МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЖИТОМИРСЬКИЙ ДЕРЖАВНИЙ ТЕХНОЛОГІЧНИЙ
УНІВЕРСИТЕТ

В.Д. ТАРАРАКА

АРХІТЕКТУРА КОМП'ЮТЕРНИХ СИСТЕМ

Навчальний посібник

За рішенням Вченої ради
ЖДТУ
протокол № 2
від 23. 03. 2018 р.

ЖДТУ
2018

УДК 681.3
Т19

Рецензенти: професор кафедри комп’ютерних інтегрованих технологій Житомирського військового інституту імені С.П. Корольова, Заслужений працівник освіти України, доктор технічних наук, професор ***Пількевич І.А.***;
завідувач кафедри комп’ютерних технологій і моделювання Житомирського національного агроекологічного університету, кандидат технічних наук, доцент ***Бродський Ю.Б.***;
завідувач кафедри метрології та інформаційно-вимірювальної техніки Житомирського державного технологічного університету, доктор технічних наук, професор ***Подчашинський Ю.А.***

Тарарака В.Д.
Т19 Архітектура комп'ютерних систем: навчальний посібник. – Житомир : ЖДТУ, 2018. – 383 с.

Навчальний посібник присвячений питанням організації та функціонування сучасних комп'ютерів та їх основних складових частин. Розглянуто структуру і функціонування комп'ютерів фон-нейманівського типу. Подана структура пам'яті комп'ютера, організація взаємодії між її рівнями, сегментна організація пам'яті та питання її захисту, принципи організації шин комп'ютера, операційних пристроїв і системи вводу/виводу. Викладено основні тенденції в архітектурі сучасних процесорів, принципи побудови і архітектурні особливості паралельних комп’ютерних систем, принципово нові архітектури комп'ютерів і особливості їх побудови. Дано поняття «нейрокомп'ютер» і приведено його архітектурні особливості.
Для студентів, які вивчають дисципліну «Архітектура комп'ютерних систем».

УДК 681. 3

ЗМІСТ

СПИСОК УМОВНИХ СКОРОЧЕНЬ

АЛП – арифметико – логічний пристрій
АСК – архітектура системи команд
БАВ – блок адресної вибірки
БПЗ – блок підсилювачив зчитування
БПФЗ – блок підсилювачів-формувачів запису
БУК – блок управління каналами
БУП – блок управління пам'яттю
БУПП – блок управління периферійними пристроями
ВВ – ввід/вивід
ВІС – велика інтегральна схема
ВПП – вузол переривань і пріоритетів
ЕОМ – електронна обчислювальна машина
ЗЕ – запам'ятовуючий елемент
ЗЗП – зовнішній запам'ятовуючий пристрій
ЗП – запам'ятовуючий пристрій
ІМС – інтегральна мікросхема
КВВ – канал вводу/виводу
КПП – контролер периферійних пристроїв
МВВ – модуль вводу/виводу
МВР – мова високого рівня
МК – мікрокоманда
МО – мікрооперація
МП – мікропроцесор
НВІС – надвелика інтегральна схема
НОЗП – надоперативний запам'ятовуючий пристрій
НЖМД – накопичувач на жорсткому магнітному диску
ОЗП – оперативний запам'ятовуючий пристрій
ОМ – обчислювальна машина
ОП – основна пам'ять
Опр – операційний пристрій
ОС – обчислювальна система
ПВВ – процесор вводу/виводу
ПДП – прямий доступ до пам'яті
ПЕ – процесорний елемент
ПЗП – постійний запам'ятовуючий пристрій
ПК – персональний комп'ютер
ПЛМ – програмована логічна матриця
ПМП – пам'ять мікропрограм
ПНШ – подвійна незалежна шина
ПП – периферійний пристрій
РгА – регістр адреси
РгАО – регістр асоціативної ознаки
РгМ – регістр маски

РгК – регістр команд
СВВ – система вводу/виводу
СУ – сигнал управління
СШ – системна шина
ФБ – функціональний блок
ФК – фіксована кома
ЦП – центральний процесор
ЧД – частковий добуток

3D-RAM – тривімірна відеопам'ять
3 D Now! – технологія обробки мультимедійної інформації
AGP – Accelerated Graphics Port – прискорений графічний порт
ATX – форм–фактор
BEDO – Burst EDO – асинхронна динамічна оперативна пам'ять з поширеним виходом і режимом пакетної передачі даних
BIOS – Basic Output System – базова система вводу/виводу
CAS – Column Address Strobe – строб адреси стовпця
CD – Compact Disk – компакт-диск
COM – Communication – послідовний порт
CD-ROM – Compact Disk Read Only Memory – компакт-диск із пам'яттю, що тільки читається
CISC – Complex Instruction Set Computer – архітектура з повним набором команд
CMOS Memory– Complementary Metal-Oxide-Semiconductor – КМОП – пам'ять
CPU – Central Processing Unit – центральний процесор
DDR SDRAM – Double Data Rate SDRAM – SDRAM – пам'ять з подвійною швидкістю обміну даними
DIMM – Dual In-Line Memory Modules – подвійний модуль ЗП
DRAM – Dynamic Random Access Memory – динамічна пам'ять з довільним доступом
DR DRAM – Direct Rambus DRAM – динамічна Rambus пам'ять
DVD – Digital Video Disk – цифровий відеодиск
EDO – Extended Data Output – асинхронна пам'ять з поширеним часом присутності даних на виході
EISA – Enhanced ISA – вдосконалена шина ISA
FIFO – First In First Out – першим прийшов – першим вийшов
Fire Wire – високошвидкісна локальна послідовна шина
Flash ROM– флеш-пам'ять
FPM – Fast Page Mode – стандартна сторінкова пам'ять
FSB – Front-Side Bus – шина «процесор–пам'ять»
ISA – Industry Standard Architecture – стандартна індустріальна архітектура (системна шина)
LIFO – Last In First Out –останнім прийшов – першим вийшов
LPT – Line Printer Terminal – паралельний порт

MCA – стандарт системної шини
MMX – multimedia txtentions – мультимедійне розширення
MPP – Massively Parallel Processing – системи з масовим паралелізмом
NLX – форм–фактор
PCI – Peripheral Component Interconnect – локальна шина для Pentium
RAR – читання після читання
RAW – читання після запису
RIMM – модуль динамічної оперативної пам'яті Rambus (Rambus – компанія по виготовленню модулів оперативної пам'яті)
RISC – Reduced Instruction Set Computer – архітектура зі скороченим набором команд
SCSI – локальна шина вводу/виводу
SDRAM – Synchronous DRAM – синхронна DRAM
SIMM – Single In-Line Memory Modules – модулі з однорядковим положенням мікросхем
SMP – Simmetric Multiprocessor – симетричні мультипроцесорні системи
TLB – Translation Look-aside Buffer – буфер асоціативної трансляції
USB – Universal Serial Bus – універсальна послідовна шина
VESA – системна шина для обміну відеоданими
VLIW – Very Long Instruction Word – архітектура процесора з довгим командним словом
WAR – запис після читання
WAW – запис після запису

ПЕРЕДМОВА

У державному освітньому стандарті вищої професійної освіти зміст дисципліни «Архітектура комп'ютерних систем» визначений у такий спосіб:

- основні характеристики, області застосування електронних обчислювальних машин (ЕОМ) різних класів;
- функціональна і структурна організація процесора;
- організація пам'яті ЕОМ;
- основні стадії виконання команди;
- організація переривань в ЕОМ;
- організація вводу/виводу;
- архітектурні особливості організації ЕОМ різних класів;
- паралельні системи;
- поняття про багатомашинні і багатопроцесорні обчислювальні системи.

