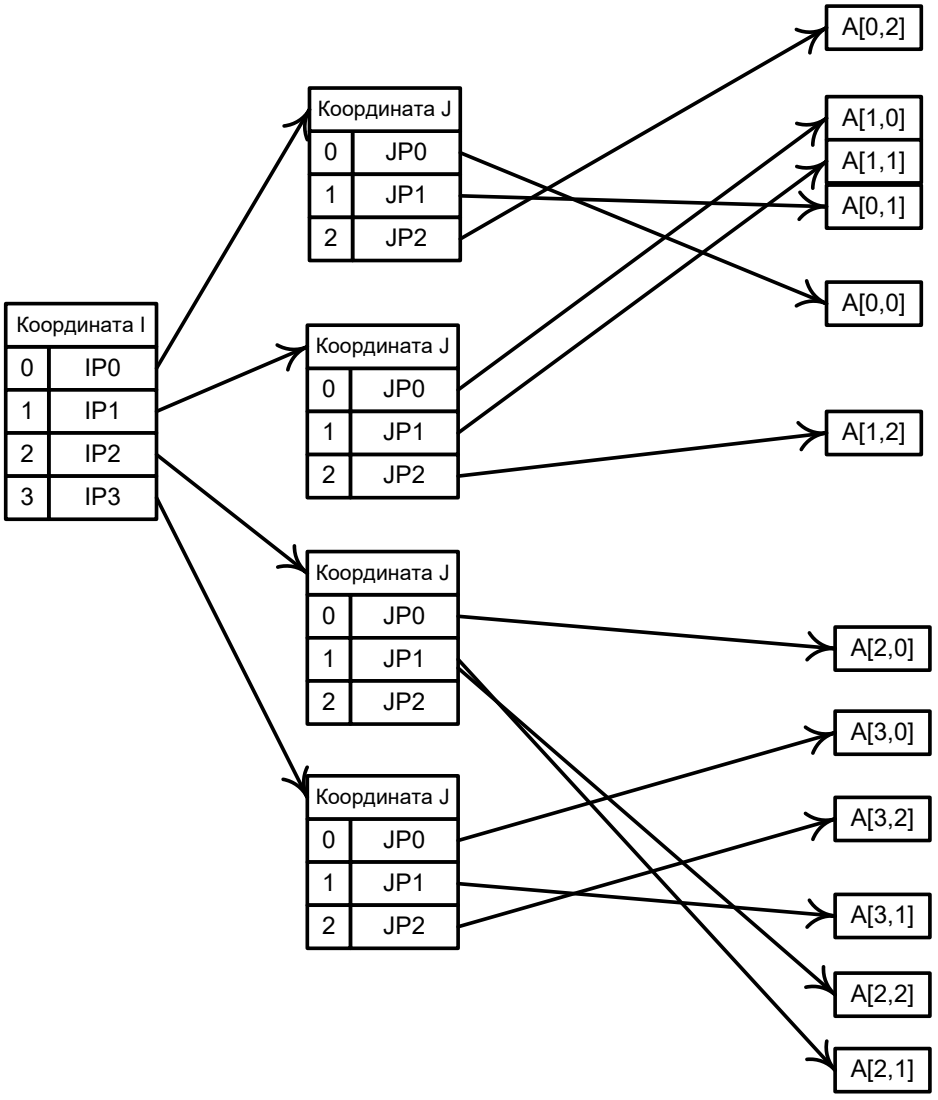

Рисунок 8.2 – Масив А розмірністю 3×4, побудований методом маргінального індексування

Маргінальне індексування. Маргінальне (від. лат. marginal – граничний, обраний) індексування – це метод, що передбачає зберігання у інформаційних векторах не даних, а вказівників на дані. Самі дані при цьому можуть знаходитись у пам’яті даних будь-де. Такий, здавалося б, складніший спосіб організації масиву виправданий, коли у процесі виконання програми є необхідність часто змінювати розміри базисів масиву, об’єднувати великі масиви, розміщувати великі або складні масиви у обмеженому просторі пам’яті, коли елементи масиву – самі по собі дані складеного типу, наприклад рядка символів. У таких

випадках дійсно більш продуктивнішими будуть алгоритми, які маніпулюють вказівниками довжиною в одне машинне слово, а не даними по 10-12 байт чи рядками по 256 символів. Схема маргінального індексування масиву А розмірністю 3 виглядає, як показано на рисунку 8.2. На сьогодні метод знайшов своє відображення у технологіях узагальненого програмування, тобто програмування шаблонами. Це бібліотека ATL, а також класи CArray та CVector у бібліотеці MFC від Microsoft [43].

8.2.2 Структури даних

Цей тип даних представлений лише на рівні прикладного програмування. Він здатен об'єднувати дані різних типів. Тобто є інформаційний вектор, що вміщує вказівники на послідовність даних різної довжини і формату у тому числі і вказівники на дані деяких типів як при маргінальному індексуванні [41].

8.2.3 Рядки символів

Тип даних, досить часто використовуваний, оскільки обчислювальна техніка обробляє великі обсяги текстової інформації. Існує кілька способів зберігання рядки символів в пам'яті комп'ютера при роботі з ними [29]. Перший і найпростіший спосіб – зберігання безперервної рядки символів у вигляді послідовності байт для ASCII або двобайтних слів для Unicode. Рядок закінчується символом кінця рядка, який пов'язаний з нулевим байтом чи словом. Звертання до такого рядка здійснюється за вказівником, який вказує на перший, а точніше нульовий за порядковим номером, символ рядка. Для обробки такого рядка на машинному рівні будуть використовуватись команди роботи з рядками (рядка довжиною до 256 символів) та команди загального призначення із непрямою адресацією у непараметричних циклах. Продуктивність застосувань, що використовують такі рядки, знижується із збільшенням довжини рядка, оскільки всі операції, пов'язані з редагуванням рядка зі зміною її довжини призводять до необхідності переміщення всіх символів, що знаходилися за точкою редагування. Тому, хоча технічно можливо використовувати безперервні рядки, довші за 256 символів (для цього у тій же мові С є типи-вказівники на рядок, такі як LPSTR) цей підхід не є бажаним для текстів, що містять до і більше тисячі символів.

Другий відомий спосіб зберігання рядків символів – метод зв'язаного списку. Тут кожен символ зберігається у вигляді пари символ-вказівник. Вказівник при цьому вказує на наступний символ рядок або сам на себе, якщо цей символ останній. Символи у пам'яті можуть розміщуватись довільно, фрагментовано, а операції редагування взагалі не передбачають переміщення символів – лише перепризначення вказівників. Насправді це не дуже ефективний метод, оскільки при кожній спробі прочитати чи отримати доступ до кожного символу, треба виконати додаткові дії з читання його адреси. Це значні втрати процесорного часу, а крім того доступ до символів у рядках лише послідовний, тобто не можна причитати N-й символ строки, не прочитавши попередні N-1

символів. Крім того, на машинному рівні цей підхід не підтримується командами роботи з рядками, що робить застосування зв'язаних списків у чистому вигляді для роботи з рядками символів непридатним.

Рисунок 8.3 – Редагування розрідженого рядка

Третій, досить продуктивний спосіб роботи з рядками символів – використання розріджених рядків. Іншими словами рядок, кінець якого позначено відомими символами «кінець рядка» та «переведення каретки», розріджується порожніми комірками пам'яті, приблизно 3-4 комірки на 5 символів. Наведемо приклад редагування розрідженого рядка, скориставшись рядками твору Г. С. Сковороди. Для наочності представлення тут символ пробілу між словами позначимо як «_», а порожня (нульова) комірка пам'яті, тобто розрідження, буде позначатись порожньою клітинкою. На рисунку 8.3 представлений один рядок.

Як видно з рисунку 8.3, додання чотирьох символів на кроці 1 призвело до трьох операцій з переміщення символів замість 29 для аналогічного нерозрідженого рядка, а додання двох символів на кроці 2 – до п'яти переміщень замість 10. Загальна довжина розрідженого рядка взагалі не змінилася. Таким чином розрідження рядка зменшує наслідки редагування рядка зі зміною кількості символів у ній з точки зору необхідності переміщення символів за точкою редагування. Інакше кажучи, продуктивність додатку, що працює із розрідженими рядками вища. Недолік методу – необхідність періодично переглядати текст, що редагується, та перерозподіляти, додавати та вилучати зайві

розрідження. Також виникають певні складнощі при виконанні порівняння текстових рядків, оскільки у двох однакових за змістом рядках можуть бути по різному розподілені розрідження. Таким чином, спосіб мало придатний у чистому вигляді для великих текстів на десятки тисяч рядків.

Четвертий спосіб представлення рядків символів – комбінований. Передбачається що є зв'язаний список коротких, наприклад, по 256 символів рядків, які у загальному тексті посилаються одне на одне. Можуть використовуватись і розріджені рядки. Вказівники на рядки можуть міститися у самих рядках, а можуть бути зібрані в окрему таблицю дескрипторів (тобто вказівників) рядків. Як правило, під час редагування тексту, робота виконується над невеликими ділянками, тобто окремими рядками списку. Тому переміщувати всі символи тексту після точки редагування немає потреби. На сьогодні це найбільш застосовуваний метод обробки великих обсягів текстової інформації.

8.3 Висновки

Викладений у цьому розділі матеріал дозволяє відслідковувати зв'язок між форматами даних на традиційно машинному рівні та їх типами на рівні прикладного програмування . Так, стає очевидним, що найбільш часто уживані на прикладному рівні цілочисельні змінні з точки зору програмування можуть мати різне призначення: представляти цілі числа, коди символів, бути вказівниками непрямої адресації, не мати певного призначення або підтримувати двійкову логіку, хоча при цьому займати машинне слово, а не окремий біт. Варто також відмітити можливість існування у прикладному програмному коді змінних, тип яких має розмірність меншу, ніж розрядність машинного слова обчислювального засобу, де йому відповідний відтрансльований код буде використано. При цьому з точки зору раціонального використання обчислювальних ресурсів тип даних з розрядністю менше доступного машинного слова не має смислу – все одно знадобиться цільне машинне слово. Ця обставина призвела до виникнення технологій віртуальних потоків на кшталт hyper treading та використання SIMD, MMX, SSE, 3DNow! і типу QWord.

Окремо слід зазначити також типи даних стандарту C99: цілочисельні long long знакові й беззнакові, а також long double. Застосування системи команд x64 для обробки таких типів даних здатне підвищити продуктивність обчислень, але коло задач, де б знадобились обчислення такої розрядності й точності обмежене. Навряд чи у задачах комерційних розрахунків, керування технологічними і виробничими процесами, у створенні й обробці медіаконтенту знадобляться такі типи даних у найближчий час.

Щодо складних типів даних, тут можна виділити задачі зберігання масивів даних простого чи складного, але одного типу, зберігання структурованих даних різних типів, а також зберігання і обробка рядків символів. Ці три задачі вимагають своїх підходів орієнтованих можливо на компактність, а можливо на швидкість чи зручність читання, запису, зберігання, редагування чи зміни обсягів.

8.4 Питання для самоперевірки

1. Для чого використовується цілочисельний тип?
2. Які розмірності цілочисельних типів використовуються?
3. Як визначити діапазон цілочисельного типу за відомою розмірністю у байтах?
4. Який тип використовується для зберігання інформації про символи?
5. Що собою являє логічний тип?
6. Для чого використовують вказівники?
7. Для чого використовують порожній тип даних на мові C++?
8. В якій формі у пам’яті комп’ютера представлене число з плаваючою крапкою?
9. Які типи даних у мові C++ передбачені для чисел з плаваючою крапкою?
10. Чому в типі даних з плаваючою крапкою не зберігають старший двійковий розряд мантиси?
11. Які методи організації збереження даних у масиві існують на традиційно машинному рівні?
12. Як розрахувати зміщення i-ї строки в інформаційному векторі m×n?
13. У чому полягає метод маргінального індексування?
14. Які переваги методу маргінального індексування?
15. Які програмні інструменти існують в C++ та MFC для організації масивів методом маргінального індексування?
16. Які методи використовуються для роботи з рядками символів?
17. Чому використання безперервного рядка обмежене довжиною до 256 символів?
18. Чому метод зв’язаного списку не використовується у чистому вигляді для зберігання й редагування рядків символів?
19. У чому полягає перевага розрідженого рядка символів у порівнянні з безперервною?
20. Опишіть комбінований метод зберігання й обробки рядків символів.

9 ПАМ'ЯТЬ ЕЛЕКТРОННО-ОБЧИСЛЮВАЛЬНИХ ПРИСТРОЇВ НА АПАРАТНОМУ РІВНІ

Пам'ять комп'ютерів є однією з необхідних складових для їх роботи. Існує багато критеріїв класифікації пам'яті, що дозволяють описати призначення пам'яті, принципи її побудови та функціонування. Не розглядаючи пам'ять як деякий функціональний вузол, тим не менш, на апаратному рівні її можна розділити на енергозалежну та енергонезалежну. Розглянемо сучасні принципи побудови енергозалежної пам'яті.

9.1 Енергозалежна пам'ять

Енергозалежна пам'ять цифрової обчислювальної машини — це пам'ять, здатна зберігати дані у комірках пам'яті лише за наявності напруги живлення. Тому її ще називають тимчасовою пам'яттю. Найменший обсяг даних, що може зберігатися – 1 біт. Найменша одиниця інформації – ознака 0 або 1, («Так» або «Ні»). У залежності від принципу реалізації енергозалежної комірки пам'яті пам'ять розділяється на статичну і динамічну.

9.1.1 Статична пам'ять

Статична пам'ять (SRAM, *static random access memory*) – напівпровідникова статична пам'ять довільного (інакше кажучи, прямого) доступу, в якій кожен двійковий розряд зберігається в схемі з позитивним зворотним зв'язком – тригері. Тригер – це схемотехнічний елемент, який може стабільно зберігати один з двох власних станів: «0» чи «1». Зберігати дані без перезапису SRAM може тільки поки є живлення. Типова принципова схема бітової комірки пам'яті SRAM показана на рисунку 8.1 [44]:

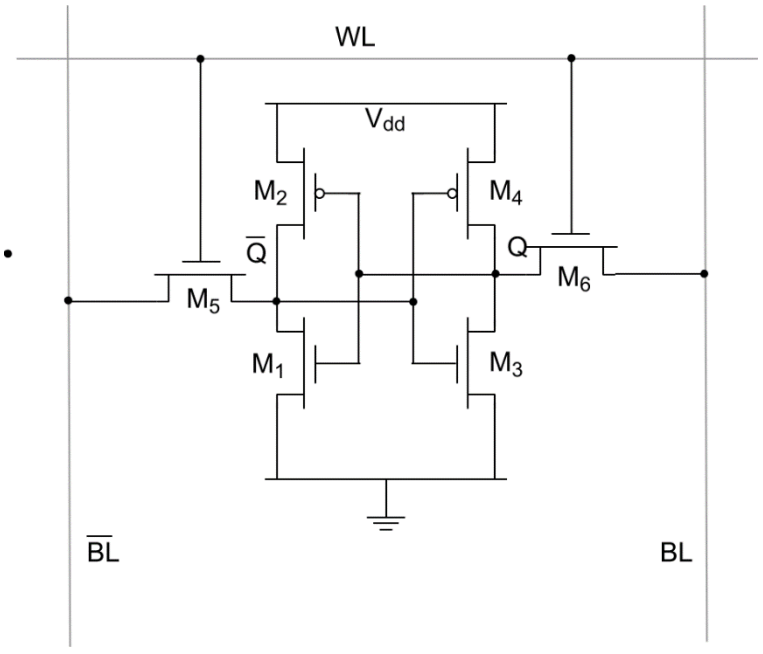

Рисунок 9.1 – Шеститранзисторна комірка статичної пам'яті

Комірка статичної двійкової пам’яті, або двійковий тригер на КМОП-технології, згідно рисунка 9.1, складається з двох перехресно (кільцем) включених інверторів і ключових транзисторів для забезпечення доступу до неї. Лінія WL (Word Line) керує двома транзисторами доступу. Лінії BL і $\overline{BL}$ (Bit Line) – бітові лінії, використовуються і для запису даних, і для читання даних.

Запис. При подачі «0» на лінію $\overline{BL}$ (запис нуля) або BL (запис одиниці) паралельно включені транзисторні пари (M5 і M1) чи (M6 і M3) утворюють логічні схеми 2АБО, тому наступне подання «1» на лінію WL відкриває транзистор M5 або M6, що призводить до відповідного перемикання тригера.

Читання. При подачі «1» на лінію WL відкриваються транзистори M5 і M6, рівні записані в тригері виставляються на лінії $\overline{BL}$ і BL і потрапляють на схеми читання.

Переваги статичної пам’яті у порівнянні з динамічною у більшій швидкодії через відсутність необхідності регенерації даних, що зберігаються, і через використання швидкодіючих транзисторних схем. Але статична пам’ять досить дорога у виготовленні, тому на сьогодні її використання обмежено кеш-пам’яттю та оперативною пам’яттю малопотужних обчислювальних систем. Останні модулі статичної пам’яті для персональних комп’ютерів, які виготовлялися масово сягали об’ємів 64…256 МВ, що на сьогодні неактуально.

9.1.2 Динамічна пам’ять

Динамічна пам’ять (DRAM, *dynamic random access memory*) – напівпровідникова динамічна пам’ять довільного доступу. DRAM широко використовується як оперативна пам’ять комп’ютерів, а також, за умови забезпечення безперебійного живлення, як постійне сховище інформації в системах, вимогливих до затримок.

Фізично DRAM складається з комірок-конденсаторів, створених в напівпровідниковому матеріалі, в кожній з яких можна зберігати певний обсяг даних, від 1 до 4 біт. Сукупність комірок такої пам’яті утворює умовний «прямокутник», що складається з певної кількості рядків і стовпців. Один такий «прямокутник» називається сторінкою, а сукупність сторінок називається банком. Весь набір комірок умовно ділиться на кілька областей.

Як пристрій, DRAM є модулем пам’яті різних конструктивів, що має електричну плату, на якій розташовані мікросхеми пам’яті і роз’єми для підключення модуля до материнської плати.