Усі ці питання освітлені в даному навчальному посібнику.

Перший розділ посібника присвячений базовим положенням.

Обговорюються поняття «організація» і «архітектура» обчислювальних машин і систем, рівні абстракції, на яких ці поняття можуть бути розкриті. Прослідковується еволюція обчислювальних машин (ОМ) і обчислювальних систем (ОС) як послідовності ідей, що визначили сучасний стан в області обчислювальної техніки.

Викладається основа для розуміння принципів функціонування обчислювальних машин з класичною фон-нейманівською архітектурою в ході виконання типових команд. Розкриваються типи і структури ОМ і ОС, їх характеристики. Аналізуються тенденції подальшого розвитку архітектури ОМ і ОС з урахуванням технологічного прогресу і останніх досягнень у проектуванні обчислювальних засобів.

У другому розділі розглядаються основні види інформації, що є об'єктом обробки і зберігання в ОМ і ОС. Приводяться основні способи подання такої інформації: формати, стандарти, розміщення в пам'яті, способи доступу до даних. Подані класифікація і характеристика команд ОМ. Розглядаються основні способи адресації. Дається поняття архітектури системи команд і обговорюються різні аспекти цієї архітектури.

У третьому розділі визначені принципи і засоби, використовувані у процесі побудови систем пам'яті ОМ. Пояснюється концепція ієрархічної побудови пам'яті. Обговорюються питання організації внутрішньої пам'яті з урахуванням її реалізації на базі напівпровідникових запам'ятовуючих пристроїв (ЗП): структура пам'яті з довільним доступом, безадресна пам'ять,

матрична організація мікросхем ЗП, основні типи оперативних і постійних запам'ятовуючих пристроїв. Описуються архітектурні аспекти внутрішньої пам'яті ОМ – модульна побудова, конвеєризація, розшарування. Значна увага приділена принципам організації і функціонування кеш-пам'яті. Обговорюються питання віртуалізації пам'яті ОМ, методи і засоби захисту пам'яті від несанкціонованого доступу.

Зміст четвертого розділу – викладення питань, що стосуються архітектури процесорів обчислювальних машин.

Це опис принципів побудови і функціонування процесорів.

Обговорюються питання побудови, функціонування і проектування пристроїв управління (ПУ) з «жорсткою» логікою і ПУ з мікропрограмною організацією, а також способи прискорення їх роботи.

Розглядаються жорсткі і магістральні структури операційних пристроїв, їх організація і класифікація, способи реалізації в ОМ основних арифметичних і логічних операцій з урахуванням обробки даних у різних формах подання і форматах. Поряд зі «стандартними» способами реалізації арифметичних операцій обговорюються і такі алгоритми, використання яких веде до істотного прискорення обчислень.

Дається поняття конвеєра команд, обговорюються принципи організації такого конвеєра і проблеми, що виникають у процесі його реалізації. Особлива увага приділяється конфліктам у конвеєрі команд і способам боротьби з ними. Пояснюється концепція суперконвеєризації. Розглядається проблема семантичного розриву і способи його подолання в ОМ з архітектурами CISC і RISC. Викладаються концепції суперскалярного і мультискалярного процесорів. Розділ завершується аналізом сучасних мікропроцесорів провідних світових фірм.

П'ятий розділ відведений принципам організації системного інтерфейсу і системи комунікацій між елементами структури ОМ. Дається поняття організації шин комп'ютера, їх цільового призначення, розкриваються типи і характеристики шин. Розглядається архітектура системної плати і різні типи форм-факторів системних плат.

Розкриваються питання організації систем вводу/виводу (СВВ), способи організації вводу/виводу (програмно-керований ввід/вивід, ввід/вивід по перериваннях, прямий доступ до пам'яті) та їх вплив на еволюцію принципів побудови СВВ. Описуються особливості систем вводу/виводу великих універсальних ОМ з їх концепцією процесорів (каналів) вводу/виводу.

У шостому розділі зосереджений матеріал, присвячений питанням побудови обчислювальних систем, що реалізують концепцію розпаралелювання обчислень. Приводиться схема класифікації паралельних обчислювальних систем.

Розглядаються системи, які згідно з класифікацією Флінна можна віднести до систем класу SIMD: векторні і векторно-конвеєрні ОС, матричні ОС, асоціативні ОС, обчислювальні системи з систолічною структурою і ОС з командним словом надвеликої довжини.

Робиться огляд систем класу MIMD: симетричні мультипроцесорні системи (SMP), кластерні ОС, системи з масовим паралелізмом (MPP). Описуються системи з нетрадиційним способом управління обчислювальним процесом – потокові обчислювальні системи, розкриваються архітектурні особливості побудови нейрокомп'ютерів.

Приводиться методика оцінки ефективності обчислювальних систем.

Навчальний посібник призначений для студентів, які вивчають дисципліну «Архітектура комп'ютерних систем», а також може використовуватись студентами та магістрантами для розробки курсових, кваліфікаційних та дипломних робіт і аспірантами, які займаються проектуванням та дослідженням комп'ютеризованих систем.

1. ОСНОВНІ ПОНЯТТЯ ТА ЗАГАЛЬНА АРХІТЕКТУРА КОМП'ЮТЕРНИХ СИСТЕМ

1.1. ОСНОВНІ ПОНЯТТЯ КОМП'ЮТЕРНОЇ ТЕХНІКИ

Вивчення будь-якого питання прийнято починати з домовленостей про термінологію. Предметом нашого розгляду будуть виключно цифрові обчислювальні машини і системи, тобто пристрої, що оперують дискретними величинами. В літературі можна знайти багато різних визначень термінів «обчислювальна машина» і «обчислювальна система». Скористаємося найбільш загальним їх визначенням, домовившись, що в міру необхідності смислове їх наповнення може уточнюватися.

Обчислювальна машина (ОМ) – це сукупність технічних і програмних засобів, яка призначена для автоматизованої обробки дискретних даних за заданим алгоритмом.

Алгоритм - одне з фундаментальних понять математики і обчислювальної техніки. Міжнародна організація стандартів (ISO) формулює поняття *алгоритм* як «кінцевий набір розпоряджень, що визначає розв'язання задачі за допомогою кінцевої кількості операцій» (ISO 2382/1-84) [25].

На ранніх етапах розвитку ОМ реалізували переважно обчислювальні алгоритми, що і знайшло віддзеркалення в самому понятті «обчислювальна машина». Зберігання, обробка і комутація сигналів в ОМ в основному реалізується електронними схемами. Тому для ОМ довгий час використовувалася абревіатура ЕОМ (електронна обчислювальна машина). Цей термін з'явився у перших ЕОМ для їх відмінності від механічних і електромеханічних рахунково-вирішальних пристроїв. У зв'язку з успіхами мікроелектроніки використання в ОМ складних електронних пристроїв – надвеликих інтегральних схем (НВІС) стало загальноприйнятим і звичним. Тому слово «електронна» стосовно ОМ уже не несе істотної інформації. Крім того, ОМ містить не тільки пристрої зберігання і обробки інформації, але і пристрої вводу і виводу. Якщо пристрої зберігання і обробки інформації реалізують на базі НВІС, то пристрої вводу і виводу разом з електронними компонентами містять електромеханічні, оптоэлектронні та інші вузли. Тому в даний час переважно використовується термін «обчислювальна машина».

В міру розвитку обчислювальної техніки та інформаційних технологій клас алгоритмів, що реалізовуються, істотно розширився. З допомогою ОМ успішно вирішується широкий клас завдань «необчислювального» характеру: обробка текстів, зображень, стиснення інформації, розпізнавання образів,

інформаційно-пошукові завдання та ін. Проте термін «обчислювальна машина» зберігся.

Існують три типи ОМ: аналогові (АОМ), цифрові (ЦОМ) і гібридні (ГОМ). У АОМ для подання інформації використовуються безперервні фізичні величини, найчастіше напруга. В ЦОМ інформація подана двійковими кодами. При цьому кожен розряд приймає одне з двох значень набору {0, 1}. Для представлення двійкової змінної використовується дискретний сигнал. ГОМ – гібридні обчислювальні машини або обчислювальні машини комбінованої дії – працюють з інформацією, яка подана як у цифровій, так і в аналоговій формі; вони включають у себе переваги АОМ і ЦОМ. ГОМ доцільно використовувати для вирішення задач управління важкими швидкодіючими технічними комплексами.

ЦОМ є більш універсальним засобом обробки інформації і по ряду найбільш важливих загальнотехнічних показників перевершує інші ОМ. Тому вони набули ширшого розповсюдження. У подальшому викладі під ОМ матимемо на увазі саме ЦОМ.

В даний час у літературі разом з терміном «обчислювальна машина» часто використовується американський термін «комп'ютер», що в перекладі з англійської мови означає «обчислювач». Цим віддається дань широкому розповсюдженню нового класу обчислювальної техніки – персональним ОМ.

Обчислювальна система (ОС) – це сукупність взаємозв'язаних і взаємодіючих процесорів або обчислювальних машин, периферійного обладнання і програмного забезпечення, призначених для підготовки і вирішення завдань користувачів.

Таким чином, формальна відмінність ОС від ОМ виражається в кількості обчислювачів. Множина обчислювачів дозволяє реалізувати в ОС паралельну обробку. З іншого боку, сучасні обчислювальні машини з одним процесором також мають певні засоби розпаралелювання обчислювального процесу. Іншими словами, грань між ОМ і ОС часто буває досить розпливчатою, що дає підставу там, де це доцільно, розглядати ОМ як одну з реалізацій ОС. І навпаки, обчислювальні системи часто будуються з традиційних ОМ і процесорів, тому багато з положень, що відносяться до ОМ, можуть бути розповсюджені і на ОС.

Під ***архітектурою*** обчислювальної машини зазвичай розуміється логічна побудова ОМ, тобто те, якою машину уявляє програміст. Таке трактування називають «вузьким», і охоплює воно перелік і формат команд, форми подання даних, механізми вводу/виводу, способи адресації пам'яті і тому подібне. З розгляду випадають питання фізичної побудови обчислювальних засобів: склад пристроїв, число регістрів процесора, ємність пам'яті, наявність спеціального блока для обробки дійсних чисел, тактова частота центрального процесора і так далі. Це коло питань прийнято визначати поняттям ***організація*** або ***структурна організація***.

Архітектура (у вузькому сенсі) і організація – це дві сторони опису ОМ і ОС. Надалі користуватимемося терміном «архітектура», об'єднуючому як архітектуру у вузькому сенсі, так і організацію ОМ. Стосовно обчислювальних систем термін «архітектура» додатково розповсюджується на питання розподілу функцій між складовими ОС і взаємодії цих складових.

1.2. ЕВОЛЮЦІЯ ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Сучасний стан обчислювальної техніки (ОТ) являє собою результат багаторічної еволюції.

У традиційному трактуванні еволюцію обчислювальної техніки уявляють як послідовну зміну поколінь ОТ. Поява терміну «покоління» відноситься до 1964 року, коли фірма IBM випустила серію комп'ютерів IBM 360, назвавши цю серію «комп'ютерами третього покоління». Сам термін має різні визначення, найбільш популярними з яких є [25]:

- «Покоління обчислювальних машин – це розбиття обчислювальних машин, що склалося останнім часом, на класи, визначувані елементною базою і продуктивністю».
- «Покоління комп'ютерів – нестрога класифікація обчислювальних систем за ступенем розвитку апаратних і, останнім часом, програмних засобів».

Описуючи еволюцію ОТ, зазвичай використовують один з двох підходів: хронологічний або технологічний. У першому випадку – це хронологія подій, що істотно вплинули на становлення ОТ. Для наших цілей більший інтерес являє технологічний підхід, коли розвиток обчислювальної техніки розглядається в термінах архітектурних рішень і технологій. За словами головного конструктора фірми DEC і одного з винахідників міні-ЕОМ Бела: – «Історія комп'ютерної індустрії майже завжди рухалася технологією».

Як вузлові моменти, що визначають появу нового покоління ОТ, зазвичай вибираються революційні ідеї або технологічні прориви, що кардинально змінюють подальший розвиток засобів автоматизації обчислень. Однією з таких ідей прийнято вважати концепцію обчислювальної машини з програмою, що зберігається в пам'яті, яка була сформульована Джоном фон Нейманом. Узявши її за точку відліку, історію розвитку ОТ можна подати у вигляді трьох етапів:

- донейманівського періоду;
- ери обчислювальних машин і систем з фон-нейманівською архітектурою;
- постнейманівської епохи – епохи паралельних і розподілених обчислень, де разом з традиційним підходом все більшу роль починають грати відмінні від фон-нейманівських принципи організації обчислювального процесу.