Принцип дії динамічної пам’яті показано на рисунку 9.2 [45]:

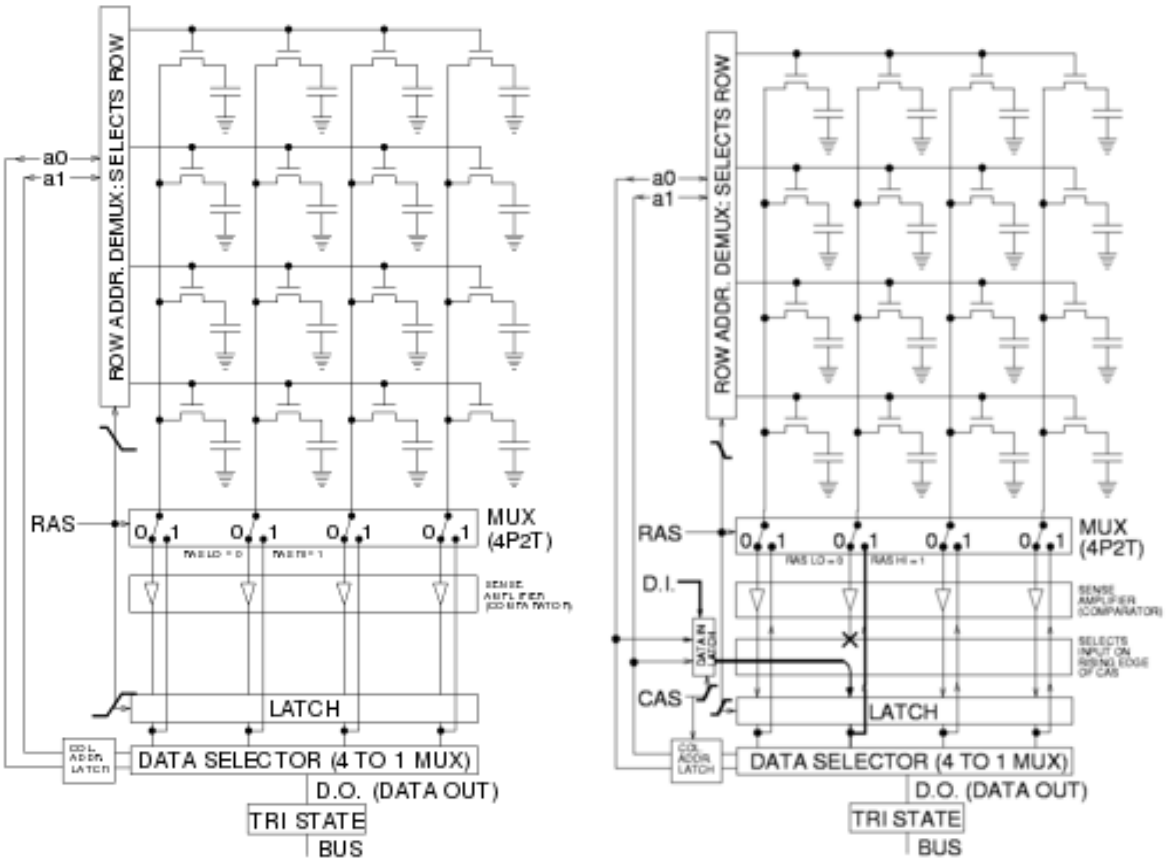

Рисунок 9.2 – Принцип дії (читання – зліва, запису – справа) DRAM для масиву 4×4

Фізично DRAM-пам’ять являє собою набір комірок пам’яті, що складаються з конденсаторів і транзисторів, розташованих усередині напівпровідникових мікросхем пам’яті.

При відсутності подачі електроенергії до пам’яті цього типу відбувається розряд конденсаторів за рахунок струмів витоку і пам’ять обнуляється. Для підтримки необхідної напруги на обкладках конденсаторів комірок і збереження їх вмісту, їх необхідно періодично заряджати через комутуючі транзисторні ключі. Така динамічна підтримка заряду конденсатора є основоположним принципом роботи пам’яті типу DRAM. Конденсатори заряджають, коли до комірки записується одиничний біт, і розряджають, коли до комірки записується нуль.

Важливим елементом пам’яті цього типу є чутливий підсилювач-компаратор (англ. Sense amp), підключений до кожного з стовпців «прямокутника». Він, реагуючи на слабкий потік електронів на відкритих транзисторах від обкладок конденсаторів, зчитує відразу весь рядок. Саме рядок є мінімальною порцією обміну з динамічною пам’яттю, обмін даними з окремою коміркою неможливий.

На відміну від швидкої, але дорогої статичної пам’яті, повільна, але дешева пам’ять DRAM виготовляється на основі конденсаторів, які швидко втрачають заряд, тому інформацію доводиться оновлювати через певні проміжки часу, щоб уникнути втрат даних. Цей процес називається регенерацією

пам’яті. Він реалізується спеціальним контролером, встановленим на материнській платі або ж на кристалі центрального процесора. Протягом часу, званого кроком регенерації, в DRAM перезаписується цілий ряд комірок, і через 8-64 мс оновлюються всі рядки пам’яті [45].

В класичному варіанті регенерація суттєво гальмує роботу системи, оскільки в цей час обмін даними з пам’яттю неможливий. Регенерація, заснована на звичайному переборі рядків, в сучасних типах DRAM не застосовується. Існує кілька більш економічних варіантів цього процесу – розширений, пакетний, розподілений; найбільш економічною є прихована (тіньова) регенерація.

Серед нових технологій регенерації – PASR (англ. Partial Array Self Refresh), застосовувана компанією Samsung в чіпах пам’яті SDRAM з низьким рівнем енергоспоживання. Регенерація комірок виконується тільки в період очікування в тих банках пам’яті, в яких є дані [46].

Паралельно з цією технологією реалізується метод TCSR (англ. Temperature Compensated Self Refresh), який призначений для регулювання швидкості процесу регенерації в залежності від робочої температури [46].

Основними характеристиками DRAM є робоча частота і таймінги. Основними таймінгами DRAM є: затримка між подачею номера рядка і номера стовпчика, звана часом повного доступу (англ. RAS to CAS delay), затримка між подачею номера стовпчика і отриманням вмісту комірки, звана часом робочого циклу (англ. CAS delay), затримка між читанням останньої клітинки і подачею номера нового рядка (англ. RAS precharge). Таймінги вимірюються у наносекундах або тактах. Чим менша величина цих таймінгів, тим швидше працює оперативна пам’ять [46].

Розробниками створювалися різні типи DRAM. Вони мали різні характеристики та використовували різні технічні рішення. Розглянемо основні типи [47]:

1. Сторінкова пам’ять (англ. Page mode DRAM, PM DRAM) була одним з перших типів комп’ютерної оперативної пам’яті. Вона випускалася на початку 1990-х років, але з ростом продуктивності процесорів і ресурсоємності додатків виникла потреба нарощувати не лише обсяг пам’яті, але і швидкість її роботи.

2. Швидка сторінкова пам’ять (англ. Fast page mode DRAM, FPM DRAM) з’явилася у 1995 році. Збільшення швидкості роботи досягалося шляхом підвищеного навантаження на апаратну частину пам’яті. Даний тип застосовувався для комп’ютерів з процесорами Intel 80486 або аналогічних процесорів інших фірм. Пам’ять могла працювати на частотах 25 і 33 МГц з часом повного доступу 70 і 60 нс і з часом робочого циклу 40 і 35 нс відповідно.

3. EDO DRAM – пам’ять з вдосконаленим виходом. Для процесорів Intel Pentium пам’ять FPM DRAM виявилася неефективною. Тому було створено пам’ять з вдосконаленим виходом (англ. Extended data out DRAM, EDO DRAM). Вона з’явилася на ринку в 1996 році для процесорів Intel Pentium і вище. Продуктивність EDO DRAM виявилася на 10-15% вище в порівнянні з

FPM DRAM. Робоча частота пам'яті – 40 і 50 МГц, час повного доступу – 60 і 50 нс, час робочого циклу – 25 і 20 нс відповідно. Ця пам'ять містить буфер, або ж регістр-засувку (англ. Data latch) вихідних даних, що забезпечує конвеєризацію роботи при читанні.

4. SDR SDRAM – синхронна DRAM. З випуском нових процесорів і збільшенням частоти системної шини, стабільність роботи EDO DRAM стала падати. Їй на зміну прийшла синхронна пам'ять — Single Data Rate Synchronous Dynamic Random Access Memory (SDR SDRAM). Особливостями цього типу пам'яті були використання тактового генератора для синхронізації всіх сигналів і використання конвеєрної обробки даних. Також пам'ять надійно працювала на системній шині 100 МГц і вище.

Якщо для FPM і EDO пам'яті вказується час читання першої комірки у ланцюзі (ланцюг – кілька послідовних комірок), то для SDRAM вказується час зчитування наступних комірок. На зчитування першої комірки йде 60-70 нс незалежно від типу пам'яті, а час читання наступних залежить від типу. Робочі частоти SDR SDRAM становлять 66, 100 або 133 МГц, час повного доступу – 40 і 30 нс, а час робочого циклу – 10 і 7,5 нс.

5. Enhanced SDRAM (ESDRAM). Для прискорення роботи DRAM за рахунок подолання затримок синхронізації пам'яті з системною шиною, було вирішено вбудувати невелику кількість SRAM в чіп, тобто створити на чіпі кеш. ESDRAM – це, по суті, SDRAM з невеликою кількістю SRAM, яка дозволяє досягати робочої частоти до 200 МГц. SRAM-кеш призначений для зберігання і вибірки найчастіше використовуваних даних, що призводить до зменшення часу доступу до даних повільної DRAM.

6. Пакетна EDO RAM (англ. Burst extended data output DRAM, BEDO DRAM). Це дешева альтернатива пам'яті типу SDRAM. ЇЇ особливістю була технологія поблокового читання даних за один такт, що дозволило працювати швидше, ніж SDRAM. Однак неможливість працювати на частоті системної шини більше 66 МГц не дозволила даному типу пам'яті стати поширеним.

7. Video RAM. Це спеціальний тип оперативної пам'яті, розроблений на основі SDRAM для використання у відеоплатах. Він дозволяв забезпечити безперервний потік даних в процесі оновлення зображення для реалізації зображень високої якості. Її продуктивність на 25% вище, ніж у оригінальної пам'яті типу SDRAM.

8. DDR SDRAM (англ. Double data rate SDRAM, DDR SDRAM або SDRAM II) – пам'ять з подвоєною швидкістю передачі даних порівняно з SDRAM. Вона виконана за відкритим стандартом SLDRAM, який використовує обидва фронти перепаду тактового сигналу, чим, власне і досягається подвоєння. Спочатку пам'ять такого типу застосовувалася у відеоплатах, але пізніше з'явилася підтримка DDR SDRAM з боку чипсетів. DDR SDRAM працює на частотах в 100, 133, 166 і 200 МГц, її час повного доступу – 30 і 22,5 нс, а час робочого циклу – 5, 3,75, 3 і 2,5 нс. Оскільки її частота синхронізації лежить в межах від 100 до 200 МГц, а дані передаються по 2 біти на один синхроімпульс, то ефективна частота передачі даних лежить в межах від 200 до

400 МГц. Такі модулі пам’яті позначаються DDR200, DDR266, DDR333, DDR400.

9. Direct RDRAM або Direct Rambus DRAM використовує, як і DDR SDRAM, обидва перепади тактового імпульсу. Це розробка компанії Rambus, що має більш високу у порівняння з DDR SDRAM швидкодію, але є закритою та включає ряд особливих рішень. Висока вартість RDRAM привела до того, що виробники комп’ютерів віддають перевагу більш дешевій DDR SDRAM. Робочі частоти пам’яті – 400, 600 і 800 МГц, час повного доступу - до 30 нс, час робочого циклу – до 2,5 нс.

10. DDR2 SDRAM – розвиток технології DDR SDRAM. За рахунок технічних змін показує вищу швидкодію. Пам’ять може працювати з тактовою частотою шини 200, 266, 333, 337, 400, 533, 575 і 600 МГц. При цьому ефективна частота передачі даних відповідно буде 400, 533, 667, 675, 800, 1066, 1150 і 1200 МГц. Час повного доступу – 25, 11,25, 9, 7,5 нс і менше. Час робочого циклу – від 5 до 1,67 нс [48].

11. DDR3 SDRAM – тип пам’яті, заснований на DDR2 SDRAM з удвічі збільшеною частотою передачі даних. Відрізняється зниженим енергоспоживанням в порівнянні з попередниками. Частота смуги пропускання лежить в межах від 800 до 2400 МГц (рекорд частоти – більше 3000 МГц) [49].

12. DDR4 SDRAM – четверте покоління пам’яті, DDR4 SDRAM що є еволюційним розвитком попередніх поколінь DDR SDRAM. Відрізняється підвищеними частотними характеристиками і зниженою напругою живлення. Спочатку стандарт DDR4 визначав ефективні частоти передачі даних від 1600 до 2400 МГц. Пропускна здатність DDR4 в перспективі може досягати 25,6 ГБ/с за умови підвищення максимальної ефективної частоти до 3200 МГц [50].

13. DDR5 SDRAM – п’яте покоління пам’яті, випущене у 2020 році. Порівняно з DDR4 зменшене енергоспоживання до 1,1 В, пропускна здатність підвищена до 120 ГБ/с. Крім того, усі мікросхеми DDR5 мають вбудований код корекції помилок, який виявляє та виправляє помилки перед надсиланням даних до процесора [51].

9.2 Енергонезалежна пам’ять

Видів енергонезалежної пам’яті багато. Але їх відразу можна розділити за призначенням:

1. Пам’ять, необхідна для забезпечення взаємодії із апаратним забезпеченням обчислювальної системи, що містить базову систему вводу-виводу (BIOS) та налагодження.

2. Пам’ять, призначена для довготривалого зберігання даних, програм, операційної системи.

На сьогодні пам’ять для довготривалого зберігання великих обсягів даних і програм також можна розділити на внутрішню (ту що постійно розміщена всередині корпусу комп’ютера) та зовнішню, або змінну.

9.2.1 Пам’ять для взаємодії із апаратним забезпеченням обчислювальної системи

Розглянемо перший тип. Відразу зробимо уточнення: енергонезалежність пам’яті може бути забезпечена на рівні використаної технології, і тоді мова йде про дійсно енергонезалежні елементи пам’яті, або ж енергонезалежність може бути забезпечена схемотехнічно. В останньому випадку мова йде про звичайний енергозалежний елемент із автономною схемою живлення від батарейки чи акумулятора. Таке рішення використовується для CMOS-пам’яті (англ. complementary metal-oxide-semiconductor, комплементарна структура метал-оксид-напівпровідник, КМОП). У CMOS-пам’яті комп’ютера зберігаються налагодження, необхідні для роботи програмного забезпечення BIOS, а також дані календаря та годинника реального часу. Якщо мова йде про інші пристрої, наприклад, промислові контролери чи програмовані реле, то тут CMOS-пам’ять може використовуватись і як пам’ять довготривалого зберігання операційної системи та прикладних програм, хоча зараз більш масово використовується інша, Flash-технологія.

Щодо енергонезалежних елементів пам’яті. Вони використовуються для так званої пам’яті ROM (англ. Read-Only Memory) – пам’яті з доступом тільки по читанню. BIOS це також ROM. Також використовують назву ПЗП – постійний запам’ятовуючий пристрій.

Пам’ять ROM окрім BIOS використовується також для зберігання програмного забезпечення принтерів, моніторів, модемів, графічних, звукових та мережевих адаптерів та для інших потреб, де необхідне енергонезалежне збереження.

Відомі типи мікросхем ROM:

1. ROM – клас напівпровідникових запам’ятовуючих пристроїв, де дані та програми заносяться у мікросхему під час виготовлення і є незмінними [52]. Зараз майже не використовуються з причини негнучкості технології.

2. PROM (англ. Programmable Read-Only Memory) – клас напівпровідникових запам’ятовуючих пристроїв, де для занесення бітів інформації використовують принцип перепалювання перемичок за допомогою спеціального пристрою – програматора. Пам’ять представляла собою двовимірний масив провідників (рядків і стовпців) на перетині яких створювалася спеціальна перемичка з металу (наприклад, ніхрому або титаново-вольфрамового сплаву) або аморфного кремнію. Програмування полягало в пропущенні через відповідну перемичку струму, який змушував її розірватися – розплавитися і випаруватися [52]. Відновлення розплавлених перемичок неможливо.

Класична технологія PROM виявилася ненадійною. Металеві перемички при програмуванні утворювали краплі і пари металу, які осідали назад на кристал в найнесподіваніших місцях з відповідними неприємними наслідками. Полікремнієві перемички мали здатність до самовідновлення за рахунок міграції атомів [53]. З цієї причини мікросхеми після програмування витримувались тривалий час при високій температурі з метою виявлення потенційних дефектів.

Звичайно, такі мікросхеми записувалися один раз. Модернізація програмного забезпечення, наприклад, BIOS потребувала заміни мікросхеми новою перезаписаною.

3. EPROM (англ. Erasable Programmable Read Only Memory) – напівпровідниковий запам'ятовуючий пристрій, постійна пам'ять, для запису даних в який використовується програматор і який допускає перезапис [54].

Є матрицею транзисторів, індивідуально запрограмованих за допомогою програматора, котрий подає більш високу напругу, ніж зазвичай використовується в цифрових схемах. Дані на EPROM можна стерти ультрафіолетовим опроміненням від ртутного джерела світла. EPROM має прозоре вікно з кварцового скла у верхній частині корпусу, через яке проводиться опромінення ультрафіолетовим світлом під час стирання [54].

Запрограмована пам'ять EPROM зберігає свої дані на десять-двадцять років, і може бути прочитана необмежену кількість разів. Вікно стирання повинно бути закрито непрозорою плівкою для запобігання випадкового стирання сонячним світлом. Старі BIOS комп'ютерів, наприклад на базі Intel 80486, часто були зроблені з використанням EPROM, а вікна стирання були закриті етикеткою, що містить назву виробника BIOS, версію BIOS і повідомлення про авторські права [54].

EPROM мають обмежену кількість циклів стирання, як правило, декілька тисяч, тому це обмеження не принципове.

На сьогодні використання EPROM досить обмежене через те, що процес їх програмування потребує більше часу у порівнянні з іншими технологіями ROM.

4. EEPROM (англ. Electrically Erasable Programmable Read-Only Memory) – перепрограмовуване ПЗП, з електричним стиранням даних. Пам'ять такого типу може стиратися і заповнюватися даними до мільйона разів. Тут використаний принцип зміни і реєстрації електричного заряду в ізольованій області (кишені) напівпровідникової CMOS-структури. Перші зразки EEPROM вимагали для запису та стирання використання програматора, причому запис у комірку пам'яті відбувався методом «гарячої інжекції», тобто «впорскуванням» заряду до кишені шляхом подання підвищеної (у 2-3 рази вищої порівняно з живленням) напруги на записувану комірку, що дозволяло електронам проходити через p-n-перехід. Стирання проводилось методом «тунелювання», тобто з поданням підвищеної напруги (до 1,5 від напруги живлення) заряди за рахунок тунельного ефекту залишали ізольовану кишеню. Зараз EEPROM використовують в основному тунельний ефект і у більшості не потребують наявності зовнішнього програматора [55].