Значно більшого поширення, проте, набула прив'язка поколінь до зміни технологій. Прийнято говорити про «механічну» еру (нульове покоління) і про п'ять наступних за нею поколінь обчислювальних систем (ОС) [27]. Перші чотири покоління традиційно пов'язують з елементною базою обчислювальних систем: електронні лампи, напівпровідникові прилади, інтегральні схеми малого ступеня інтеграції (ІМС), великі (ВІС), надвеликі (НВІС) і ультравеликі (УВІС) інтегральні мікросхеми. П'яте покоління в загальноприйнятій інтерпретації асоціюють не стільки з новою елементною базою, скільки з інтелектуальними можливостями ОС. Роботи із створення ОС п'ятого покоління велися в рамках чотирьох достатньо незалежних програм, що здійснювалися вченими США, Японії, країн Західної Європи і країн Ради економічної взаємодопомоги.

З огляду на те, що жодна з програм не привела до очікуваних результатів, розмови про ОС п'ятого покоління помалу утихають. Трактування п'ятого покоління явно випадає з «технологічного» принципу. З іншого боку, зарахування всіх ОС на базі надвеликих інтегральних схем (НВІС) до четвертого покоління не відбиває принципових змін в архітектурі ОС, що здійснилися за останні роки. Щоб в якійсь мірі прослідкувати роль таких змін, скористаємося дещо відмінним трактуванням, яке виділяє шість поколінь ОС. Спробуємо стисло охарактеризувати кожне з них, виділяючи найбільш значущі події [25].

Нульове покоління (1492−1945)

Для повноти картини згадаємо дві події, що стались до нашої ери: перші рахівниці−абак, винайдені в стародавньому Вавілоні за 3000 років до н. э., і їх «сучасніший» варіант з кісточками на дроті, що з'явився в Китаї приблизно за 500 років також до н.е.

«Механічна» ера (нульове покоління) в еволюції ОТ пов'язана з механічними, а пізніше – електромеханічними обчислювальними пристроями. Основним елементом механічних пристроїв було зубчате колесо. Починаючи з XX століття роль базового елемента переходить до електромеханічного реле. Не зменшуючи значення багатьох ідей «механічної» ери, необхідно зазначити, що жодне із створених пристроїв не можна з повною підставою назвати обчислювальною машиною в сучасному її розумінні. Щоб підкреслити це, замість терміну «обчислювальна машина» використовуватимемо такі слова, як «обчислювач», «калькулятор» і тому подібне.

Хронологія основних подій «механічної» ери виглядає таким чином.

1492 рік. У одному зі своїх щоденників Леонардо да Вінчі приводить малюнок тринадцятирозрядного десяткового підсумовуючого пристрою на основі зубчатих коліс.

1623 рік. Вільгельм Шиккард (Wilhelm Schickard, 1592–1635), професор університету Тюбінгена, розробляє пристрій на основі зубчатих коліс («рахуючий годинник») для складання і віднімання шестирозрядних десяткових чисел. Чи був пристрій реалізований за життя винахідника, достовірно невідомо, але в 1960 році він був відтворений і проявив себе цілком працездатним.

1642 рік. Блез Паскаль (Blaise Pascal, 1623–1663) представляє «Паськалін» – перший реально здійснений механічний цифровий обчислювальний пристрій, що здобув популярність. Прототип пристрою підсумовував і віднімав п'ятирозрядні десяткові числа. **1673 рік.** Готфрід Вільгельм Лейбніц (Gottfried Wilhelm Leibniz, 1646–1716) створює «покроковий обчислювач» – десятковий пристрій для виконання всіх чотирьох арифметичних операцій над 12-розрядними десятковими числами. Результат множення представлявся 16 цифрами. Крім зубчатих коліс в пристрої використовувався новий елемент – ступінчастий валик.

1786 рік. Німецький військовий інженер Іоганн Мюллер (Johann Mueller, 1746–1830) висуває ідею «різницевої машини» – спеціалізованого калькулятора для табулювання логарифмів, що обчислюються різницевим методом. Калькулятор, побудований на ступінчастих валиках Лейбніца, вийшов достатньо невеликим (13 см у висоту і 30 см в діаметрі), але при цьому міг виконувати всі чотири арифметичні дії над 14-розрядними числами.

1801 рік. Жозеф Марія Жаккард (Joseph Marie Jacquard, 1752–1834) будує ткацький верстат з програмним управлінням, програма роботи якого задається за допомогою комплекту перфокарт.

1832 рік. Англійський математик Чарльз Беббідж (Charles Babbage, 1792–1871) створює сегмент різницевої машини, що оперує шестирозрядними числами і різницями другого порядку.

1834 рік. Пер Георг Шутц (Per George Scheutz, 1785–1873) із Стокгольма, використовуючи короткий опис проекту Беббіджа, створює з дерева невелику різницеву машину.

1836 рік. Беббідж розробляє проект «аналітичної машини». Проект передбачає три зчитувачі з перфокарт для введення програм і даних, пам'ять (по Беббіджу – «склад») на п'ятдесят 40-розрядних чисел, два акумулятори для зберігання проміжних результатів. У програмуванні машини передбачена концепція умовного переходу. У проект закладений також і прообраз мікропрограмування – зміст інструкцій передбачалося задавати шляхом позиціонування металевих штирів у циліндрі з отворами. За оцінками автора, підсумовування повинне було займати 3с, а множення і ділення – 2-4 хв.

1843 рік. Георг Шутц спільно з сином Едвардом (Edvard Scheutz, 1821 – 1881) створюють різницеву машину з принтером для роботи з різницями третього порядку.

1871 рік. Беббідж створює прототип одного з пристроїв своєї аналітичної машини – «млин» (так він охрестив те, що зараз прийнято називати центральним процесором), а також принтер.

1885 рік. Дорр Фельт (Dorr E. Felt, 1862–1930) з Чікаго будує свій «комптометр» – перший калькулятор, де числа вводяться натисненням клавіш.

1890 рік. Результати перепису населення в США обробляються за допомогою перфокарточного табулятора, створеного Германом Холлерітом (Herman Hollerith, 1860–1929) з Массачусетського технологічного інституту.

1892 рік. Вільям Барроуз (William S. Burroughs, 1857–1898) пропонує пристрій, схожий з калькулятором Фельта, але надійніший, і від цієї події бере старт індустрія офісних калькуляторів.

1937 рік. Джорж Стібітц (George Stibitz, 1904–1995) з Bell Telephone Laboratories демонструє перший однобітовий двійковий обчислювач на базі електромеханічних реле.

1937 рік. Алан Тьюрінг (Alan M. Turing, 1912–1954) з Кембріджського університету публікує статтю, в якій висловлює концепцію теоретичної спрощеної обчислювальної машини, що надалі отримала назву машини Тюрінга.

1938 рік. Клод Шеннон (Claude E. Shannon, 1916–2001) публікує статтю про реалізацію символічної логіки на базі реле.

1938 рік. Німецький інженер Конрад Цузе (Konrad Zuse, 1910–1995) будує механічний програмований обчислювач Z1 з пам'яттю на 1000 біт. Останнім часом Z1 все частіше називають першим у світі комп'ютером.

1939 рік. Джордж Стібітц і Семюель Вільямс (Samuel Williams, 1911–1977) представили Model I – калькулятор на базі релейної логіки, керований за допомогою модифікованого телетайпу, що дозволило підключатися до калькулятора по телефонній лінії. Пізніші модифікації допускали також певний ступінь програмування.

1940 рік. Наступна робота Цузе – електромеханічна машина Z2, основу якої складала релейна логіка, хоча пам'ять, як і в Z1, була механічною.

1941 рік. Цузе створює електромеханічний програмований обчислювач Z3. Обчислювач містить 2600 електромеханічних реле. Z3 – це перша спроба реалізації принципу програмного управління, хоча і не в повному об'ємі (у загальноприйнятому розумінні цей принцип ще не був сформульований). Зокрема, не передбачалася можливість умовного переходу. Програма зберігалася на перфострічці. Ємність пам'яті складала 64 22-бітових слова. Операція множення займала 3-5 с.

1943 рік. Група вчених Гарвардського університету на чолі з Говардом Айкеном (Howard Aiken, 1900 – 1973) розробляє обчислювач ASCC Mark I (Automatic Sequence-Controlled Calculator Mark I) - перший програмно керований обчислювач, що здобув широку популярність. Довжина пристрою склала 18м, а

вага - 5т. Машина складалася з багатьох обчислювачів, які обробляли свої частини загального завдання під управлінням єдиного пристрою управління. Команди зчитувалися з паперової перфострічки і виконувалися в порядку зчитування. Дані зчитувалися з перфокарт. Обчислювач обробляв 23-розрядні числа, при цьому додавання займало 0,3с, множення – 4с, а ділення – 10с.

1945 рік. Цузе завершує Z4 - покращену версію обчислювача Z3. По архітектурі у Z4 дуже багато загальних рис з сучасними ЕОМ: пам'ять і процесор представлені окремими пристроями, процесор може обробляти числа з плаваючою комою і, на додаток до чотирьох основних арифметичних операцій, здатний обчислювати квадратний корінь. Програма зберігається на перфострічці і прочитується послідовно.

Не зменшуючи важливості кожного з перерахованих фактів, як найважливіший момент «механічної» епохи все-таки виділимо аналітичну машину Чарльза Беббіджа і пов'язані з нею ідеї.

Перше покоління (1937–1953)

На роль першої в історії електронної обчислювальної машини в різні періоди претендувало декілька розробок. Загальним у них було використання схем на базі електронно–вакуумных ламп замість електромеханічних реле. Передбачалося, що електронні ключі будуть значно надійніші, оскільки в них відсутні рухомі частини, проте технологія того часу була настільки недосконалою, що за надійністю електронні лампи виявилися ненабагато кращими, ніж реле. Проте у електронних компонентів була одна важлива перевага: виконані на них ключі могли перемикатися приблизно в тисячу разів швидше за свої електромеханічні аналоги.

Першою електронною обчислювальною машиною (ЕОМ) найчастіше називають спеціалізований калькулятор ABC (Atanasoff-Berry Computer). Розроблений він був в період з 1939 по 1942 рік професором Джоном Атанасовим (John V. Atanasoff, 1903–1995) спільно з аспірантом Кліффордом Беррі (Clifford Berry, 1918–1963) і призначався для вирішення системи лінійних рівнянь (до 29 рівнянь з 29 змінними). ABC володів пам'яттю на 50 слів довжиною 50 біт, а елементами, що запам'ятовують, служили конденсатори з ланцюгами регенерації. Як вторинна пам'ять використовувалися перфокарти, де отвори не перфорувалися, а пропалювалися. ABC почав вважатися першою ЕОМ, після того, як судовим рішенням були анульовані патенти творців іншого електронного калькулятора – ENIAC. Необхідно все ж таки відзначити, що ні ABC, ні ENIAC не є обчислювальними машинами в сучасному розумінні цього терміну і їх правильніше класифікувати як калькулятори.

Другим претендентом на першість вважається обчислювач Colossus, побудований в 1943 році в Англії в містечку Bletchley Park поблизу Кембріджа. Винахідником машини був професор Макс Ньюмен (Max Newman, 1987–1984),

а виготовив його Томмі Флауерс (Tommy Flowers, 1905–1998). Colossus був створений для розшифровки кодів німецької шифрувальної машини «Лоренц Шлюссельцузат–40». До складу команди розробників входив також Алан Тьюрінг. Машина була виконана у вигляді восьми стійок заввишки 2,3 м, а загальна довжина її складала 5,5 м. У логічних схемах машини і в системі оптичного зчитування інформації використовувалося 2400 електронних ламп. Інформація зчитувалася з п'яти довгих паперових кілець, що оберталися із швидкістю 5000 символів/с.

Нарешті, третій кандидат на роль першої електронної обчислювальної машини – вже згадуваний програмований електронний калькулятор загального призначення ENIAC (Electronic Numerical Integrator and Computer – електронний цифровий інтегратор і обчислювач). Ідея калькулятора, висунута в 1942 році Джоном Мочлі (John J. Mauchly, 1907–1980) з університету Пенсільванії, була реалізована ним спільно з Преспером Еккертом (J. Presper Eckert, 1919–1995) в 1946 році. Із самого початку ENIAC активно використовувався в програмі розробки водневої бомби. Машина експлуатувалася до 1955 року і застосовувалася для генерування випадкових чисел, прогнозу погоди і проектування аеродинамічних труб. ENIAC важив 30 тон, містив 18 000 радіоламп, мав розміри 2,5 х 30 м і забезпечував виконання 5000 додавань і 360 множень в секунду. Використовувалася десяткова система числення. Програма задавалася схемою комутації тригерів на 40 набірних полях. Коли всі лампи працювали, інженерний персонал міг налаштовувати ENIAC на нове завдання, вручну змінивши підключення 6000 проводів. Під час пробної експлуатації з'ясувалося, що надійність машини надзвичайно низька – пошук несправностей займав від декількох годин до декількох діб. За своєю структурою ENIAC нагадував механічні обчислювальні машини. 10 тригерів з'єднувалися в кільце, утворюючи десятковий лічильник, який виконував роль рахункового колеса механічної машини. Десять таких кілець плюс два тригери для подання знака числа представляли регістр, що запам'ятовує. Всього в ENIAC було 20 таких регістрів.

При всій важливості кожної з трьох розглянутих розробок основна подія, яка виникла в цей період, пов'язана з ім'ям Джона фон Неймана. Американський математик Джон фон Нейман (John von Neumann, 1903–1957) взяв участь в проекті ENIAC як консультант. Ще до завершення ENIAC Еккерт, Мочлі і фон Нейман приступили до нового проекту – EDVAC, головною особливістю якого стала ідея *програми, що зберігалася в пам'яті*.