5. Flash-технологія є розвитком технології EEPROM. ЇЇ відмінність від базової полягає у більшій швидкодії, оскільки сучасні EEPROM здійснюють доступ на читання та запис до кожної комірки пам'яті окремо, а Flash працює відразу зі сторінковими блоками по 2 КБ, 4 КБ і вище, в залежності від самого Flash-модулю. Звичайно, швидкість доступу до окремої комірки тут уповільнено, оскільки дані у цьому випадку будуть зчитуватися зі сторінки, яка

повинна бути перед цим переміщена у спеціальний кеш модуля пам'яті [56]. Але, оскільки флеш-пам'ять використовується переважно з пристроями поблокового читання-запису, це несуттєво. На сьогодні технологія Flash не повністю витіснила класичну EEPROM, оскільки іноді раціональніше використовувати побайтовий чи послівний доступ до пам'яті. Наприклад, це виправдано у однокристальних мікро-ЕОМ інтелектуальних побутових, керуючих чи вимірювальних приладах для запису та зберігання програм. Саме тому серії однокристальних мікроконтролерів, таких як x51, PIC, Atmega, промислові контролери ICP DAS, Holit разом з флеш-пам'яттю для енергонезалежного зберігання даних використовують класичну EEPROM у тому числі для зберігання програмного коду.

9.2.2 Жорсткі магнітні диски

На сьогодні для довгочасного зберігання інформації в якості внутрішніх пристроїв найбільш поширені накопичувачі на жорстких магнітних дисках, HDD (англ. hard (magnetic) disk drive, HDD, HMDD), або так звані «вінчестери».

Жорсткий диск складається з п'яти основних частин [57]:

1. Інтегральна схема, яка синхронізує роботу диска з комп'ютером і керує всіма процесами.

2. Електромотор (привод диска), який змушує обертатися диск зі швидкістю 5400 або 7200 об/хв, а інтегральна схема підтримує швидкість обертання постійною.

3. Коромисло, яке може як записувати, так і зчитувати інформацію. Кінець коромисла зазвичай розділений, для того щоб можна було працювати відразу з декількома дисками. Голівка коромисла не стикається з дисками, є зазор між поверхнею диска і головкою, розмір якого приблизно в п'ять тисяч разів менше товщини людської волосини. Несправність коромисла або ж збої у його роботі викликані, наприклад, перепадами напруги, можуть призвести до контакту голівки коромисла з поверхнею диска. Це викликає механічні ушкодження магнітної поверхні та можливість втрати інформації. Крім того, відбиті фрагменти диску під час подальшої роботи можуть літати всередині камери, де розміщено диск та коромисло, знову стикатися з дисковою поверхнею і спричинити додаткові ушкодження. Щоб цього не сталося, камера покрита спеціальним вловлюючим та затримуючим матеріалом.

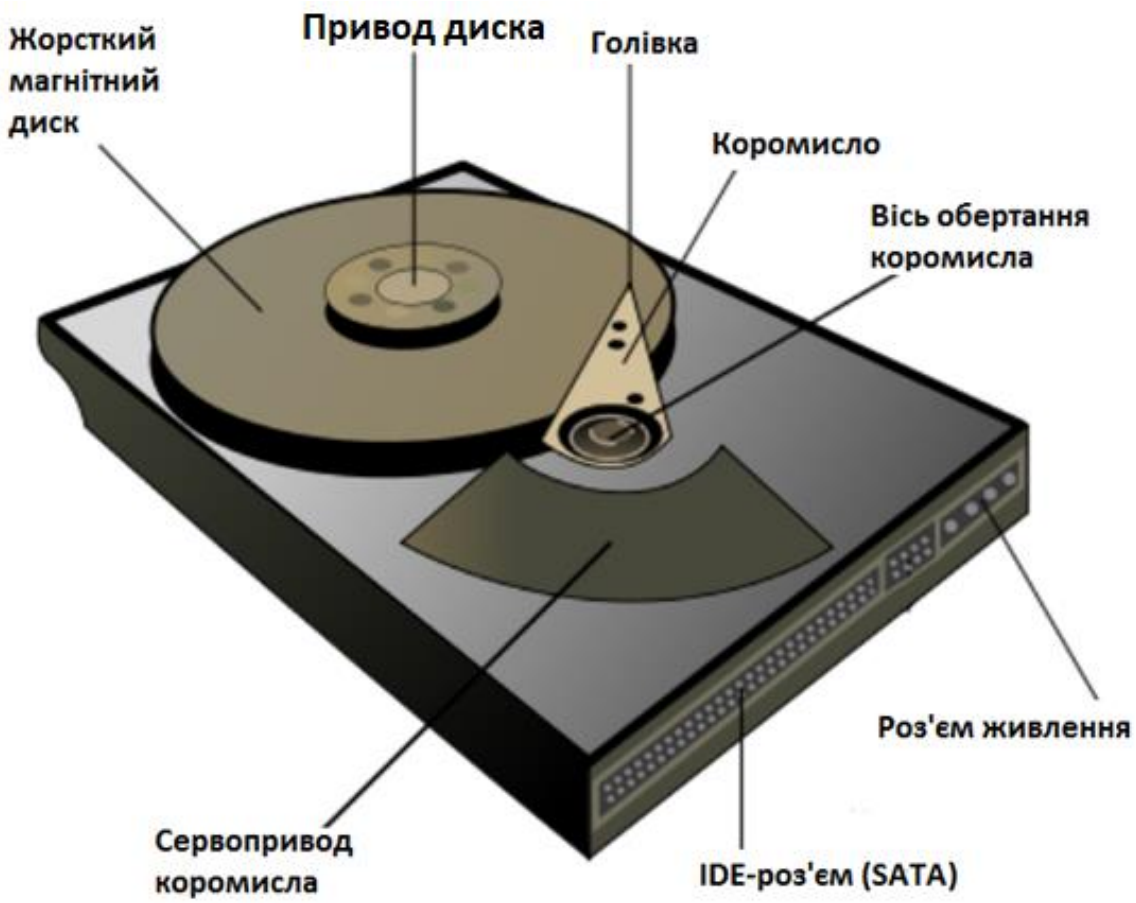

Рисунок 9.3 – Будова HMDD

Під час зчитування інформації коромисло зміщується над поверхнею диску до 60 разів за секунду. Зміщення здійснює електромагнітна система, сервопривод коромисла.

4. Сам жорсткий диск, куди записується і звідки зчитується інформація, пластин диску може бути кілька (зазвичай 1-2).

5. Корпус, в який встановлюються всі інші компоненти. Матеріали застосовуються такі: майже весь корпус виконаний з пластмаси, але верхня кришка завжди металева.

Корпус в зібраному вигляді нерідко називають гермозоною. Оскільки при таких високих швидкостях обертання диска навіть порошинка, яка потрапила всередину, може призвести до ушкоджень магнітної поверхні, гермозона заповнена очищеним та осушеним повітрям або нейтральний газом, наприклад азотом.

Коли жорсткий диск включається в роботу – це означає або на нього здійснюється запис, або з нього йде читання інформації, або з нього завантажується операційна система, шпиндель починає набирати обертів, а оскільки жорсткі диски закріплені на самому шпинделі, відповідно вони разом з ним теж починають обертатися. Поки оберти дисків не досягнули швидкості, щоб між головкою коромисла і диском утворилася повітряна подушка, коромисло, щоб уникнути пошкоджень, знаходиться в спеціальній паркувальній зоні. Як тільки оберти досягають потрібного рівня, сервопривод приводить у рух

коромисло, яке позиціонується в те місце, куди потрібно записати або звідки прочитати інформацію.

Дані на HMDD зберігаються у вузьких доріжках на поверхні диска, що мають вигляд концентричних кіл із центром, що збігається з віссю обертання дисків. При виробництві на диск наноситься більше 200 тисяч таких доріжок. Диск може мати двостороннє магнітне покриття, тоді доріжки наносяться на обох поверхнях диску. Вважаючи, що дисків крім того може бути декілька, утворюється група доріжок однакового радіусу. Ця група називається циліндром. Циліндри мають свої номери, які використовуються під час низькорівневого керування введенням та виведенням [57].

Кожна з доріжок під час низькорівневого форматування розділена на сектори по 512 байт, хоча зараз виробники масово переходять до 4096 байт [57]. При цьому кількість секторів на доріжці залежить від того, внутрішня це доріжка чи зовнішня, на зовнішній можливо розмістити секторів більше. Кількість секторів на доріжку знаходиться у межах від 16 до 1600, але об'єктивно цю інформацію отримати складно, оскільки існуюче сервісне та системне програмне забезпечення повертає ті дані, які надає операційна система, драйвери пристроїв чи контролер диска. Наскільки вони відповідають дійсності – невідомо, але це і не потрібно користувачеві чи програмісту, сфера відповідальності яких обмежена наданою моделлю фізичної організації даних.

Карти доріжок і секторів дозволяють визначити, куди записати або де зчитати інформацію. Вся інформація про сектори та доріжки знаходиться в пам'яті інтегральної мікросхеми, яка, на відміну від інших компонентів жорсткого диска, розміщена не всередині корпусу, а зовні і зазвичай знизу.

Принцип занесення інформації на жорсткий диск наступний. Головка коромисла намагнічує мікроскопічну область в феромагнітному шарі дискової поверхні, встановлюючи магнітний момент такої комірки в один зі станів: 0 або 1. Таким чином, будь-яка інформація, записана на жорсткому диску, фактично, є певною послідовністю бітів. При цьому кожен квадратний сантиметр поверхні жорсткого диска включає в себе кілька десятків мільярдів бітів.

9.2.3 Твердотільні внутрішні накопичувачі

Цей тип накопичувачів використовує так звану SSD технологію.

SSD (solid state drive, накопичувач на твердотільній пам'яті, твердотільний накопичувач) – накопичувач інформації, заснований на кристалах енергонезалежної пам'яті [58]. Це відносно новий вид носіїв інформації, який є розвитком вже розглянутої раніше технології Flash.

SSD містить такі ж інтерфейси введення-виведення як і сучасні жорсткі диски, що дозволяє їм витісняти з ринку HMDD. У SSD не використовуються рухомі частини та елементи як в електромеханічних пристроях (жорсткі диски, zip-дискети), що виключає ймовірність зносу механічним шляхом [58].

На сьогоднішній день швидкість запису та читання SSD накопичувачів висока – на рівні з HMDD, котрі використовують інтерфейс SATA3 зі швидкістю до 6 Гб\сек. Існують також гібридні версії SSD і HDD накопичувачів [58].

Твердотільний накопичувач складається з самих мікросхем Flash, керуючого контролера, мікросхеми енергозалежної кеш-пам'яті і плати, на якій все це розміщено. Іноді в SSD накопичувачах використовується невелика батарея, щоб у випадку відключення живлення, всі дані з кешу можна було б переписати до енегронезалежної пам'яті.

9.2.4 USB-Flash-накопичувачі та карти

На сьогодні найбільш широко вживана зовнішня змінна пам'ять – зовнішні USB-Flash-накопичувачі із інтерфейсом USB, USB 2.0, USB 3.0, USB 3.1, останнім часом використовується також інтерфейс USB 4.0 [59]. Але існують і інші різновиди накопичувачів, що використовуються не лише у персональних комп'ютерах. Всі вони використовують Flash-технологію і відрізняються одне від одного в основному конструктивом. Розглянемо деякі з них [60]:

1. Compact Flash (CF) – це знімний носій з паралельним інтерфейсом. Мають 50-контактний контактний роз'єм, швидкість обміну до 33 Мб/с і обсяг до 130 Гб. Популярні у використанні цифрових фотокамер. Підтримують 2 режиму роботи:

- PCMCIA-стандарту для карт введення-виведення;
- IDE (ATA)-інтерфейс для використання як жорсткий диск.

Рисунок 9.4 – Compact Flash (CF)

2. Memory Stick Duo – ці карти флеш-пам'яті мають послідовний 10-ти контактний інтерфейс. Розроблено фірмою Sony і використовуються в її ж продукції.

Рисунок 8.5 – Memory Stick Duo

3. miniSD – сучасна карта пам'яті, призначена для портативних пристроїв – мобільні телефони, навігатори, MP3-плеєри, електронні книги і подібні пристрої, які мають невеликий розмір.

Рисунок 8.6 – miniSD

4. MultiMedia (MMC) – змінний флеш-накопичувач, складається з пластикового корпусу і семи контактів інтерфейсу, серед яких використовуються лише шість, може працювати в SPI режимі і MMC (протоколи передачі даних):

- MMC – команди можуть передаватися одночасно з даними на частотах до 20МГц;
- SPI – є частиною протоколу, але більш дешевий варіант.

Рисунок 8.7 – MultiMedia (MMC)

5. SD – те ж саме, що MMC, але є відмінності: на дві шини передачі даних більше; є захист від перезапису.

Рисунок 8.8 – SD

6. SmartMedia – збирається без пайки на гнучкій пластиковій основі і містить у собі тільки контакти і мікросхему пам'яті, невеликий обсяг до 256 МБ. Напруги живлення дві: 5В ключ-куточок знаходиться зліва, 3В – справа.

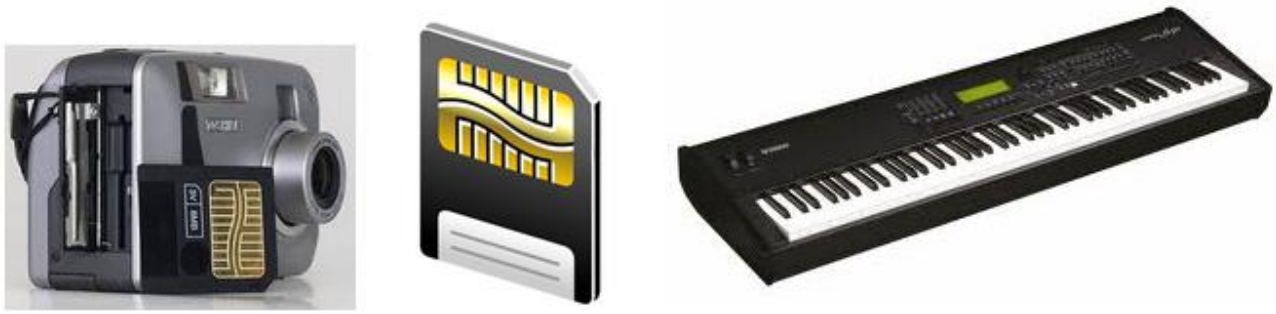

Рисунок 8.9 – SmartMedia

7. xD-Picture — використовується в сучасних фотоапаратах Olympus/Fujifilm. Швидкість запису даних — 3 Мб/с, читання — 5 Мб/с, і обсяг від 256 МБ до 2 ГБ.

Рисунок 8.10 – xD-Picture

На швидкісні характеристики карт існує класифікація, так званий SD Speed Class, який у вигляді маркування наноситься на корпус карти:

SD Class 2 - (швидкість запису не менше 2 МБ / с) - 13х;
SD Class 4 - (швидкість запису не менше 4 МБ / с) - 26х;
SD Class 6 - (швидкість запису не менше 6 МБ / с) - 40х;

SD Class 10 - (швидкість запису не менше 10 МБ / с) - 66х;
SD Class 16 - (швидкість запису не менше 16 МБ / с) - 106х.

9.2.5 Зовнішні HDMD-накопичувачі та SSD-накопичувачі

Як змінні носії для невеликих обсягів даних і USB-накопичувачі і Flash-карти зі своїми функціями справляються. Але існує потреба у зберіганні та забезпеченні мобільності великих обсягів даних – до 1…2 ТБ і більше. У цьому випадку використовують так звані зовнішні HDMD та SSD-накопичувачі з інтерфейсом переважно USB 3.0 (більш ранні моделі використовували USB 2.0) та швидкістю передачі даних до 5 ГБ/с. Вони відрізняються від внутрішніх конструктивом – більшою механічною міцністю та стійкістю до ударів, вібрацій, пилу та вологості за рахунок спеціальних рішень при виготовлені корпусу [61].

Рисунок 8.11– Зовнішній HDMD

9.2.6 Диски DVD

На сьогодні ця технологія, що прийшла на заміну CD-дискам застаріває, але все ще використовується. Диски DVD використовують не лише для зберігання і перегляду відео- та мультимедіа-файлів, але і для продажу і розповсюдження програмного забезпечення, драйверів пристроїв, просто довготривалого зберігання інформації. Розглянемо принцип роботи DVD-дисків і пристроїв.

Схема роботи пристрою читання – запису DVD-дисків наступна:

1. Лазерний діод випромінює малопотужний пучок світла довжиною 650 нм, який, проходячи через направляючу призму і роздільник променю,

потрапляє на відбиваюче дзеркало. Під час запису потужність лазерного променю значно зростає, при стиранні даних також, але менше.

2. Під керуванням процесора привода, каретка з відбиваючим дзеркалом переміщається до потрібної доріжки, як і у HDD-диска.

3. Лазерний промінь відбивається від диска, потрапляє на дзеркало, потім на роздільник променю і далі на направляючу призму.

4. З призми промінь потрапляє в фотодатчик, фотодатчик посилає сигнали у вбудований в привод DVD-дисків мікропроцесор, де дані обробляються і передаються по шлейфу на материнську плату.

Основою запису і зберігання даних на дисках DVD-R і DVD-RW (одноразово записуваних та дисків зі стиранням) є технологія зміни фазового стану речовини. При записи і зчитуванні інформації використовується відмінність відбивної здатності поверхні в залежності від того, чи знаходиться вона в кристалічному або аморфному стані.

При зчитуванні інформації з диска вимірюється різниця між темними аморфними і яскравими прозорими западинами амфотерного матеріалу, що містить біти даних. Цю технологію цілком можна назвати оптичною – для читання і запису достатній всього лише лазер.

Існують як односторонні, так і двосторонні DVD-диски. Останні містять напівпрозорий верхній шар з доріжками, такий, що дозволяє променю отримувати доступ і до нижнього шару. Ємність жорсткого диску залежить від конкретного типу і може варіюватися від 4,7 до 17 ГБ, що разом із необхідністю мати пристрій читання та запису значно знижує його конкурентоспроможність порівняно з іншими технологіями для зовнішніх носіїв.

Западини на інформаційному шару диска утворюють єдину спіральну доріжку (в кожному шарі) з відстанню 0,74 мікрона між витками, що відповідає щільності доріжок 1351 виток на міліметр, або 34324 витка на дюйм. В цілому це становить 49324 витка, а загальна довжина доріжки досягає 11,8 км. Доріжка розбита на сектори, кожен з яких містить 2048 байт даних. Диск розділений на чотири основні області.

Область фіксування (посадки) диска. Являє собою центральну частину компакт-диска з отвором діаметром 15 мм для валу програвача. Ця область не містить якої-небудь інформації або даних.

Початкова область. Починається від внутрішньої частини диску і включає в себе буферні зони, код посилання і зону службових даних, що містить інформацію про диск. Зона службових даних складається з 16 секторів, продубльованих 192 рази, що становить 3072 сектора даних. У цих секторах розташовані дані про диск, зокрема вказані категорія диска і номер версії, розмір і структура диска, максимальна швидкість передачі даних, щільність запису і розподіл зони даних. В цілому початкова область займає до 196 607 (2FFFFh) секторів диска.

Базова структура всіх секторів DVD. Сектори буферної зони початкової області містять тільки символи 00h (шістнадцяткові нулі).

Область даних містить відео-, аудіо- або інші дані і починається з сектора під номером 196608 (30000h). В цілому область даних одношарового одностороннього диска може містити до 2292897 секторів.