Технологія програмування в даний період була ще на зачатковому рівні. Перші програми складалися в машинних кодах – числах, безпосередньо записуваних у пам'ять ЕОМ. Лише у 50-х роках почалося використання мови асемблера, що дозволяло замість числового запису команд використовувати

символьну їх нотацію, після чого спеціальною програмою, також званою асемблером, ці символьні позначення транслювалися у відповідні коди.

Незважаючи на свою примітивність, машини першого покоління виявилися досить корисними для інженерних цілей і в прикладних науках. Так, Атанасофф підрахував, що вирішення системи з восьми рівнянь з вісьма змінними за допомогою популярного тоді електромеханічного калькулятора Маршана зайняло б вісім годин. У разі ж 29 рівнянь з 29 змінними, з якими калькулятор ABC справлявся менш ніж за годину, пристрій з калькулятором Маршана витратив би 381 годину. З першим завданням в рамках проекту водневої бомби ENIAC справився за 20 секунд, на противагу 40 годинам, які знадобилися б у випадку використання механічних калькуляторів.

В 1947 році під керівництвом С. А. Лебедєва початі роботи із створення малої електронної рахункової машини (МЕСМ). Ця ЕОМ була запущена в експлуатацію в 1951 році і стала першою ЕОМ в СРСР і континентальній Європі.

В 1952 році Еккерт і Мочлі створили першу комерційно успішну машину UNIVAC. Саме за допомогою цієї ЕОМ було передбачено, що Ейзенхауер у результаті президентських виборів переможе Стівенсона з розривом у 438 голосів (фактичний розрив склав 442 голоси).

Також у 1952 році в дослідну експлуатацію була запущена обчислювальна машина М-1 (І. С. Брук, Н. Я. Матюхін, А. Б. Залкінд). М-1 вміщала 730 електронних ламп, оперативну пам'ять ємністю 256 25-розрядних слів, рулонний телетайп і володіла продуктивністю 15-20 операцій/с. Вперше була застосована двоадресна система команд. Трохи пізніше групою випускників МЕІ під керівництвом І. С. Брука створена машина М-2 з ємністю оперативної пам'яті 512 34-розрядних слів і швидкодією 2000 операцій/с.

У квітні 1953 року в експлуатацію поступила сама швидкодіюча в Європі ЕОМ БЕСМ (С. А. Лебєдєв). Її швидкодія склала 8000-10000 операцій/с. Приблизно в той же час випущена лампова ЕОМ «Стріла» (Ю. А. Базилевський, Б. І. Рамеєв) з швидкодією 2000 операцій/с.

Друге покоління (1954–1962)

Друге покоління характеризується рядом досягнень в елементній базі, структурі і програмному забезпеченні. Прийнято вважати, що приводом для виділення нового покоління ЕОМ стали технологічні зміни і, головним чином, перехід від електронних ламп до напівпровідникових діодів і транзисторів з часом перемикання близько 0,3 мс.

Першою ЕОМ, що була виконана повністю на напівпровідникових діодах і транзисторах, стала TRADIC (TRAnisitor Digital Computer), побудована в Bell Labs за замовленням військово-повітряних сил США як прототип бортової ЕОМ. Машина складалася з 700 транзисторів і 10000 германієвих діодів. За два

роки експлуатації TRADIC відмовили тільки 17 напівпровідникових елементів, що говорить про прорив в області надійності, в порівнянні з машинами на електронних лампах. Іншою повністю напівпровідниковою ЕОМ стала TX-0, створена в 1957 році в Массачусетському технологічному інституті.

З другим поколінням ЕОМ асоціюють ще одне принципове технологічне удосконалення – перехід від пристроїв пам'яті на базі ртутних ліній затримки до пристроїв на магнітних комірках. У запам'ятовуючих пристроях (ЗП) на лініях затримки дані зберігалися у вигляді акустичної хвилі, безперервно циркулюючої по кільцю з ліній затримки, а доступ до елемента даних ставав можливим лише у момент проходження відповідної ділянки хвилі поблизу пристрою зчитування/запису. Головною перевагою ЗП на магнітних комірках став вільний доступ до даних, коли у будь-який момент доступний будь-який елемент даних, причому час доступу не залежить від того, який це елемент.

Технологічний прогрес доповнюють важливі зміни в архітектурі ЕОМ. Перш за все, це стосується появи у складі процесора ЕОМ індексних регістрів, що дозволило спростити доступ до елементів масивів. Раніше, під час циклічної обробки елементів масиву, необхідно було модифікувати код команди, тобто адресу елемента масиву, що зберігається в ньому. Як наслідок, у ході обчислень коди деяких команд постійно змінювалися, що затрудняло відлагодження програми. З використанням індексних регістрів адреса елемента масиву обчислюється як сума адресної частини команди і вмісту індексного регістра. Це дозволяє звернутися до будь-якого елемента масиву, не зачіпаючи код команди, а лише модифікуючи вміст індексного регістра.

Другою принциповою зміною в структурі ЕОМ стало додавання апаратного блока обробки чисел у форматі з плаваючою комою. До цього обробка дійсних чисел проводилася за допомогою підпрограм, кожна з яких імітувала виконання якоїсь одної операції з плаваючою комою, використовуючи для цієї мети звичайний цілочисельний арифметико-логічний пристрій.

Третє значуще нововведення в архітектурі ЕОМ – поява у складі обчислювальної машини процесорів вводу/виводу, що дозволяють звільнити центральний процесор від рутинних операцій по управлінню вводом/виводом і що забезпечують вищу пропускну спроможність тракту «пам'ять – пристрої вводу/виводу (ПВВ)».

До другого покоління відносяться і дві перші супер-ЕОМ, розроблені для прискорення чисельних обчислень у наукових застосуваннях. Термін «супер-ЕОМ» спочатку застосовувався по відношенню до ЕОМ, продуктивність яких на один або більше порядків перевишувала таку для інших обчислювальних машин того ж покоління. У другому поколінні цьому визначенню відповідали дві ЕОМ (правильніше сказати системи): LARC (Livermore Atomic Research Computer) і IBM 7030. Крім іншого, в цих ЕОМ знайшли втілення ще дві

новинки: поєднання операцій процесора із зверненням до пам'яті і прості форми паралельної обробки даних.

Помітною подією даного періоду стала поява в 1958 році машини М-20. У цій ЕОМ, зокрема, були реалізовані: часткове поєднання операцій, апаратні засоби підтримки програмних циклів, можливість паралельної роботи процесора і пристрою виводу. Оперативна пам'ять ємністю 4096 45-розрядних слів була виконана на магнітних комірках.

Шестидесяті роки XX століття стали періодом бурхливого розвитку обчислювальної техніки в СРСР. За цей період розроблені і запущені у виробництво обчислювальні машини «Урал-1», «Урал-4», «Урал-11», «Урал-14», БЕСМ-2, М-40, «Мінськ-1», «Мінськ-2», «Мінськ-22», «Мінськ-32». У 1960 році під керівництвом В. М. Глушкова і Б. Н. Малиновського розроблена перша напівпровідникова управляюча машина «Дніпро».

Нарешті, не можна не відзначити значні події у сфері програмного забезпечення, а саме створення мов програмування високого рівня: Фортран (1956), Алгол (1958) і Кобол (1959).

Третє покоління (1963–1972)

Третє покоління ознаменувалося різким збільшенням обчислювальної потужності ЕОМ, що стало результатом великих успіхів в області архітектури, технології і програмного забезпечення. Основні технологічні досягнення зв’язані з переходом від дискретних напівпровідникових елементів до інтегральних мікросхем і початком застосування напівпровідникових запам'ятовуючих пристроїв, що почали витісняти ЗП на магнітних комірках. Істотні зміни відбулися і в архітектурі ЕОМ. Це, перш за все, мікропрограмування як ефективна техніка побудови пристроїв управління складних процесорів а також настання ери конвеєризації і паралельної обробки. В області програмного забезпечення визначальними віхами стали перші операційні системи і реалізація режиму розділення часу.

В перших ЕОМ третього покоління використовувалися інтегральні схеми з малим ступенем інтеграції (small-scale integrated circuits, SSI), де на одному кристалі розміщується близько 10 транзисторів. Ближче до кінця даного періоду на зміну SSI почали приходити інтегральні схеми середнього ступеня інтеграції (medium-scale integrated circuits, MSI), в яких число транзисторів на кристалі збільшилося на порядок. Все частіше застосовуються переваги паралельної обробки, що реалізовуються за рахунок множинних функціональних блоків, поєднання в часі роботи центрального процесора і операцій вводу/виводу, конвеєризації потоків команд і даних.

В 1964 році Сеймур Крей (Seymour Cray, 1925–1996) побудував обчислювальну систему CDC 6600, в архітектуру якої вперше був закладений функціональний паралелізм. Завдяки наявності 10 незалежних функціональних

блоків, здатних працювати паралельно, і 32 незалежних модулів пам'яті вдалося досягти швидкодії в 1 MFLOPS (мільйон операцій з плаваючою комою в секунду). П'ятьма роками пізніше Крей створив CDC 7600 з конвейєризованими функціональними блоками і швидкодією 10 MFLOPS. CDC 7600 називають першою конвеєрною обчислювальною системою (конвеєрним процесором). Революційною віхою в історії ОТ стало створення сімейства обчислювальних машин IBM 360, архітектура і програмне забезпечення яких на довгі роки служили еталоном для подальших великих універсальних ЕОМ (mainframes). У машинах цього сімейства знайшли втілення багато нових для того періоду ідей, зокрема: попередня вибірка команд, окремі блоки для операцій з фіксованою і плаваючою комою, конвеєризація команд, кеш-пам'ять. До третього покоління ОС відносяться також перші паралельні обчислювальні системи: SOLOMON корпорації Wes-tinghause і ILLIAC IV - сумісна розробка університету Ілінойса і компанії Burroughs. Третє покоління ОТ ознаменувалося також появою перших конвеєрно-векторних ОС: TI-ASC (Texas Instruments Advanced Scientific Computer) і STAR-100 фірми CBC.

Серед обчислювальних машин, розроблених у цей період в СРСР, перш за все необхідно відзначити «швидкодіючу електронно-рахункову машину» – БЕСМ-6 (С. А. Лебєдєв) з продуктивністю 1 млн. операцій/с. Продовженням лінії М-20 стали М-220 і М-222 з продуктивністю до 200000 операцій/с. Оригінальна ЕОМ для інженерних розрахунків «Мир-1» була створена під керівництвом В. М. Глушкова. Як вхідна мова цієї ЕОМ використана мова програмування високого рівня Аналітик.

У сфері програмного забезпечення необхідно відзначити створення в 1970 році Кеном Томпсоном (Kenneth Thompson) з Bell Labs мови B, прямого попередника популярної мови програмування C, і поява ранньої версії операційної системи UNIX.

Четверте покоління (1972–1984)

Відлік четвертого покоління зазвичай ведуть з переходу на інтегральні мікросхеми великого (large-scale integration, LSI) і надвеликого (very large-scale integration, VLSI) ступеня інтеграції. До перших відносять схеми, що містять близько 1000 транзисторів на кристалі, тоді як число транзисторів на одному кристалі VLSI має порядок 100000. За таких рівнів інтеграції стало можливим умістити в одну мікросхему не тільки центральний процесор, але і обчислювальну машину (ЦП, основну пам'ять і систему вводу/виводу).

Кінець 70-х і початок 80-х років – це час становлення і подальшого переможного ходу мікропроцесорів і мікро-ЕОМ, що, проте, не знижує важливості змін, які відбулися в архітектурі інших типів обчислювальних машин і систем.

Однією з найбільш значущих подій в області архітектури ЕОМ стала ідея обчислювальної машини зі скороченим набором команд (RISC, Redused Instruction Set Computer), висунута в 1975 році і вперше реалізована в 1980 році. У спрощеному викладі суть концепції RISC полягає у приведенні набору команд ЕОМ до простих команд, о найчастіше використовуються. Це дозволяє спростити схемотехніку процесора і добитися різкого скорочення часу виконання кожній з «простих» команд. Складніші команди реалізуються як підпрограми, складені з швидких «простих» команд.

У ЕОМ і ОС четвертого покоління практично не використовуються ЗП на магнітних комірках і основна пам'ять будується з напівпровідникових запам'ятовуючих пристроїв (ЗП). До цього використання напівпровідникових ЗП обмежувалося лише регістрами і кеш-пам'яттю.

У сфері високопродуктивних обчислень домінують векторні обчислювальні системи, більш відомі як супер-ЕОМ. Розробляється нова паралельна архітектура, проте подібні роботи поки що носять експериментальний характер. На заміну великим ЕОМ, що працюють у режимі розділення часу, приходять індивідуальні мікро-ЕОМ і робочі станції (цим терміном позначають мережний комп'ютер, що використовує ресурси сервера).

В області програмного забезпечення виділимо появу мов програмування надвисокого рівня, таких як FP (functional programming - функціональне програмування) і Пролог (Prolog, programming in logic). Ці мови орієнтовані на *декларативний стиль програмування*, на відміну від Паскаля, С, Фортрана і так далі – мов *імперативного стилю програмування*. У випадку декларативного стилю програміст дає математичний опис того, що повинне бути обчислене, а деталі того, яким чином це повинно бути зроблено, покладаються на компілятор і операційну систему. Такі мови поки використовуються недостатньо широко, але виглядають багатообіцяючими для ОС з масовим паралелізмом, які включають більше 1000 процесорів.

Дві події в області програмного забезпечення пов'язані з Кеном Томпсоном (Kenneth Thompson) і Деннісом Рітчи (Dennis Ritchie) з Bell Labs. Це створення мови програмування С і її використання під час написання операційної системи UNIX для машини DEC PDP-11. Така форма написання операційної системи дозволила швидко розповсюдити UNIX на багато ЕОМ.