Кінцева (або середня) зона. Відзначає завершення області даних. Сектори кінцевої зони містять тільки значення 00h. У тому випадку, якщо диск має два шари запису і записаний в режимі зворотного зчитування, де другий шар починається з зовнішньої сторони диска і зчитується в протилежному по відношенню до першого шару напрямку, ця зона називається середньою.

9.3 Висновки

Викладений у цьому розділі матеріал розглядає принцип роботи і технології побудови пам'яті різного призначення. Існуючі на сьогодні дві базових технології енергозалежної пам'яті: статична й динамічна використовуються у технічних рішеннях оперативної, відео- і кеш-пам'яті в залежності від своїх переваг. Зокрема, динамічна пам'ять широко використовується саме для оперативної пам'яті насамперед через дешевше схемотехнічне рішення. Більше того, технологія динамічної пам'яті розвивається за рахунок удосконалення технологій напівпровідників, мініатюризації, використання швидкісного синхронного інтерфейсу обміну з подвоєною частотою синхронізації.

Енергонезалежна пам'ять, яка використовується для в якості BIOS комп'ютера у своєму розвитку поступово переходить від руйнуючих технологій запису до EERPOM та flash-технологій, які ближчі до рішень для динамічної пам'яті, хоча мають свої властивості. Ці властивості на сьогодні дозволяють далі розвивати однокристальні мікроконтролери, як складову промислових комп'ютерів та IoT.

Щодо енергонезалежної пам'яті тривалого зберігання, то тут, поряд із загальним підвищенням продуктивності HDMD розвивається SSD-технологія, яка є спеціалізованою гілкою flash-технології. З одного боку це призводить до підвищення швидкодії такого виду пам'яті, з іншого – термін служби SSD-накопичувача більш обмежений через обмеженість кількості перезаписів. Тому на сьогодні доцільно використовувати SSD невеликого обсягу для операційної системи й, можливо, для виконуваних програм. Для даних все ж краще використовувати HDMD.

Зовнішні накопичувачі енергонезалежної пам'яті тривалого зберігання на нинішній час можуть бути реалізовані у вигляді переносного робочого місця з операційною системою чи великого диску для зберігання програм і даних. Це зовнішні HDMD та SSD-накопичувачі. Або компактних сховищ малих обсягів – USB-flash-накопичувачів. В обох випадках персональні комп'ютери та цілий ряд промислових контролерів надають уніфікований інтерфейс взаємодії з такими носіями. З іншого боку, у ніші flash-карт спостерігається досить значна різноманітність інтерфейсів, форм-факторів і форматів, що ускладнює користування.

І наостанок. Хоча технологія оптичних DVD-дисків в Україні вважається застарілою, вона все ще використовується для зберігання ліцензій деяких

комерційних і промислових програм, для ліцензованого програмного забезпечення, для довготривалого зберігання даних, або для даних, розповсюдження яких обмежено, наприклад, авторським правом. Це помітно, наприклад, в ряді країн Європи, де заборонено отримувати фільми чи програми через торенти чи неліцензовані сайти, але їх можна придбати через роздрібну торгівлю на DVD-дисках.

9.4 Питання для самоперевірки

1. Для чого використовують енергозалежну пам’ять?
2. Для чого використовують енергонезалежну пам’ять?
3. Що таке статична енергозалежна пам’ять?
4. Що таке динамічна енергозалежна пам’ять?
5. Що таке PM DRAM та FPM DRAM?
6. Чим відрізняється EDO DRAM від FPM DRAM з точки зору властивостей і технічного рішення?
7. Чим відрізняється SDR SDRAM від EDO DRAM та FPM DRAM з точки зору властивостей і технічного рішення?
8. У чому особливість ESDRAM у порівнянні з SDR SDRAM?
9. У чому особливість роботи EDO RAM?
10. У чому особливість Video RAM порівняно з SDRAM?
11. У чому особливість всіх технологій DDR SDRAM, включаючи Direct RDRAM, у порівнянні з іншими?
12. Що таке ROM-технологія?
13. Що таке PROM-технологія?
14. Що таке EPROM-технологія?
15. Що таке EEPROM-технологія?
16. Що таке Flash-технологія?
17. Як побудовано HMDD?
18. Що таке SSD-технологія?
19. Які типи USB-Flash-накопичувачів та карти Ви знаєте?
20. Що собою являє технологія зберігання на DVD-дисках?

10 ПАМ'ЯТЬ КОМП'ЮТЕРА НА РІВНІ ОПЕРАЦІЙНОЇ СИСТЕМИ

10.1 Оверлейна модель пам'яті

Ця модель пам'яті широко використовувалась застарілими на сьогодні операційними системами, такими як MS DOS, TR DOS чи Novell DOS. Незважаючи на те, що для універсальних комп'ютерів вони не використовуються, ці операційні системи все ще мають свою нішу застосування серед деяких моделей та серій промислових контролерів. Крім того, для тих же промислових контролерів, побутових та вимірювальних приладів, на їх основі розроблена спеціалізована операційна система MiniOS. Власне, наведені операційні системи через обмеження продуктивності обладнання, що їх використовує, не надають програмісту універсальної моделі оперативної пам'яті, хоча і підтримують фон-Нейманівську архітектуру. З цих причин прикладні програми займають той же адресний простір, що і операційна система, який надається апаратним забезпеченням. Тобто і операційна система, і додаткові драйвери (програми керування обладнанням), і резидентні програми (такі що виконують сервісні функції у фоновому режимі паралельно основній програмі), і власне, основна прикладна програма мають бути розміщені у тому об'ємі оперативної пам'яті, що надається конкретним комп'ютером. Такий режим роботи називається реальним або незахищеним режимом роботи обчислювальної системи [62]. Наведені особливості однозадачних, а насправді, оскільки є фонові процеси, умовно однозадачних, операційних систем призводять до погіршення можливостей перенесення програм між апаратними платформами навіть у рамках однієї операційної системи.

Проблеми незахищеного режиму у згаданих вище ОС вирішуються методом модульного або оверлейного програмування. Метод полягає у наступному. Програма, розмір якої перевищує розмір оперативної пам'яті, розділюється на окремі модулі, що звуться також оверлеями. Є основний або початковий модуль, який завантажується в оперативну пам'ять під час запуску програми. За необхідністю, він може особливими командами до ОС завантажувати чи вивантажувати з пам'яті додаткові модулі таким чином, щоб загальний обсяг завантаженого фрагменту програми в оперативній пам'яті не лише не перевищував обсяг самої пам'яті, але й дозволяв у ній розмістити ще й саму ОС, її необхідні графічні оболонки та резидентне системне програмне забезпечення [63]. Початковий модуль при цьому зветься завантажувачем. У програми може бути більше одного завантажувача, оскільки завантажені оверлеї, у свою чергу, також можуть потребувати завантаження додаткових модулів. Недоліки такого методу очевидні:

1. Розбиття програми на модулі виконує сам програміст. Це вимагає від нього додаткової роботи, не пов'язаної з вирішенням прикладної задачі, та підвищує імовірність помилок.

2. У результаті немає гарантії, що написана програма зможе використовуватись на будь-якому комп'ютері, навіть з цією ж ОС, оскільки можуть відрізнятися як апаратні об'єми оперативної пам'яті, так і доступні об'єми. На

доступні об’єми впливає також і загальний обсяг оперативної пам’яті, раніше зайнятий системним та іншим резидентним програмним забезпеченням.

З вказаних причин, у сучасних операційних системах використовується інший підхід – віртуальна пам’ять.

10.2 Віртуальна пам’ять

Власне, термін «віртуальний» походить від латинського virtualis – можливий, потенційний, такий, що має силу чи ефект. Тут мова не йде про фізичне існування чи неіснування. Римляни використовували його для позначення сукупності всіх чудових якостей, притаманних чоловікам (фізична сила, доблесна поведінка, моральна гідність). Якщо цими умовами вважати функціональну структуру і можливості сучасної ОС, то віртуальна пам’ять – це пам’ять, яка існує в умовах виконання програмного коду операційної системи. Інакше кажучи, ОС надає програмісту деяку модель оперативної пам’яті достатньо великого, умовно «нескінченного» обсягу для написання прикладних програм, позбавляючи його необхідності керування обмеженим ресурсом фізичної оперативної пам’яті. В обчислювальній техніці слід уникати трактовки терміну «віртуальний» як «неіснуючий». Тут віртуальні об’єкти існують, діють і можуть проявляти себе у матеріальному світі об’єктів чи впливати на рішення суб’єктів (людей чи організацій).

На сьогодні можна виділити дві моделі віртуальної пам’яті – сторінкову та сегментну (в деяких джерелах її називають сегментно-сторінковою). Розглянемо кожну з них окремо.

10.2.1 Сторінкова модель віртуальної пам’яті

Модель базується на понятті сторінки – блоку простору віртуальних адрес. Одиниця віртуальної пам’яті називається сторінковий блок. Сторінкові блоки однакові за розміром, який зазвичай складає 4-8 КБ. Розмір кратний розміру сектору магнітного диску. Віртуальні і фізичні адресні простори комп’ютера і ОС також розбиваються на блоки розміром в одну сторінку.

Блок основної пам’яті, рівний за об’ємом сторінці, називають сторінковим кадром або фреймом. Сторінкам віртуальної і фізичної пам’яті присвоєно номери. Під час звертання процесора до деякої комірки пам’яті процесор виставляє її адресу, яка складається з номера віртуальної сторінки і зсуву щодо початку сторінки. Ця віртуальна адреса надходить в схему перетворення адрес для подальшого перетворення її в фізичну, причому перетворенню піддається тільки номер віртуальної сторінки, оскільки зміщення для віртуальної і оперативної пам’яті однакові. Схема перетворення адрес називається **MMU (Memory Management Unit)**. Це апаратний компонент комп’ютера (рисунок 10.1), який виконує:

- перетворення віртуальних адрес у фізичні (адресна трансляція);
- забезпечення захисту пам’яті (перевірка прав доступу);
- обробку сторінкових переривань (page faults);
- підтримку кешу адрес (TLB) для прискорення трансляції.

Якщо MMU виявляє, що сторінка з необхідним номером відсутня в оперативній пам’яті, то формується сигнал сторінкового збою (page fault), який ще називають сторінковим перериванням. У результаті обробки переривання потрібна сторінка завантажується диспетчером пам’яті із зовнішньої пам’яті в основну [64].

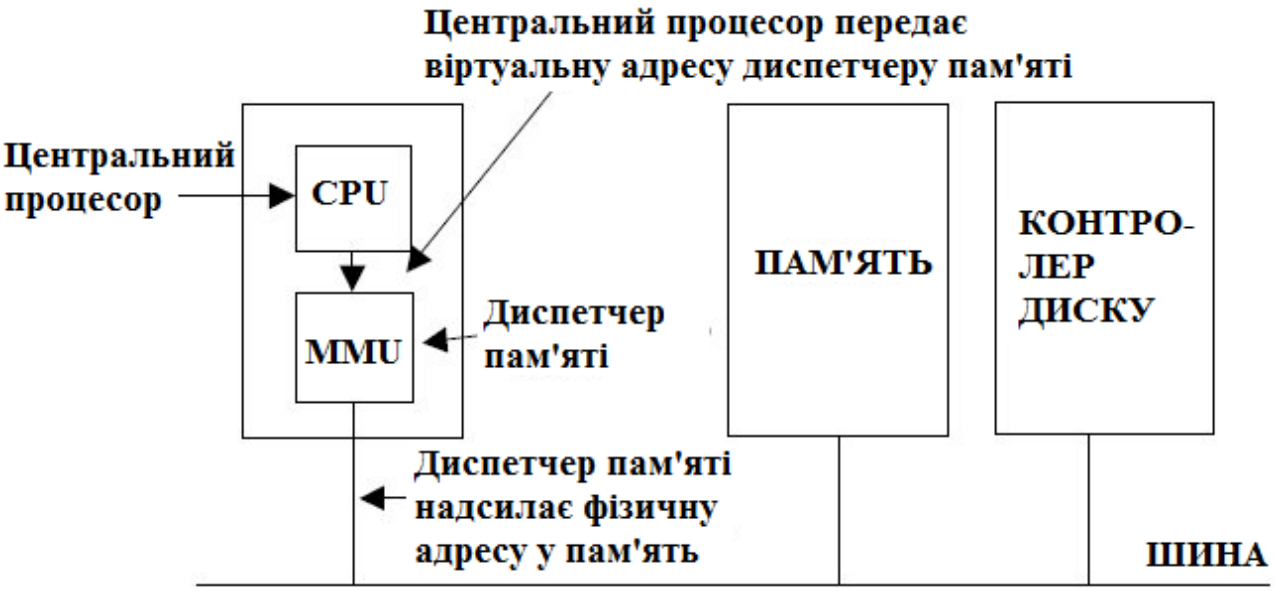

Рисунок 10.1 – Робота диспетчера пам’яті

При створенні процесу операційна система завантажує в оперативну пам’ять кілька його віртуальних сторінок. Копія всього віртуального адресного простору процесу знаходиться на диску у файлі (для Windows) або сегменті (для клонів Unix) підкачки. Суміжні віртуальні сторінки не обов’язково знаходяться у суміжних фізичних сторінках. Для кожного процесу ОС створює таблицю сторінок – інформаційну структуру, яка містить записи про всі віртуальні сторінки процесу (рисунок 10.2).

Запис таблиці (дескриптор сторінки) включає наступну інформацію [65]:

- номер фізичної сторінки (Nф.), в яку завантажена дана віртуальна сторінка;
- ознака присутності P, що встановлюється в одиницю, якщо дана сторінка знаходиться в оперативній пам’яті;
- ознака модифікації сторінки D, який встановлюється в одиницю всякий раз, коли проводиться запис за адресою, що належить до даної сторінки;
- ознака звернення A на сторінку, званий також бітом доступу, який встановлюється в одиницю при кожному зверненні за адресою, що належить до даної сторінки;
- інші керуючі біти, службові, наприклад, для цілей захисту або спільного використання пам’яті на рівні сторінок.

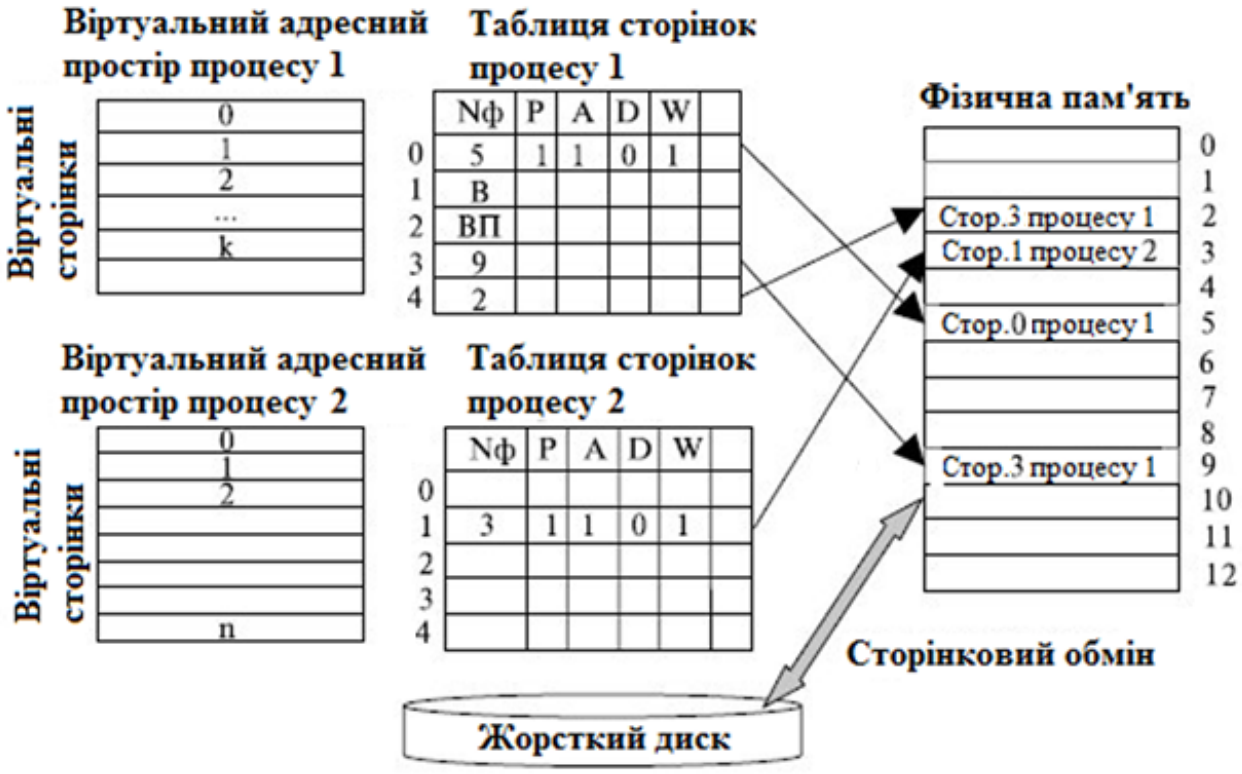

Рисунок 10.2 – Робота диспетчера пам’яті ОС

Перераховані ознаки в більшості моделей процесів встановлюються апаратно схемами процесора при виконанні операцій з пам’яттю. Інформація з таблиці сторінок використовується для вирішення питання про необхідність переміщення тієї чи іншої сторінки між пам’яттю і диском, а також для перетворення віртуальної адреси у фізичну. Самі таблиці сторінок, так само як і описувані ними сторінки, розміщуються в оперативній пам’яті.

Оскільки процес може задіяти великий обсяг віртуальної пам’яті (наприклад, в Windows 2000 він дорівнює 2^{32} = 4 Гбайт), при використанні сторінки об’ємом 4 Кбайт (2^{12}) буде потрібно 220 записів в таблиці сторінок для кожного процесу. Зрозуміло, що виділяти таку кількість оперативної пам’яті під таблиці сторінок недоцільно. Для подолання цієї проблеми більшість схем віртуальної пам’яті зберігає таблиці сторінок в віртуальній пам’яті. Це означає, що самі таблиці сторінок стають об’єктами сторінкової організації. При роботі процесу як мінімум частина його таблиці сторінок повинна розташовуватися в фізичній оперативній пам’яті, в тому числі запис про сторінку, що виконується в даний момент. Адреса таблиці сторінок включається в контекст процесу. При активізації чергового процесу операційна система завантажує адресу його таблиці сторінок в спеціальний регістр.

При кожному зверненні до пам’яті виконується пошук номера віртуальної сторінки, що містить потрібну адресу, потім за цим номером визначається потрібний елемент таблиці сторінок і з нього витягується інформація, що описує сторінку. Далі аналізується ознака присутності, і якщо дана віртуальна сторінка знаходиться в оперативній пам’яті, то виконується перетворення

віртуальної адреси у фізичну. Якщо ж потрібна віртуальна сторінка в даний момент вивантажено на диск, то відбувається сторінкове переривання.

Виконуючий процес переводиться в стан очікування, активізуючи процес з черги процесів, що знаходяться в стані готовності. Паралельно програма обробки сторінкового переривання знаходить на диску необхідну віртуальну сторінку і намагається її завантажити в оперативну пам'ять. Якщо в пам'яті є вільна фізична сторінка, то завантаження виконується негайно. Якщо ж вільних сторінок немає, то на підставі прийнятої в даній системі стратегії заміщення сторінок вирішується питання про те, яку сторінку слід вивантажити з оперативної пам'яті.

Рисунок 10.3 – Перетворення віртуальної адреси

Після того, як обрано сторінку, яка повинна покинути оперативну пам'ять, її біт присутності обнуляється, потім аналізується її ознака модифікації. Якщо заміщувана сторінка за час останнього використання в оперативній пам'яті була модифікована, то її нова версія повинна бути переписана на диск. Якщо ні, то на диску вже є її актуальна копія, тому ніякого запису на диск не проводиться. І нарешті, фізична сторінка оголошується вільною. З міркувань безпеки в деяких системах сторінка, що звільняється, обнуляється, щоб неможливо було використовувати вміст вивантаженої сторінки.

Віртуальна адреса при сторінковому розподілі може бути представлена у вигляді пари (P, Sv), де P – номер віртуальної сторінки процесу (нумерація сторінок починається з 0), а Sv - зміщення в межах віртуальної сторінки (рисунок 10.3). Фізична адреса також може бути представлена у вигляді пари (N, Sf), де N – номер фізичної сторінки, а Sf – зміщення в межах фізичної сторінки.

Завдання підсистеми віртуальної пам’яті полягає в відображенні пари значень (P, Sv) в пару (N, Sf).

Щоб зрозуміти механізм реалізації цього відображення, слід зупинитися на двох базових властивостях сторінкової організації. Як уже зазначалося, обсяг сторінки, як віртуальної, так і фізичної, вибирається рівним ступеню двійки 2^k (k = 8 і більше). Звідси випливає, що зміщення Sv і Sf може бути отримано відділенням k молодших розділів в двійковій запису віртуального і, відповідно, фізичної адреси сторінки. При цьому старші розділи адреси, що залишилися, являють собою двійковий запис номера віртуальної і, відповідно, фізичної сторінки. Доповнивши ці номери до нулів, можна отримати початкову адресу віртуальної і фізичної сторінок.

Підвантаження сторінки пам’яті організовується внаслідок сторінкового переривання. У той момент, коли програмний код сторінки оперативної пам’яті виконується і потрібне завантаження наступної сторінки, виникає сторінкове переривання [66].

Таблиця сторінки містить початкову адресу сторінки оперативної пам’яті, яка складається з адрес команд програмного коду, реалізованого на цій сторінці. Для прискорення роботи зі сторінкою в оперативній пам’яті або в спеціальному контролері або засобами самого процесора організовується буфер швидкого перетворення адреси, який називається асоціативною пам’яттю, всередині якої відбувається складання віртуальних адрес з адресами сторінок.

В сучасних обчислювальних системах підтримуються багаторівневі таблиці сторінок. Принцип їх роботи в тому, що може підтримуватися деяка сукупність таблиць сторінок, які самі розміщені в окремих сторінках. В оперативній пам’яті постійно знаходяться таблиці сторінок вищого рівня, які використовуються тільки для роботи зі сторінками, що містять таблиці сторінок нижчого рівня.

Таблиці сторінок нижчого рівня можуть бути вивантажені з оперативної пам’яті. В оперативну пам’ять завантажується деяка кількість сторінок, з іншого боку завантаження сторінок уповільнює роботу системи, тобто розв’язання прикладної задачі, має місце непродуктивне пересилання даних. Чим більше в одиницю часу виникає переривань, тим повільніше працює система.

Розумно заздалегідь завантажити в оперативну пам’ять всі ті сторінки, які будуть використовуватися під час роботи. Було проведено спостереження і з'ясовано, що ділянки найчастіше використовуваного коду або даних концентруються на невеликій кількості сторінок. У будь-який момент часу t існує множина, в яку входять всі сторінки, які використовуються при послідовному зверненні до пам’яті, так звана робоча множина.

10.2.2 Сегментна модель віртуальної пам’яті

У традиційному розумінні в системі з сегментної організацією пам’яті знімається обмеження на фіксований розмір блоку пам’яті, а потім виконується довільне розбиття виконуваної програми на сегменти.

Сегмент – це одиниця логічного розбиття програм (процедура, модуль, область даних), що має змінний розмір і ім'я.

Сегментна організація [67] забезпечує простий і природній поділ загальних сегментів між декількома користувачами і захист сегментів за допомогою зв'язування з ними прав доступу, що підрозділяються на:

- виконання E;
- читання R;
- запис W;
- розширення A;
- і їх різних комбінацій.

Для сегментної організації в таблиці відображення сегментів з кожним сегментом пов'язано дескриптор, який містить адресу розміщення сегмента в оперативній пам'яті, його довжину і права доступу (ERWA), ознаку звертання до сегменту за останній інтервал часу, а також ознака присутності в фізичній оперативній пам'яті. Якщо віртуальні адресні простори декількох процесів включають один і той же сегмент, то в таблицях сегментів цих процесів робляться посилання на одну і ту ж ділянку оперативної пам'яті, в яку цей сегмент завантажується в єдиному екземплярі.

Переваги сегментної організації порівняно зі сторінковою:

- зменшено внутрішню фрагментацію, яка у сторінковій організації складає у середньому половину розміру сторінки на процес (програму);
- дані і програма цілком знаходяться в одному блоці, що прискорює роботу;
- спрощено управління доступом до сегменту з боку загальних процесів і його захист.

Недоліки:

- ускладнене управління і захист оперативної пам'яті. Тут необхідні ключі захисту індивідуальних сегментів пам'яті;
- велика зовнішня фрагментація, пов'язана з великою різницею між сегментами окремих процесів (програм). За таких умов, у багатозадачній операційній системі може виникнути ситуація, коли кілька невеликих процесів вивільнили оперативну пам'ять і є достатньо вільного об'єму, але один великий процес завантажити не можна, оскільки є зовнішня фрагментація на кілька замалих ділянок. Вихід – дефрагментувати (ущільнити розміщення процесів) оперативну пам'ять, але це суттєво знижує продуктивність системи.

Для боротьби із зовнішньою фрагментацією, окрім дефрагментації, існують методи раціонального використання пам'яті. Їх два:

- метод найкращої відповідності. Сегмент розміщуються у найменшу вільну і таку, що підходить за розмірами, ділянку оперативної пам'яті. Пам'ять витрачається раціонально;
- метод першої відповідності. Без додаткових дій з порівняння сегмент розміщується у першу, що підходить за розмірами, ділянку оперативної пам'яті. При цьому туди ж може вміститись ще один або кілька сегментів. Перевага над попереднім методом у тому, що тут не утворюється багато вільних

ділянок пам’яті малого розміру, куди потім майже неможливо розмістити сегмент. Тобто, втрати оперативної пам’яті через вплив зовнішньої фрагментації видаються меншими.

Насправді, перевага першого чи другого методу на сьогодні не доведена. А проблема зовнішньої фрагментації разом зі складністю роботи з оперативною пам’яттю призвели до того, що на сьогодні сегментна організація віртуальної пам’яті майже не використовується.

10.2.3 Сторінково-сегментна організація віртуальної пам’яті

За такої організації сегментна модель віртуальної пам’яті – абстракція більш високого рівня, ніж сторінкова модель, і пов’язана вона з роботою трансляторів.

Під час трансляції програмного коду утворюється кілька окремих ділянок [68]:

- ділянка даних;
- ділянка коду;
- ділянка стеку;
- ділянка адрес і посилань;
- лістинг програми.

Заздалегідь розміри таких сторінок визначити складно. Якщо задати завеликі обсяги – буде нераціональна витрата пам’яті, якщо замалі – ділянки можуть накладатись одна на одну, що призведе до псування їх вмісту. Сторінкова модель пам’яті прозора для програмного коду, тому цю проблему не вирішує.

Щоб проблему накладення ділянок вхідного коду ефективно вирішити, було запропоновано ізолювати кожну з них в окремому віртуальному адресному просторі – сегменті. Так з’явилися сегмент коду, даних, стека, що явно присутні у мовах асемблерів x86 та x64. Звичайно сегменти адрес, посилань і лістингу також неявно присутні під час трансляції.

Вищезгадані сегменти, на відміну від чистої сегментної моделі пам’яті, складаються із цілого числа сторінок віртуальної пам’яті, хоча властивість непостійної довжини сегмента зберігається. Такий підхід ще має назву двовимірна **модель пам’яті.** Розглянемо її роботу докладніше.

Завантаження процесу виконується операційною системою посторінково, при цьому частина сторінок розміщається в оперативній пам’яті, а частина на диску. Для кожного сегмента створюється своя таблиця сторінок, структура якої повністю збігається зі структурою таблиці сторінок, використовуваної при сторінковому розподілі. Для кожного процесу створюється таблиця сегментів, в якій вказуються адреси таблиць сторінок для всіх сегментів даного процесу. Адреса таблиці сегментів завантажується в спеціальний регістр процесора, коли активізується відповідний процес.

Трирівнева адресація (**але для двовимірної моделі віртуальної пам’яті)** [69] використовує дві таблиці відображення (таблиця сегментів + таблиця сторінок сегментів).

$$V = (S, P, d)$$

де S — номер сегмента; P — номер сторінки; d — зміщення.

Динамічне управління адресами тут виконується більш складно в два етапи, і тому в таких системах застосовується високошвидкісна асоціативна пам'ять, по типу кеш-пам'яті.

Звичайна адресована пам'ять — це пам'ять, до якої звертаються за адресою і вибирають значення.

Переваги сторінково-сегментної організації:

- комбінована організація ефективна для великих програм (процедур), що мають свою локальність;
- робить більш ефективним колективне використання загальних (поділюваних) сегментів, так як рядки різних таблиць сегментів будуть вказувати на одну таблицю сторінок колективно використовуваного сегмента.

10.2.4 Дворівнева сторінкова організація віртуальної пам'яті

Дворівнева сторінкова організація утворюється з гіперсторінок, що традиційно називаються сегментами, які в свою чергу поділяються на сторінки. Віртуальна адреса складається також з трьох компонентів (гіперсторінка, сторінка, зміщення) [70].

Переваги:

- скорочення обсягу пам**'**яті під таблиці сторінок шляхом розміщення в оперативній пам'яті тільки активних частин таблиць сторінок;
- з'являється можливість моделювати сегментну організацію, тобто забезпечити ефективне розділення загальних процедур і захист процесів за допомогою простішої схеми адресації.

Сторінки при цьому мають об'єм 2 або 4 кБ, гіперсторінки від 64 до 1024 Кб**.**

У загальному випадку інформація постійно знаходиться на нижчому рівні ієрархічної структури пам**'**яті, а механізм управління ієрархією забезпечує передачу на верхній рівень найбільш ймовірної для обробки інформації. У цьому випадку верхній рівень працює як кеш по відношенню до нижнього рівня. Спочатку ідея розглядалася тільки до пам'яті між **центральним процесором** і **оперативною пам'яттю**, потім поширилася в загальному сенсі до ієрархії пам'яті**.**

10.3 Висновки

Оперативна пам'ять комп'ютерів є обмеженим ресурсом, що ускладнює її використання для вирішення прикладних задач. Необхідність ефективної експлуатації такої пам'яті для порівняно апаратно-незалежної роботи додатків призвела до появи спочатку оверлейної, а згодом і сторінкової моделі пам'яті. І хоча на сьогодні оверлейна модель все ще використовується під час розробки компонентів операційних систем, таких, наприклад, як Linux, основною технологією, що отримала свій подальший розвиток, стала віртуальна сторінкова пам'ять, яка емулюється операційною системою і надає розробнику

прикладного програмного забезпечення повнофункціональну модель оперативної пам’яті дуже великого, умовно безкінечного об'єму. Вдалість рішення зі сторінковою віртуальною пам’яттю підтверджує наявність у сучасних процесорів апаратного модуля MMU для перетворення віртуальних адрес у фізичні в реальному часі, який також викликає сторінкові переривання. Еволюція сторінкової віртуальної пам’яті призвела до її гібридизації з менш поширеною сегментною моделлю, в результаті чого з’явилась сторінково-сегментна модель. Ця модель розділяє простір віртуальної пам’яті трансльованої, такої що проходить зв’язування та запущеної на виконання програми на спеціалізовані за своїм призначенням сегменти, які складаються з різної кількості, але цільних сторінок. Перевага такого підходу полягає у більшій захищеності частин програми під час обробки та виконання, а також спрощенні методів керування за рахунок Гарвардської моделі віртуальної пам’яті. З іншого боку, сторінково-сегментна організація пам’яті призводить до необхідності використання тривимірної схеми адресації, що підвищує кількість обчислень під час отримання фізичних адрес, а отже, знижує продуктивність. Тому існують альтернативні й додаткові підходи для підтримки тривимірної моделі, такі як використання гіперсторінок та асоціативної пам’яті, що будуть розглянуті в наступному розділі.

10.4 Питання для самоперевірки

1. Що таке оверлейна модель пам’яті?
2. В яких операційних системах для написання програм доцільно використовувати оверлейну модель пам’яті?
3. Навіщо потрібна віртуальна пам’ять?
4. Який розмір сторінкового блоку?
5. Як працює MMU?
6. Яку інформацію містить дескриптор сторінки віртуальної пам’яті?
7. Для чого потрібен диспетчер пам’яті ОС?
8. Як працює диспетчер пам’яті ОС?
9. З яких частин складається віртуальна адреса команди (даного)?
10. Як відбувається перетворення віртуальної адреси команди (даного) у фізичну?
11. Що собою являє застаріла сегментна модель віртуальної пам’яті?
12. Що таке зовнішня фрагментація пам’яті?
13. Що таке внутрішня фрагментація пам’яті?
14. Що собою являє сторінково-сегментна модель віртуальної пам’яті?
15. Які переваги сторінково-сегментної моделі віртуальної пам’яті?

11 ПРИНЦИПИ ПОБУДОВИ АСОЦІАТИВНОЇ ПАМ'ЯТІ

11.1 Принцип роботи

Асоціативна пам'ять – це засіб для пошуку інформації за пов'язаним з нею числовим значенням ключа. Принципи організації асоціативної пам'яті використовуються для зберігання даних у таблицях баз даних, для організації віртуальної пам'яті обчислювальних систем і комп'ютерів, для кеш-пам'яті комп'ютерів.

Асоціативна віртуальна пам'ять – це альтернативна модель пам'яті, яка дозволяє здійснювати прямий доступ до записів з фізичними адресами за ключовим значенням. Це значення можна назвати ключем пошуку запису в таблиці. Також може використовуватись індекс. Індекс є результатом застосування деякого алгоритму чи навіть математичної функції до ключа пошуку. Індекс чи ключ пошуку дозволяє вибирати табличний запис з адресою відповідної сторінки у фізичній пам'яті [70]. Схема роботи асоціативної пам'яті показана на рисунку 11.1. Асоціативна пам'ять прискорює роботу сторінково-сегментної структури віртуальної пам'яті у схемах з трирівневою адресацією.

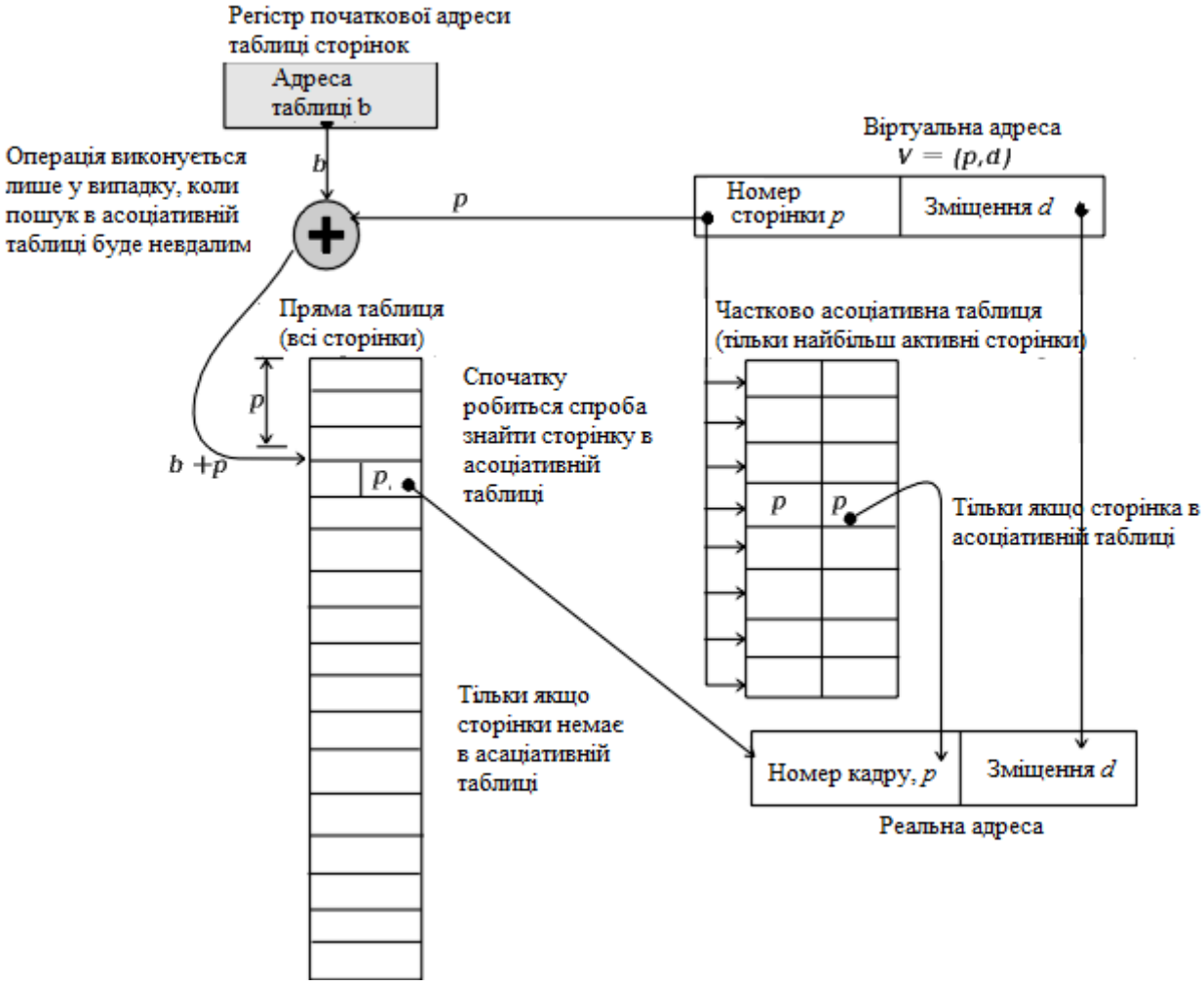

Рисунок 11.1 – Робота асоціативної пам'яті

11.2 Алгоритми пошуку

Відмінності видів асоціативної пам'яті пов'язані з алгоритмом чи методом пошуку. Будь-яка процедура пошуку в якості вхідного параметра використовує ключ пошуку, тобто деякий символ, ім'я чи значення. На виході процедури пошуку повинно з'явитися значення, асоційоване з ключем пошуку.

У найпростіших випадках використовується 2 простих алгоритми пошуку, а також методики, основані на них [71]:

1. Лінійний пошук. Для нього потрібна таблиця, в якій ведеться пошук з асоційованими парами ключ-значення, система ключів пошуку, дані про кількість записів у таблиці.

Алгоритм лінійного пошуку наступний:

1) Береться ключ до першого запису таблиці і порівнюється з ключем пошуку.

2) Якщо ключі співпали, зчитується асоційоване з ним значення або запис. Пошук закінчено.

3) Якщо ключі не співпали, зчитується ключ наступного запису і дії повторюються до тих пір, поки не співпадуть ключі або не буде досягнутий кінець таблиці.

Переваги: простота алгоритму, немає необхідності в попередньому сортуванні даних.

Недоліки: порівняно великі витрати часу на виконання.

2. Двійковий пошук. Для реалізації двійкового пошуку необхідні впорядковані, сортовані таблиці даних. Упорядкування має бути або за ключем, якщо він має числове вираження, або за індексом запису.

Для пошуку процедура двійкового пошуку в якості вхідної інформації повинна мати індекс запису, ключ запису або лише ключ, якщо він і є індексом, інформацію про розмірність таблиці і саму таблицю.

Алгоритм двійкового пошуку наступний:

1) Оцінюється діапазон поля пошуку, який на першому етапі дорівнює величині таблиці і зчитується індекс з середини таблиці. Зчитаний індекс порівнюється з індексом ключа пошуку.

2) Якщо індекс співпав, то порівнюються ключі пошуку (не обов'язково) і зчитується асоційоване значення.

3) Якщо пошуковий індекс більше зчитаного індексу ключа пошуку у таблиці, діапазон пошуку ділиться навпіл і в якості поля пошуку приймається половина таблиці з меншим індексом, якщо пошуковий індекс менше зчитаного, теж саме виконується для половини таблиці з більшими індексами ключів. Алгоритм повторюється доти, доки не буде знайдений шуканий запис або не буде чого ділити.

Переваги: швидший у порівнянні з лінійним алгоритм пошуку.

Недолік: складніший в реалізації і вимагає упорядкування таблиці, тобто сортування її даних.

11.3 Алгоритми сортування

Сортування даних масивах і таблицях необхідне, перш за все, для реалізації пошуку. Більшість сучасних алгоритмів сортування базуються на двох базових методах: обмінне (бульбашкове) сортування і сортування Шелла.

1. Алгоритм бульбашкового сортування [72]:

1) Порівняти ключі перших двох записів. Якщо перший запис більше другого, вони міняються місцями.

2) Порівняти ключі другого і третього запису, і якщо їх ключі стоять не у порядку зростання – поміняти їх місцями.

3) Тим же способом послідовно обробити пари ключів до кінця таблиці. При цьому найбільше значення ключа буде у результаті перенесене у запис N – останній запис, порядковий номер якого вказує на розмірність таблиці.

4) Описані у пп.1-3 дії повторюються для перших N-1 позицій таблиці, потім для перших N-2, доти, поки не буде досягнуто початку таблиці.

Переваги алгоритму: простота реалізації.

Недолік: великий час сортування, яке зі збільшенням кількості записів збільшується і сприяє збільшенню часу сортування в гіперболічної прогресії.

2. Алгоритм сортування Шелла. Алгоритм сортування Шелла показано на рисунку 11.2:

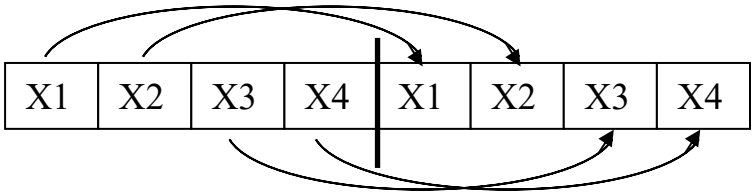

Рисунок 11.2 – Принцип сортування Шелла

Порядок дій згідно сортування Шелла наступний [73]:

1. Вся таблиця записів умовно розбивається навпіл і порівнюється перший елемент першої таблиці з першим елементом другої таблиці, другий елемент першої таблиці з другим елементом другої таблиці і так далі до кінця таблиць. Записи з меншими значеннями індексів переносяться у першу таблицю, з більшими – у другу.

2. Обидві таблиці також розбиваються кожна навпіл, і дії за п.1 виконуються для половинок першої і другої таблиць.

3. Пункти 2 і 3 виконуються доти, поки у фрагментах таблиці не залишиться по одному елементу і не буде виконано порівняння всіх пар.

Переваги: швидший алгоритм сортування.

Недолік: складний для реалізації, вимагає корекції для НЕПАРНИХ записів у порівнюваних таблицях.

11.4 Метод хеш-кодування

У базах даних, сховищах з великими обсягами, або там, де є жорсткі вимоги до швидкодії, замість сортування записів застосовують так зване хеш-кодування. Розглянемо його докладніше.

Кожен запис хеш-таблиці включає символьний ключ, індекс і збережені дані. Хеш-кодування полягає в отриманні з імені ключа деякого значення, яке є номером позиції в хеш-таблиці. Функція, за допомогою якої обчислюються хеш-позиції, називається хеш-функцією.

Хеш-функція, як і інше поняття – ключ кодування – терміни досить умовні. Це деяке двійкове число, котре може бути представлене у вигляді, наприклад, поліному з ВИКЛЮЧАЮЧИМ АБО:

$$p_0*2^0 \oplus p_1*2^1 \oplus p_2*2^2 \oplus \ldots \oplus p_n*2^n,$$

де p є {0;1} – значення 0 або 1 у відповідних розрядах цього числа, що представляє хеш-функцію; n – розрядність числа, що представляє хеш-функцію.

Сутність хеш-кодування в тому, щоб за ключем пошуку і за допомогою заздалегідь відомої хеш-функції обчислити номер позиції в хеш-таблиці, де повинен бути записаний індекс та посилання на дані у невпорядкованому сховищі або самі дані [74].

Якщо у результаті застосування хеш-функції під час внесення даних у сховище утворився номер позиції, де вже є запис, то нові дані мають бути записані у наступну позицію або далі, у першу незайняту і пошук цих даних буде здійснюватися спочатку за хеш-ключем, а потім за лінійним алгоритмом. Таким чином, у випадку невдалого вибору хеш-функції більшість індексів записуватимуться не на свої місця, а на вільні. Тоді пошук в хеш-таблиці вироджується в лінійний і продуктивність хеш-пошуку падає.

Таким чином, вибір та обґрунтування хеш-функції є нетривіальною задачею як при розробці апаратних засобів комп'ютерних систем і електронних сховищ, так і баз даних, тому для описаного вище методу рекомендують обирати хеш-функції, що можуть бути представлені у вигляді натурального двійкового поліному, або застосовувати більш складні алгоритми.

11.5 Висновки

Розглянута у представленому розділі модель асоціативної пам'яті представляє собою альтернативу трирівневій моделі оперативної пам'яті. Вона спрямована на прискорення роботи зі сторінками сегментів оперативної пам'яті шляхом прямого обчислення позиції фізичної адреси сторінки процесу в оперативній пам'яті за ключем пошуку. При цьому фізичні адреси розташовані в спеціальній асоціативній таблиці. Асоціативна таблиця, частіш за все, створюється не для всього процесу, а для найбільш часто використовуваних сторінок. Тому такий підхід виправданий також і для організації доступу до сторінок в кеш-пам'яті комп'ютера. Взагалі асоціативні таблиці й пошук за ключем притаманні не лише асоціативній пам'яті у складі моделей віртуальної пам'яті, але і для організації записів у таблицях даних взагалі. При цьому і для віртуальної пам'яті, і для інших випадків застосовують алгоритми пошуку, які,

за виключенням лінійного методу потребують сортування й впорядкування записів. Більшість методів сортування мають у своїй основі бульбашковий або метод Шелла, але для великих обсягів даних або там, де потрібна швидкодія і під час пошуку і під час первинного впорядкування записів використовують методи хеш-кодування за різного вигляду математичними функціями. Вибір правильної функції для ключа пошуку під час створення таблиці записів дозволяє в більшості випадків напряму звертатись до потрібного запису. Некоректний вибір навпаки, збільшує кількість випадків додаткового уточнюючого пошуку за лінійним алгоритмом, що знижує швидкодію. Тому вибір і обґрунтування виду хеш-функції є складною і важливою задачею.

11.6 Питання для самоперевірки

1. Що таке асоціативна пам’ять і для чого її використовують?
2. Що таке асоціативна віртуальна пам’ять?
3. Що таке ключ пошуку в асоціативній таблиці ?
4. Як утворюється індекс запису в асоціативній таблиці?
5. Навіщо потрібна асоціативна пам’ять під час підтримки моделі віртуальної пам’яті?
6. Як працює алгоритм лінійного пошуку?
7. Чи потрібно для лінійного пошуку сортувати таблицю за ключами чи індексами?
8. Як працює алгоритм двійкового пошуку?
9. Чи потрібно для двійкового пошуку сортувати таблицю?
10. У чому відмінність ключа від індекса?
11. Які переваги алгоритму двійкового пошуку над лінійним?
12. Які переваги алгоритму лінійного пошуку над двійковим?
13. Як працює алгоритм бульбашкового сортування?
14. Як працює алгоритм сортування Шелла?
15. У чому переваги алгоритму бульбашкового сортування?
16. У чому переваги алгоритму сортування Шелла?
17. У чому недоліки алгоритму сортування Шелла?
18. З чого складається запис хеш-таблиці?
19. У чому сутність хеш-кодування під час розміщення та пошуку записів у хеш-таблиці?
20. Якщо після застосування ключа до хеш-функції утворився індекс, який вказує на дані з іншим ключем, як працює хеш-алгоритм?

12 ОРГАНІЗАЦІЯ ВВЕДЕННЯ-ВИВЕДЕННЯ ДАНИХ

12.1 Під’єднання зовнішніх пристроїв до комп’ютера

Зовнішні (периферійні) пристрої під’єднуються до обчислювальних вузлів за допомогою відповідних інтерфейсів. Інтерфейси характеризуються певними типами роз’ємів, сигналів та протоколами обміну. По відношенню до комп’ютера зовнішніми пристроями можуть бути інші комп’ютери та пристрої введення-виведення, датчики, прилади автоматики виконавчих механізмів, телекомунікаційні вузли. Щодо пристроїв введення-виведення комп’ютера до них належать наступні [76]:

– основні засоби введення, зокрема клавіатура, миша, джойстик;

– оптичні пристрої, зокрема DVD-привод, зчитувач штрихових кодів, цифрова відеокамера, інше;

– магнітні пристрої, наприклад, HMDD-диск, пристрій для зчитування коду на магнітних стрічках, RFID-міток, інше;

– екранні пристрої, такі як рідкокристалічні екрани різних технологій, сенсорні екрани, світлове перо;

– звукові пристрої, такі як вбудовані динаміки, колонки, мікрофони, навушники, інше;

– засоби створення документів на паперових носіях, наприклад, різні принтери, графопобудовувачі, інше.

Рисунок 12.1 демонструє роботу системи введення-виведення комп’ютера. Тут представлені деякі периферійні пристрої введення та виведення даних з властивими їм пристроями керування, буферною чи кеш-пам’яттю та інтерфейсами обміну. З боку системної плати комп’ютера, у свою чергу, також представлені інтерфейсні функціональні блоки, шина введення-виведення та спеціалізовані контролери, які керують обміном даними між периферією та пам’яттю комп’ютера. Інтерфейси розробляються спеціально під конкретний пристрій введення-виведення, хоча існує певна уніфікація, наприклад, інтерфейси USB. Інтерфейси повідомляють контролери про готовність зовнішніх пристроїв до приймання наступного блоку чи окремого екземпляра даних. Також через інтерфейси зовнішні пристрої отримують інформацію про те, що внутрішні пристрої комп’ютера готові одержати від них черговий блок даних.

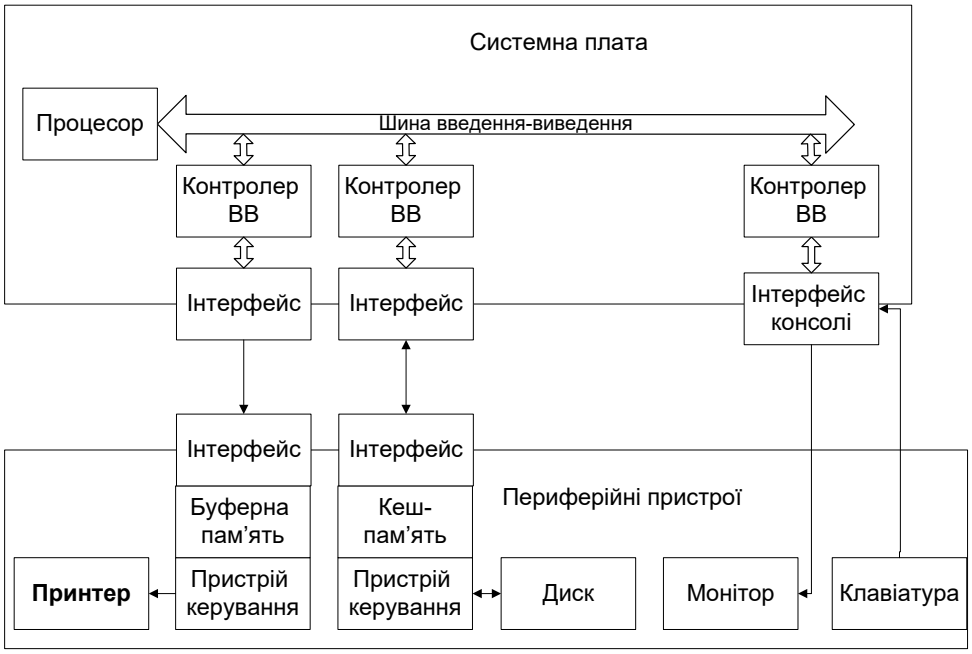

Рисунок 12.1 – Організація введення-виведення на прикладі шинної архітектури обчислювального вузла

Звичайно, приклад, показаний на рисунку 12.1 не відображає всі особливості введення-виведення на сучасній системній платі. Це, скоріше, архітектура промислового обчислювального пристрою чи контролера для дослідів. Щодо введення-виведення для саме персонального комп'ютера, то варто звернутись до рисунків 4.2 – 4.4 розділу 4, там де розглянуто шинно-мостова та хабова архітектури. Там показано шини введення-виведення різних типів, спеціалізовані чипсети, а також згадано контролери DMA та APIC.

Якщо розглянути рисунок 12.1, можна помітити, що для деяких периферійних пристроїв представлено буферну чи кеш-пам'ять, а для деяких, зокрема консолі, ні. Це пов'язано з тим, що ряд зовнішніх пристроїв, наприклад, жорсткий диск чи принтер, працюють з великими об'ємами даних – сторінками, дисковими блоками, сегментами чи екстентами. З іншого боку, згадані пристрої працюють повільніше операційної частини комп'ютера. Для узгодження роботи швидкої операційної системи й більш повільних периферійних пристроїв потрібна буферизація. Але буферизація виправдана не завжди. Так, скан-коди клавіатури чи сигнали від миші надходять порівняно повільно і їх обсяг малий, причому потрібна швидка реакція від обчислювальної системи в інтерактивному режимі. Також питання виведення на монітор – тут йде потік великого обсягу готових сигналів на засіб швидкого виведення. І перший, і другий випадок не потребують додаткової буферизації.

12.2 Способи розпізнавання пристроїв уведення-виведення

Способи розпізнавання пристроїв введення-виведення залежать від схем їх підключення до процесора.

На рисунку 11.2 показано варіанти організації системи введення виведення обчислювального пристрою.

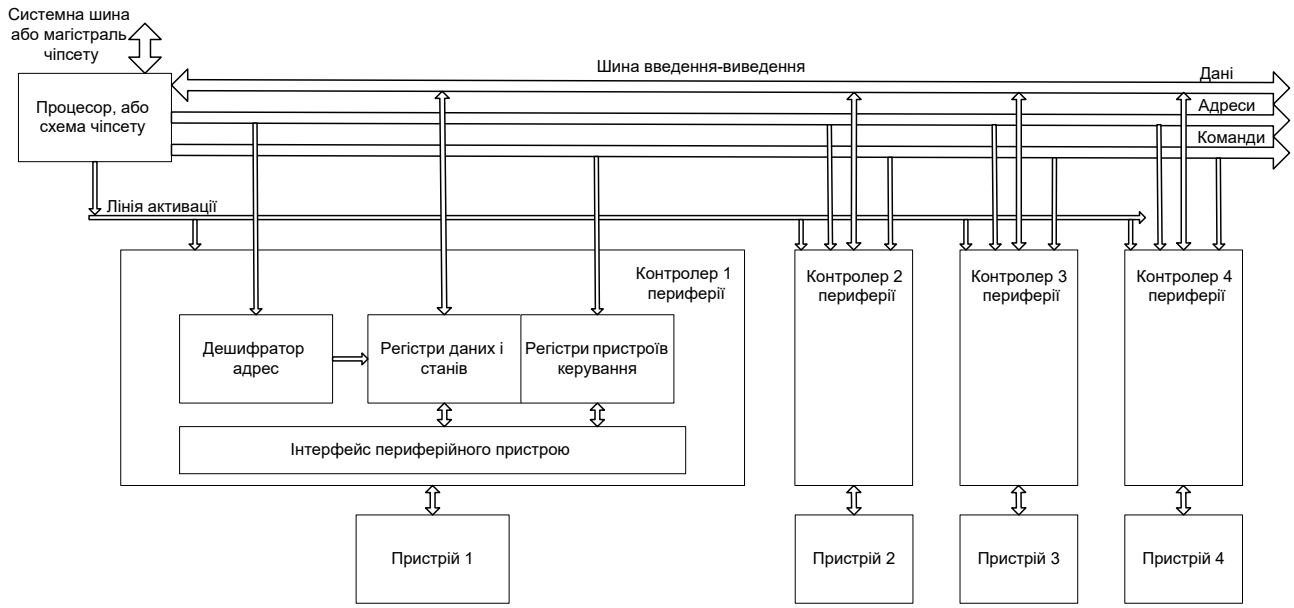

Рисунок 12.2 – Варіанти організації системи введення виведення обчислювального пристрою

У випадку, показаному на рисунку 12.2, процесор, контролер прямого доступу до пам’яті, контролер південного хабу комп’ютера чи інший засіб керування введенням-виведенням звертається до контролера зовнішнього пристрою за його номером, який можна вважати фізичною адресою простору пам’яті, незалежного від оперативної. Цей простір відповідає саме за взаємодію з пристроями введення-виведення. При цьому контролери периферії використовують дешифратори адрес, за якими через шину введення-виведення може здійснюватися обмін даними й командами між пристроєм периферії та відповідними пристроями системної плати.

На рисунку 12.2 системна шина показана як окрема від шини введення-виведення. Але в простіших системах можлива загальна шина пам’яті та введення-виведення, як це реалізовано на внутрішньоконтролерній магістралі мікроконтролерів x51 [36].

В системах може бути використана, але необов’язково, окрема лінія активації, по якій керуючий вузол обирає для обміну даними певний пристрій введення виведення [35], а у випадку із загальною шиною і певну ділянку оперативної пам’яті. Сигнали активації можуть сприйматись відповідними контролерами або як окремі розряди шини адреси, що підлягають дешифрації, або як дійсно сигнали активації. В останньому випадку контролер має перервати свій поточний стан – сон з пониженим енергоспоживанням, самодіагностику,

цикл обміну з периферійним пристроєм, інші дії, а потім підготуватись до прийому команд чи даних. Іноді лінія активації може використовуватись як частина шини керування, наприклад, подавати сигнали керування для введення чи виведення даних, або запитувати прямий обмін з пам'яттю, або запитувати переривання процесора на обслуговування керованого пристрою.

Для складних пристроїв введення-виведення лінії активації може бути недостатньо, або вона може бути і непотрібною. Відповідно до рисунка 12.2, контролер периферії має деяку групу регістрів даних, доступних через шину даних для введення-виведення і групу регістрів керування, доступних через шину керування. Всі ці регістри мають свої адреси в деякому адресному просторі контролера периферії. І вони взаємодіють з інтерфейсом периферійного пристрою – електронною схемою, призначеною для роботи з пристроєм певного призначення: принтером, звуковим динаміком, провідною чи безпровідною мережею тощо.

Функціональні вузли контролера периферії виконують наступне:

– дешифрацію власної, а отже і пристрою, адреси, за якою до нього йде звернення від процесора чи інших пристроїв системної плати;

– дешифрацію адрес регістрів даних і команд, якщо це передбачено регламентом взаємодії з периферійним пристроєм;

– приймання і дешифрацію команд керування пристроєм, якщо такий функціонал передбачений;

– керує взаємодією і обміном даними між пристроєм введення-виведення та процесором чи іншим пристроєм системної плати.

12.3 Методи керування введенням-виведенням

Для комп'ютерних система існує чотири загальних методи керування введенням-виведенням:

– програмнокероване введення-виведення;

– кероване перериваннями введення-виведення (введення-виведення за перериваннями);

– прямий доступ до пам'яті;

– введення-виведення під керуванням периферійних процесорів (каналів).

Розглянемо кожний з них окремо.

12.3.1 Програмнокероване введення-виведення

В обчислювальних системах, що використовують програмнокероване введення-виведення, в інтерфейсній схемі кожного пристрою введення-виведення наявний регістр команд і станів. Процесор постійно опитує регістр команд і станів кожного пристрою введення-виведення, на предмет надходження даних або готовності прийняти чергові дані. Такий метод ще називають

введенням-виведенням за запитами. Як тільки процесор виявляє стан готовності до прийому чи передачі, він надсилає відповідну команду в регістр відповідного пристрою введення-виведення.

Переваги методу полягають в його простоті й програмному контролі над поведінкою периферійного пристрою. А недоліки є наслідком стану активного очікування процесора. Тобто процесор, коли вирішує задачу взаємодії з периферією, не виконує іншої роботи. Продуктивність обчислювальної системи знижується до продуктивності периферійного пристрою. Очевидно, що розглянутий метод не годиться для застосування в універсальних обчислювальних засобах і може бути корисним у спеціальних системах, наприклад, недорогих маршрутизаторах, засобах автономного протоколювання сигналів і подій, засобах керування освітленням, чи щось подібне.

12.3.2 Введення-виведення за перериваннями

Цей метод виключає активне очікування процесором готовності периферійного пристрою до обміну. Процесор виконує свою задачу, а в цей час повільний пристрій периферії простоює або виконує свою. Після того, як чергова порція даних була прийнята у регістри контролера периферії або відправлена назовні, контролер формує по виділеній лінії сигнал-запит на переривання процесора. Цей запит надходить не напряму процесору, а спеціальному контролеру переривань, можливо, вбудованому в мікросхему процесора, як це реалізовано в контролерах PIC та x51 [35, 36]. Контролер переривань в залежності від налаштувань пріоритетів переривань і поточних дозволів надає процесу запит на переривання вирішення поточної задачі на обслуговування пристрою введення виведення. Процесор, в залежності від налаштувань, може або почекати завершення більш пріоритетних завдань-обробників переривань, або перервати поточну роботу. Під час переривання процесор відповідає контролеру переривань сигналом-дозволом на обслуговування переривання. Після цього поточне значення програмного лічильника процесора буде збережене і замінене на вектор переривання – початкова адреса програми-обробника переривання, яка буде отримана від контролера переривань. Обробник переривання свою роботу починає зі збереження поточного стану активних регістрів процесора, потім виконує корисну роботу. Перед завершенням обробник відновлює стани активних регістрів, щоб програма, яка виконувалась до цього, коректно продовжила роботу. Потім процесор сигналізує контролеру переривань про те, що переривання, під час якого читались чи записувались на зовнішній пристрій дані, оброблене і можна продовжити виконання перерваної роботи. Пристрій, що отримав або передав дані, також знімає свій запит.

12.3.3 Прямий доступ до пам’яті

Цей режим використовується для звільнення процесора від участі у введенні-виведенні з пристроями, передача даних з яких здійснюється з частотою, незалежною від процесора, коли передача блоків даних здійснюється прямо

між пам’яттю і пристроями введення-виведення. Наприклад, магнітний диск не можна зупинити після записування символу. Або на лазерний принтер відправляється відразу великий блок даних, часто весь документ відразу. У таких випадках використовують контролер прямого доступу до пам’яті. Контролер ПДП може мати канали обміну даними, кількість яких зумовлена кількістю пристроїв, які використовують ПДП. Для організації прямого доступу до пам’яті процесор задає контролеру: адресу пам’яті, з якої буде здійснюватися обмін; кількість регістрів пам’яті, які приймають участь у обміні; адресу області даних на пристрої введення-виведення. Контролер прямого доступу до пам’яті та процесор розділяють шину пам’яті, тому готовий до обміну контролер має надіслати запит процесору запит на використання шини і отримати відповідний дозвіл. Після закінчення введення-виведення, чи при наявності помилки, система прямого доступу до пам’яті повідомляє про це процесор відповідним сигналом переривання. Пристрої введення-виведення мають перевагу над процесором при взаємодії з пам’яттю, але оскільки пересилання інформації від пристроїв введення-виведення здійснюється блоками не досить довгої тривалості, що у поєднанні з використанням кешу — це не суттєво впливає на продуктивність процесора. Тим не менш, процесор надає дозвіл контролеру ПДП працювати і повинен вживати спеціальних заходів у випадку збою чи помилки сеансу прямого обміну даними між периферією і пам’яттю.

12.3.4 Введення-виведення під керуванням периферійних процесорів

Ця технологія використовується великими системами, орієнтованими на багатьох користувачів, наприклад, мейнфреймами. Інша назва технології – канальне введення-виведення. Вона полягає в тому, що окремі процесори введення-виведення, які ще називають **каналами**, керують різними трактами введення-виведення. Тракти для більш повільних пристроїв можуть підключатись до одного процесора введення-виведення. Таким чином утворюється **мультиплексований канальний тракт**. Якщо пристрій швидкий з точки зору обміну даними, наприклад, жорсткий накопичувач, для нього буде виділено окремий процесор-канал. Такий процесор називається **селекторним каналом**. Процесори введення-виведення є пристроями, здатними виконувати програми, які виконують арифметично-логічні команди та команди переходу. Разом з тим вони оптимізовані саме для виконання операцій введення-виведення і виконують програми, які попередньо були розміщені центральним процесором в основній пам’яті. Ці програми включають команди пересилання даних і команди керування пристроями введення-виведення.

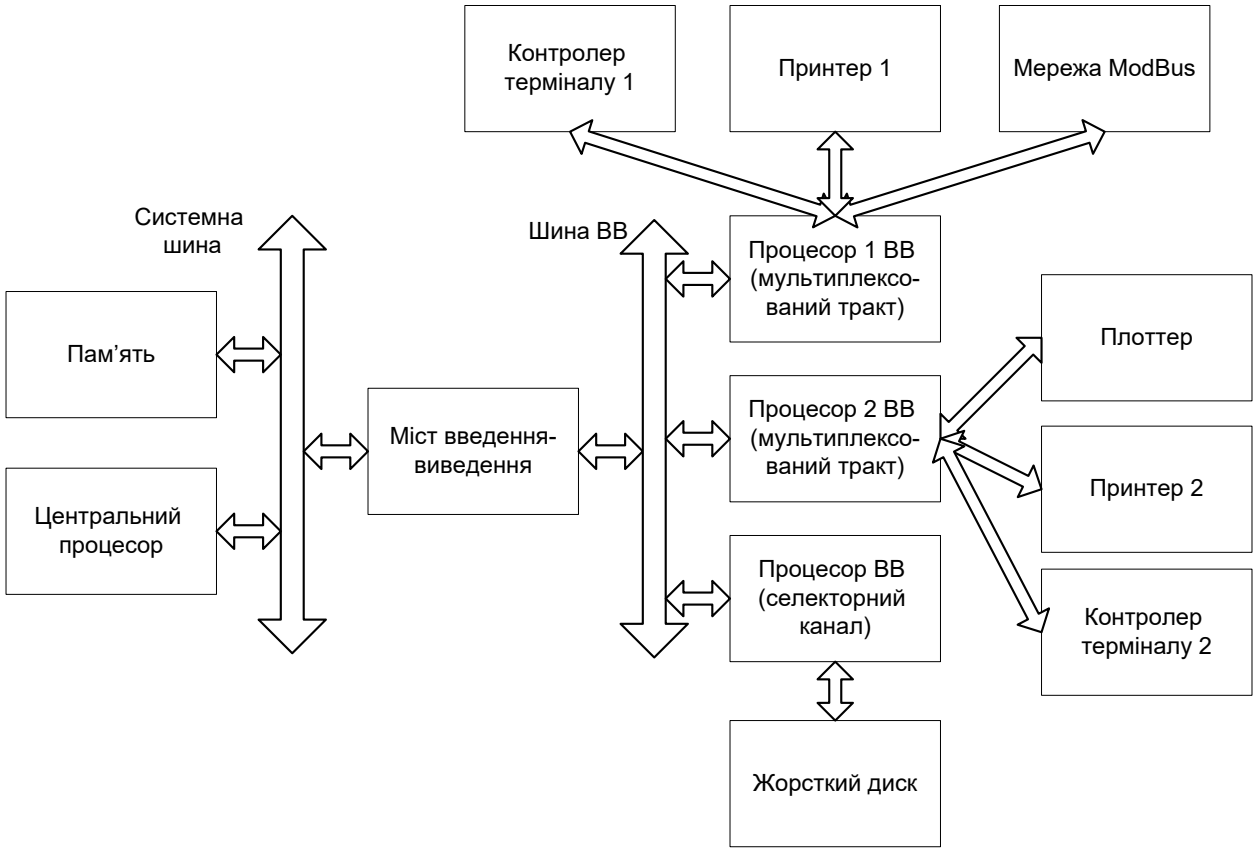

Рисунок 12.3 – Конфігурація системи канального введення-виведення

Технологія канальних процесорів надає більше можливостей по забезпеченню протоколів обміну, керуванням периферією, обміном великими обсягами даних незалежно від центрального процесора. Це дозволяє використовувати канальне введення-виведення в середовищах високопродуктивної обробки запитів, де його вартість і складність можуть бути виправдані.

12.3 Висновки

Розділ розглядає організацію введення-виведення між системною платою комп'ютера і периферійними пристроями. Так чи інакше взаємодію внутрішньої обчислювальної частини комп'ютера із зовнішніми периферійними пристроями забезпечують спеціалізовані контролери введення-виведення та електричні інтерфейси. Периферійні пристрої в залежності від принципу їх дії і призначення можуть бути обладнані спеціалізованими схемами інтерфейсів і засобами буферизації та кешування даних. З боку контролерів периферії також можуть бути застосовані як мінімум проміжні буферні регістри даних. Самі контролери периферії мають, як правило, внутрішні регістри не лише для обміну даними, але й для відображення поточної інформації про стан процесу приймання чи передачі даних для керування процесом введення-виведення, регістри для керування режимами роботи самого контролера, дешифратор адрес плати контролера периферії, щоб процесор, схема чипсету чи інші пристрої могли взаємодіяти з контролером периферії. Для контролерів периферії може, але необов'язково, використовуватись додаткова лінія активації

контролера, хоча це рішення більш притаманне скоріше обчислювальним системам спеціального призначення. З точки зору архітектури, контролери периферії можуть бути у тому ж адресному просторі й фізично на системній шині обчислювального вузла, разом з процесором і основною пам'яттю, а можуть бути відокремлені основним процесором чи процесором введення-виведення чи схемою чипсету в окрему шину введення-виведення селекторну чи мультиплексовану. В будь-якому випадку шина введення-виведення матиме функціональні підшини, подібні підшинам системної.

З точки зору керування процесом введення-виведення розглянуто основні методи з різною інтенсивністю використання центрального процесора з точки зору використання їх для обчислювальних систем різного призначення.

12.4 Питання для самоперевірки

1. Які існують пристрої введення-виведення?
2. Які інтерфейси введення-виведення Ви знаєте?
3. Навіщо потрібна буферна пам'ять в пристроях введення-виведення?
4. Які існують способи розпізнавання пристроїв введення-виведення?
5. Для чого використовується лінія активації контролера пристрою?
6. Які функції виконують розряди шини адреси під час роботи з контролером периферії?
7. Які функції виконує контролер периферії?
8. Які методи керування введенням виведенням вважаються основними?
9. В чому полягає метод програмнокерованого введення-виведення?
10. Які переваги програмнокерованого введення-виведення?
11. Які недоліки програмнокерованого введення-виведення?
12. В чому полягає метод введення-виведення за перериваннями?
13. Які функції виконує контролер переривань?
14. Що таке обробник переривань?
15. Де зберігається вектор переривань?
16. У чому перевага введення-виведення за перериваннями перед керованим?
17. В чому полягає метод прямого доступу до пам'яті?
18. Як взаємодіють процесор і контролер прямого доступу до пам'яті?
19. Що собою являє технологія введення-виведення під керуванням периферійних процесорів?
20. Чим відрізняється мультиплексований канал від селекторного?

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

AGP	–	англ. Accelerated Graphics Port, прискорений графічний порт;
APIC	–	англ. Advanced Programmable Interrupt Controller, розширений програмований контролер переривань;
ATA	–	англ. Advanced Technology Attachment, інтерфейс передової технології підключення;
ANSI	–	англ. American National Standards Institute, Американський національний інститут стандартів;
ASCII	–	англ. American Standard Code for Information Interchange, американський стандартний код для обміну інформацією;
BEDO DRAM	–	англ. Burst Extended Data Output Dynamic Random Access Memory, динамічна пам’ять з блоковим (пакетним) розширеним виведенням даних;
BIOS	–	англ. Basic Input/Output System, базова система введення-виведення даних;
CAS	–	англ. Column Address Strobe, активація потрібної колонки пам’яті у вже активованому рядку;
CD	–	англ. Compact Disc, компакт-диск;
CISC	–	англ. Complex Instruction Set Computer, комп’ютер з повним набором команд;
CMOS	–	англ. Complementary Metal-Oxide-Semiconductor, комплементарна метал-оксид-напівпровідникова технологія;
CPU	–	англ. Central Processing Unit, центральний процесорний вузол;
DDR SDRAM	–	англ. Double Data Rate Synchronous Dynamic Random Access Memory, синхронна динамічна оперативна пам’ять із подвоєною швидкістю передачі даних;
DVD	–	англ. Digital Versatile Disc або Digital Video Disc, цифровий універсальний диск або цифровий відеодиск відповідно, обидва варіанти вважаються правильними;
DMA	–	англ. Direct Memory Access, прямий доступ до пам’яті;
Direct RDRAM, Direct Rambus DRAM		англ. Direct Rambus Dynamic Random Access Memory, динамічна оперативна пам’ять Rambus із прямим високошвидкісним інтерфейсом;
DRAM	–	англ. Dinamic Random Access memory, напівпровідникова динамічна пам’ять довільного доступу;

EDO DRAM – англ. Extended Data Out Dynamic Random Access Memory, динамічна оперативна пам’ять з розширеним виведенням даних;

EEPROM англ. Electrically Erasable Programmable Read-Only Memory, програмована пам’ять тільки для читання з можливістю електричного стирання;

EPROM – англ. Erasable Programmable Read Only Memory, програмована пам’ять тільки для читання з можливістю електричного стирання;

ESDRAM – англ. Enhanced Synchronous Dynamic Random Access Memory, покращена синхронна динамічна оперативна пам’ять;

FIFO – англ. firs input – first output, перший увійшов – перший вийшов;

FPM DRAM – англ. Fast Page Mode Dynamic Random Access Memory, динамічна оперативна пам’ять із швидким посторінковим доступом;

FSB – англ. Front Side Bus, фронтальна (системна) шина;

FWH – англ. Firmware Hub, концентратор мікропрограмного забезпечення;

HMDD, HDD – англ. Hard (Magnetic) Disk Drive, накопичувач даних на жорсткому (магнітному) диску;

HMI – англ. Human Machine Interface, людино-машинний інтерфейс;

(E)ISA – англ. Extended Industry Standard Architecture, розширена архітектура стандартної промислової шини;

IRQ – англ. Interrupt Request, запит на переривання процесора;

LPC – англ. low pin count, шина з малою кількістю виводів;

LIFO – англ. last input – first output, останній увійшов – перший вийшов;

MCA – англ. Micro Channel Architecture, архітектура мікроканальної шини;

MCH – англ. Memory Controller Hub, концентратор контролерів пам’яті;

PASR англ. Partial Array Self Refresh, часткове самовідновлення масиву пам’яті;

MISC – англ. Minimal Instruction Set Computer, комп’ютер з мінімальним набором команд;

MMU – англ. Memory Management Unit, блок керування пам’яттю;

MMX – англ. Multimedia Extensions, мультимедійні розширення;

ICH2 – англ. I/O Controller Hub 2, концентратор контролерів введення-виведення даних другого покоління;

IDE – англ. Integrated Drive Electronics, інтегроніка дискового приводу, або інтегрована електроніка накопичувача;

I/O devices – англ. Input/Output devices, пристрої введення-виведення даних;
IoT – англ. Internet of Things, інтернет речей;
RAM – англ. Random Access Memory, оперативна пам’ять з довільним доступом;
RISC – англ. Reduced Instruction Set Computer, комп’ютер зі скороченим набором команд;
ROM – англ. Read Only Memory, пам’ять тільки для читання;
SRAM – англ. Static Random Access Memory, статична пам’ять довільного (інакше кажучи, прямого) доступу;
SSD – англ. Solid State Drive, накопичувач на твердотільній пам’яті, твердотільний накопичувач;
TLB – англ. Translation Lookaside Buffer, буфер швидкого пошуку трансляцій адрес;
PCI – англ. Peripheral Component Interconnect, з’єднання між периферійними компонентами;
PCI-E, PCI Express – англ. Peripheral Component Interconnect Express, швидкісне з’єднання між периферійними компонентами (експрес-варіант);
PCMCIA – англ. Personal Computer Memory Card International Association, Асоціація виробників плат пам’яті персональних комп’ютерів;
VESA – англ. Video Electronics Standards Association, Асоціація стандартів відеоелектроніки;
Video RAM – англ. Video Random Access Memory, відеопам’ять з довільним доступом;
VLB – англ. Vesa Local Bus, локальна шина VESA;
VLIW – англ. Very Long Instruction Word, дуже довга машинна команда;
PM DRAM – англ. Page Mode Dynamic Random Access Memory, динамічна оперативна пам’ять з посторінковим режимом доступу;
POST – англ. Power-On Self-Test, самотестування під час увімкнення живлення;
PROM – англ. англ. Programmable Read-Only Memory, програмована пам’ять тільки для читання;
RAS – англ. Row Address Strobe, активація потрібного рядка пам’яті;
RAS-to-CAS Delay, tRCD – англ. Row Address Strobe to Column Address Strobe Delay, затримка від появи стробу рядка до появи стробу колонки;
RTS – англ. Real Time Clock, годинник реального часу;
SATA – англ. Serial Advanced Technology Attachment, послідовний інтерфейс передової технології підключення;

SCADA	–	англ. Supervisory Control And Data Acquisition, диспетчерське керування та збір даних;
SDR SDRAM	–	англ. Single Data Rate Synchronous Dynamic Random Access Memory, синхронна динамічна оперативна пам’ять з одинарною швидкістю передачі даних;
SIMD	–	англ. Single Instruction-stream Multiple Data-stream, один потік команд для кількох потоків даних;
SISD	–	англ. Single Instruction-stream Single Data-stream, один потік команд для одного потоку даних;
SSE	–	англ. Streaming SIMD Extensions, потокове розширення для архітектури процесора, яке використовує технологію «один потік команд для кількох потоків даних»;
TCSR	–	англ. Temperature Compensated Self Refresh, температурно компенсоване самовідновлення;
USB	–	англ. Universal Serial Bus, універсальна послідовна шина;
ПІД-регулятор	–	пропорційно-інтегрально-диференціальний регулятор;
АС	–	автоматизована система;
АЛП	–	арифметико-логічний пристрій;
БМК	–	базові матричні кристали;
ЕОМ	–	електрична обчислювальна машина;
КМОП	–	кремній-метал-оксид-напівпровідник;
Контролер ВВ	–	контролер введення-виведення даних;
ОЗП	–	оперативний запам’ятовувальний пристрій;
ОС	–	операційна система;
ПЛІС	–	програмовані логічні інтегральні схеми;
ПК	–	персональний комп’ютер;
ПЛ	–	програмний лічильник;
РЗП	–	регістри загального призначення;
ЦПВ	–	центральний процесорний вузол.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Jones A. A. Portable Cosmos: Revealing the Antikythera Mechanism, Scientific Wonder of the Ancient World : Monograph. New York: Oxford University Press, 2017. 288 p.

2. Індо-арабська система числення. Режим доступу: https://uk.wikipedia.org/wiki/Індо-арабська_система_числення (дата оновлення: 26.09.2025).

3. Стефанович Д., Янковий В. Метрична система: від вибору одиниць до міжнародної конвенції… // Сторінки історії. Вип. 31. Київ : КПІ, 2010. С. 52–59.

4. Лічильна машина // Вікіпедія. Режим доступу: https://uk.wikipedia.org/wiki/Лічильна_машина (дата оновлення: 31.07.2025).

5. Wilhelm Schickard // Wikipedia. Режим доступу: https://en.wikipedia.org/wiki/Wilhelm_Schickard#History (дата оновлення: 12.10.2025).

6. Готфрід Вільгельм Лейбніц // Вікіпедія. Режим доступу: https://uk.wikipedia.org/wiki/Готфрід_Вільгельм_Лейбніц (дата оновлення: 6.02.2026).

7. Жакардовий ткацький верстат // Вікіпедія. Режим доступу: https://uk.wikipedia.org/wiki/Жакардовий_ткацький_верстат (дата оновлення: 4.02.2026).

8. Чарлз Беббідж // Вікіпедія. Режим доступу: https://uk.wikipedia.org/wiki/Чарлз_Беббідж (дата оновлення: 21.03.2026).

9. Лук’янюк В. Табулятор Германа Холлеріта // Цей день в історії. 2012. 20 серпня. URL: https://www.jnsm.com.ua/h/0108M/ (дата оновлення: 20.08.2012).

10. Технології Стародавнього Єгипту // Вікіпедія. Режим доступу: https://uk.wikipedia.org/wiki/Технології_Стародавнього_Єгипту (дата оновлення: 26.08.2025).

11. Інтегратор Лук’янова… // Proexpress. Режим доступу: https://proexpress.com.ua/uk/integrator-lykianova-ili-kak-v-sssr-naychilis-reshat-differencialnye-yravneniia-vodoi (дата оновлення: 15.02.2020).

12. Phillips Machine // Grokipedia. Режим доступу: https://grokipedia.com/page/Phillips_Machine (дата оновлення: 08.11.2025).

13. Сташкевич П. М., Лукінюк М. В. Технічні засоби автоматизації. Математичні операції на пневматичних елементах та їх використання в системах керування [Електронний ресурс] : навч. посібн. / КПІ ім. І. Сікорського. Київ : КПІ ім. І. Сікорського, 2019. 319 с. 1 електрон. опт. диск (1 файл : 4,52 Мбайт).

14. Історія створення годинника коротко. Сонячний годинник // Lada-fm. Режим доступу: https://lada-fm.com.ua/istoriya-stvorennya-godinnika-korotko-sonyachnij-godinnik/ (дата оновлення: 31.05.2017).

15. Астролябія – чудесний комп’ютер давнини // Lifeglobe. Режим доступу: https://lifeglobe.net/entry/1845 (дата оновлення: 3.12.2025).
16. Пам'ять на магнітних осердях // Вікіпедія. Режим доступу: https://uk.wikipedia.org/wiki/Пам%27ять_на_магнітних_осердях (дата оновлення: 24.05.2023).
17. Вчені з IBM створили картридж на 300 ТБ… // Futurum. Режим доступу: https://futurum.today/vcheni-z-ibm-zmohly-stvoryty-kartrydzh-na-300-terabait-shcho-vmishchaietsia-v-doloniu/ (дата оновлення: 02.08.2017).
18. Нечволода Л. В., Решетняк Т. В., Сташкевич І. І. Архітектура обчислювальних систем : навч. посіб. : Вид. 2-ге, перероб. і доп. Краматорськ–Тернопіль : ДДМА, 2023. – 204 с.
19. Illingworth V., Pyle I. C., Glaser E. Dictionary of Computing. 3rd ed. Oxford; New York : Oxford University Press, 1991. 521 p.
20. Мельник А. О. Архітектура комп'ютера : підручник. Луцьк : Волинська обласна друкарня, 2008. 470 с.
21. Jiménez M., Palomera R., Couvertier I. Introduction to Embedded Systems: Using Microcontrollers and the MSP430. New York : Springer, 2014. 648 p.
22. Тарарака В. Д. Архітектура комп'ютерних систем : навч. посіб. Житомир : ЖДТУ, 2018. 383 с.
23. Материнська плата // Вікіпедія. Режим доступу: https://uk.wikipedia.org/wiki/Материнська_плата (дата оновлення: 14.03.2026).
24. PCMCIA // Вікіпедія. Режим доступу: https://uk.wikipedia.org/wiki/PCMCIA (дата оновлення: 09.12.2024).
25. PCI Express // Вікіпедія. Режим доступу: https://uk.wikipedia.org/wiki/PCI_Express (дата оновлення: 05.09.2023)
26. Комп’ютерний кластер // Вікіпедія. Режим доступу: https://uk.wikipedia.org/wiki/Комп%27ютерний_кластер (дата оновлення: 31.08.2025).
27. Абрамов В. О. Архітектура електронно-обчислювальних машин : навч. посіб. Київ : КМПУ імені Б. Д. Грінченка, 2007. 84 с.
28. Шеховцов В. А. Операційні системи : підручник. Київ : BHV, 2005. 573 с.
29. Tanenbaum A. S., Austin T. Structured Computer Organization. 6th ed. New Delhi : Pearson Education India, 2016. 784 p.
30. Marwedel P. Embedded System Design. 4th ed. Cham : Springer International Publishing AG, 2021. 433 p.
31. Intel. 8086 16-bit HMOS Microprocessor. Santa Clara, CA : Intel Corp., 1978. 30 p.
32. Smith B. E., Johnson M. T. Programming the Intel 80386. Glenview, Illinois : Scott, Foresman, 1987. 346 p.
33. Intel. Intel i486 Microprocessor. Santa Clara, CA : Intel Corp., 1989. 176 p.

34. Gerber R., Binstock A. Programming with Hyper-Threading Technology. Hillsboro, OR : Intel Press, 2004. 220 p.

35. Advanced Micro Devices, Inc. AMD64 Architecture Programmer's Manual. Vol. 1: Application Programming. Rev. 3.21. Sunnyvale, CA : AMD, 2013. 352 p.

36. Katzen S. The Quintessential PIC Microcontroller. 2nd ed. London : Springer, 2005. 567 p.

37. Susnea I., Mitescu M. Microcontrollers in Practice : монографія. Berlin ; Heidelberg : Springer, 2005. 251 p.

38. Архітектура комп'ютерів. Машинні команди та програмування на асемблері : навч. посіб. / О. С. Тонкошкур, О. Б. Гниленко, Н. О. Матвєєва, О. С. Морозов. Дніпро : Нова Ідеологія, 2018. 179 с.

39. MMX Instruction Set [Електронний ресурс]. Taipei : National Taiwan University, 2026. Режим доступу: https://www.csie.ntu.edu.tw/~cyy/courses/assembly/docs/ch11_MMX.pdf (дата оновлення: 21.03.2026).

40. Streaming SIMD Extensions (SSE) // Вікіпедія. Режим доступу: https://en.wikipedia.org/wiki/Streaming_SIMD_Extensions (дата оновлення: 06.10.2025).

41. Stroustrup B. The C++ Programming Language. 4th ed. Upper Saddle River, NJ : Addison-Wesley, 2013. 1376 p.

42. Hart M. J. Windows System Programming. 4th ed. Upper Saddle River, NJ : Addison-Wesley, 2010. 609 p.

43. Visual Studio IDE documentation [Електронний ресурс]. Redmond, WA : Microsoft, [б. д.]. Режим доступу: https://learn.microsoft.com/en-us/visualstudio/ide/ (дата оновлення: 20.03.2026).

44. Silawat H., Singh S. Review of Design Challenges in Static Random-Access Memory (SRAM) Cell Circuits Based on Low Power Design (LPD) [Електронний ресурс] // International Journal of Innovative Research in Technology and Science (IJIRTS). 2023. Vol. 11, Issue 5. P. 1–6. Режим доступу: https://ijirts.org/ijirtsold/volume11issue5/IJIRTS2391.pdf.

45. Dynamic RAM // Wikipedia. Режим доступу: https://en.wikipedia.org/wiki/Dynamic_random-access_memory (дата оновлення: 01.03.2026).

46. Lin Hsiu-Min. Combining deep power-down with self-refresh mode: new opportunity for power savings in mobile DRAM [Електронний ресурс] / Paris : Surperformance SAS (MarketScreener), [б. д.]. Режим доступу: https://www.marketscreener.com/news/latest/Combining-deep-power-down-with-self-refresh-mode-new-opportunity-for-power-savings-in-mobile-DRAM-27007219/ (дата оновлення: 27.07.2018).

47. A Study of DRAM [Електронний ресурс] / V. Cuppu, B. Jacob, R. Krishnamurthy, Y. Zhang. Madison, WI : University of Wisconsin-Madison, 1999. 12 p. Режим доступу: https://pages.cs.wisc.edu/~markhill/restricted/isca99_dram.pdf.

48. Swindle J. Modern DRAM (DDR2/DDR3) Architecture [Електронний ресурс]. Richardson, TX : MindShare, Inc., 2009. Режим доступу: https://picture.iczhiku.com/resource/eetop/WHKETazWaFPFtXBx.pdf.

49. Li S., Reddy D., Jacob B. A. Performance & Power Comparison of Modern High-Speed DRAM Architectures [Електронний ресурс] / S. Li, D. Reddy, B. Jacob // Proceedings of the International Symposium on Memory Systems (MEMSYS 2018). Old Town Alexandria, VA : ACM, 2018. 13 p. Режим доступу: https://user.eng.umd.edu/~blj/papers/memsys2018-dramsim.pdf.

50. DDR4 SDRAM // Wikipedia. Режим доступу: https://en.wikipedia.org/wiki/DDR4_SDRAM (дата оновлення: 19.01.2026).

51. DDR5 SDRAM // Wikipedia. Режим доступу: https://en.wikipedia.org/wiki/DDR5_SDRAM (дата оновлення: 24.01.2026).

52. Microcircuits, Digital, 1024 Bit Schottky, Bipolar, Programmable Read-Only Memory (PROM) : MIL-M-38510/203E [Електронний ресурс]. Columbus, OH : Defense Logistics Agency, 2007. 32 p. Режим доступу: https://landandmaritimeapps.dla.mil/Downloads/MilSpec/Docs/MIL-M-38510/mil38510ss203.pdf.

53. Peters L. Electromigration Concerns Grow In Advanced Packages [Електронний ресурс] / [s.l.] : Semiconductor Engineering, 18.04.2024. Режим доступу: https://semiengineering.com/electromigration-concerns-grow-in-advanced-packages/.

54. EPROM // Вікіпедія. Режим доступу: https://uk.wikipedia.org/wiki/EPROM (дата оновлення: 30.07.2025).

55. Вичерпний посібник з технології EEPROM [Електронний ресурс]. [s.l.] : Allelco Electronics, 2024. Режим доступу: https://ua.allelcoelec.com/blog/A-Comprehensive-Guide-to-EEPROM-Technology.html.

56. FLASH-пам’ять: структура та функціонування [Електронний ресурс]. Łódź : Transfer Multisort Elektronik (TME), 2025. Режим доступу: https://www.tme.eu/ua/news/library-articles/page/68624/flash-pamiat-struktura-ta-funktsionuvannia/.

57. Disk Drives : Lecture 2 [Електронний ресурс] / University of California, San Diego. La Jolla, CA : UCSD, 2017. Режим доступу: https://cseweb.ucsd.edu/classes/sp17/cse291-a/applications/ln/lecture2.html.

58. Відаль С. Що таке SSD і як він працює? [Електронний ресурс] / [s.l.] : Tecnobits, 2023. Режим доступу: https://lnk.ua/Q2dj5xrDT.

59. З чого складається USB-флеш-накопичувач? [Електронний ресурс] / Sysdev Laboratories. [s.l.] : Sysdev Laboratories, 2022. Режим доступу: https://www.sysdevlabs.com/uk/articles/storage-devices/usb-sticks/.

60. Flash Memory for cameras [Електронний ресурс]. [s.l.] : PCHardware, 2025. Режим доступу: https://pchardware.co.uk/flashmemory.php.

61. Шевчук О. Особливості використання зовнішнього HDD… [Електронний ресурс] / MOYO, 2024. Режим доступу:

https://www.moyo.ua/ua/news/osobennosti-ispolzovaniya-vneshnego-zhestkogo-diska-6-plyusov-i-3-minusa.html

62. Зайцев В. Г., Цибаєв Є. І. Комп'ютерні системи реального часу. Київ : КПІ, 2019. 160 с.

63. Overlay (programming) // Wikipedia. Режим доступу: https://en.wikipedia.org/wiki/Overlay_(programming) (дата оновлення: 24.01.2026).

64. Вовк П. Б. Архітектура комп'ютерів. [Електронний ресурс]. Луцьк : [б. в.], 2022. Режим доступу: https://sites.google.com/view/vovkpetro/головна

65. Page Table Entries in Page Table [Електронний ресурс]. [s.l.] : GeeksforGeeks, 2025. Режим доступу: https://www.geeksforgeeks.org/operating-systems/page-table-entries-in-page-table/.

66. Silberschatz A., Galvin P. B., Gagne G. Operating System Concepts. 10th ed. Hoboken : Wiley, 2018. 976 p.

67. Gaurav S. Segmentation in Operating System [Електронний ресурс] / Scaler Topics. [s.l.] : Scaler, 2024. Режим доступу: https://www.scaler.com/topics/operating-system/segmentation-in-os/.

68. Stallings W. Operating Systems: Internals and Design Principles. 9th ed. Upper Saddle River, NJ : Pearson Education, 2018. 976 p.

69. Tanenbaum A. S., Bos H. Modern Operating Systems. 4th ed. Boston : Pearson, 2015. 1136 p.

70. Матвієнко М. П., Розен В. П., Закладний О. М. Архітектура комп'ютерів : навч. посібн. Київ : Кондор, 2020. 330 с.

71. Креневич А. П. Алгоритми і структури даних : навч. посіб. Київ : ВПЦ КНУ, 2021. 200 с.

72. Knuth D. E. The Art of Computer Programming. Vol. 4B: Combinatorial Algorithms, Part 2. Boston : Addison-Wesley, 2022. 736 p.

73. Грудзинський Ю. Є. Алгоритми та структури даних : навч. посіб. Київ : КПІ, 2022. 215 с.

74. Коваленко А. Є. Теорія інформації і кодування : навч. посіб. Київ : КПІ, 2020. 248 с.

76. Ходукін М. А. Архітектура комп'ютера: процес введення-виведення : навч. посіб. [Електронний ресурс]. Дніпро: ДУЕТ, 2020. 10 с. Режим доступу: https://www.duet.edu.ua/uploads/DocS/10st2.pdf

Навчальне видання

Ткаченко Сергій Миколайович

АРХІТЕКТУРА КОМП'ЮТЕРІВ

Навчальний посібник

Видано в авторській редакції

Електронний ресурс.
Підписано до видання 18.03.2026. Авт. арк. 7,42.

Підготовлено до видання
в Національному технічному університеті «Дніпровська політехніка».
Свідоцтво про внесення до Державного реєстру ДК № 1842 від 11.06.2004.
49005, м. Дніпро, просп. Дмитра Яворницького, 19.