П'яте покоління (1984–1990)

Головним приводом для виділення обчислювальних систем другої половини 80-х років у самостійне покоління став стрімкий розвиток ОС з сотнями процесорів, що стало спонукальним мотивом для прогресу в області паралельних обчислень. Раніше паралелізм обчислень виражався лише у вигляді конвеєризації, векторної обробки і розподілу роботи між невеликим числом процесорів. Обчислювальні системи п'ятого покоління забезпечують

такий розподіл завдань по множині процесорів, при якому кожен з процесорів може виконувати завдання окремого користувача.

В рамках п'ятого покоління в архітектурі обчислювальних систем сформувалися два принципово різних підходи: архітектура із спільно використовуваною пам'яттю і архітектура з розподіленою пам'яттю.

Характерним прикладом першого підходу може служити система Sequent Balance 8000, в якій є велика основна пам'ять, що розділяється 20 процесорами. Крім цього, кожен процесор оснащений власною кеш-пам'яттю. Кожен з процесорів може виконувати задачу свого користувача, але при цьому у складі програмного забезпечення є бібліотека підпрограм, що дозволяє програмістові залучати для вирішення своєї задачі більше одного процесора.

Другий напрям розвитку систем п'ятого покоління – системи з розподіленою пам'яттю, де кожен процесор володіє своїм модулем пам'яті, а зв'язок між процесорами забезпечується мережею взаємозв'язків. Прикладом такої ОС може служити система iPSC-1 фірми Intel, відоміша як «гіперкуб». Максимальний варіант системи включав 128 процесорів. Застосування розподіленої пам'яті дозволило усунути обмеження в пропускній спроможності тракту «процесор-пам'ять», але потенційним «вузьким місцем» тут стає мережа взаємозв'язків.

Нарешті, третій напрям в архітектурі обчислювальних систем п'ятого покоління – це ОС, в яких декілька тисяч достатньо простих процесорів працюють під управлінням єдиного пристрою управління і одночасно проводять одну і ту ж операцію, але кожен над своїми даними. До цього класу можна віднести Connection Machine фірми Thinking Machines Inc. і MP-1 фірми MasPar Inc.

У наукових обчисленнях, як і раніше, провідну роль грають векторні супер-ЕОМ. Багато виробників пропонують ефективніші варіанти з декількома векторними процесорами, але число таких процесорів зазвичай невелике (від 2 до 8).

RISC-архітектура виходить із стадії експериментів і стає базовою архітектурою для робочих станцій (workstations).

Знаковою прикметою даного періоду став стрімкий розвиток технологій глобальних і локальних комп'ютерних мереж. Це стимулювало зміни в технології роботи індивідуальних користувачів. На противагу могутнім універсальним ОС, що працюють у режимі розділення часу, користувачі все більше віддають перевагу підключеним до мережі індивідуальним робочим станціям. Такий підхід дозволяє для вирішення невеликих завдань задіювати індивідуальну машину, а у разі необхідності великої обчислювальної потужності звернутися до ресурсів приєднаних до тієї ж мережі могутніх файл-серверів або супер-ЕОМ.

Шосте покоління (1990 –)

На ранніх стадіях еволюції обчислювальних засобів зміна поколінь асоціювалася з революційними технологічними проривами. Кожне з перших чотирьох поколінь мало чітко виражені відмінні ознаки і цілком певні хронологічні рамки. Подальше ділення на покоління вже не так очевидно і може бути зрозуміле лише у разі ретроспективного погляду на розвиток обчислювальної техніки. П'яте і шосте покоління в еволюції ОТ – це віддзеркалення нової якості, що виникла в результаті послідовного накопичення приватних досягнень, головним чином в архітектурі обчислювальних систем і, в дещо меншій мірі, у сфері технологій.

Приводом для початку відліку нового покоління стали значні успіхи в області паралельних обчислень, пов'язані з широким розповсюдженням обчислювальних систем з масовим паралелізмом. Особливість організації таких систем, що позначаються абревіатурою MPP (massively parallel processing), полягає в сукупності великої кількості (до декількох тисяч) взаємодіючих, але достатньо автономних обчислювальних машин. За обчислювальною потужністю такі системи вже успішно конкурують з супер-ЕОМ, які, як раніше наголошувалося, за своєю суттю є векторними ОС. Поява обчислювальних систем з масовим паралелізмом дала підставу говорити про продуктивність, вимірювану в TFLOPS (1 TFLOPS відповідає 10^{12} операціям з плаваючою комою в секунду).

Друга характерна риса шостого покоління – різко збільшений рівень робочих станцій. У процесорах нових робочих станцій успішно поєднуються RISC-архітектура, конвеєризація і паралельна обробка. Деякі робочі станції за продуктивністю співставні з супер-ЕОМ четвертого покоління. Вражаючі характеристики робочих станцій породили інтерес до гетерогенних (неоднорідних) обчислень, коли програма, що запущена на одній робочій станції, може знайти в локальній мережі не зайняті в даний момент інші станції, після чого обчислення розпаралелюються і на ці простоюючі станції.

Нарешті, третьою прикметою шостого покоління в еволюції ОТ стає вибухове зростання глобальних мереж.

Завершуючи обговорення еволюції ОТ, відзначимо, що верхня межа шостого покоління хронологічно поки не визначена і подальший розвиток обчислювальної техніки може внести до його характеристики нові корективи.

1.3. КОНЦЕПЦІЯ ОБЧИСЛЮВАЛЬНИХ МАШИН З ПРОГРАМОЮ, ЩО ЗБЕРІГАЄТЬСЯ В ПАМ'ЯТІ

В основі архітектури сучасних ОМ лежить представлення алгоритму розв'язання задачі у вигляді програми послідовних обчислень. Згідно зі

стандартом ISO 2382/1-84, програма для ОМ – це «впорядкована послідовність команд, що підлягає обробці».

ОМ, де певним чином закодовані команди програми зберігаються в пам'яті, відома під назвою *обчислювальної машини з програмою, що зберігається в пам'яті*. Ідея належить творцям обчислювача ENIAC Еккерту, Мочлі і фон Нейману. Ще до завершення робіт над ENIAC вони приступили до нового проекту – EDVAC, головною особливістю якого стала концепція програми, що зберігалася в пам'яті. Ця концепція на довгі роки визначила базові принципи побудови подальших поколінь обчислювальних машин. Щодо авторства існує декілька версій, але оскільки в закінченому вигляді ідея вперше була викладена в 1945 році в статті фон Неймана, саме його прізвище фігурує в позначенні архітектури подібних машин, які складають переважну частину сучасного парку ОМ і ОС.

Суть фон-нейманівської концепції обчислювальної машини можна звести до чотирьох принципів: *двійкового кодування; програмного управління; однорідності пам'яті; адресності.*

Принцип двійкового кодування. Згідно з цим принципом, вся інформація, як дані, так і команди, кодуються двійковими цифрами 0 і 1. Кожен тип інформації подається двійковою послідовністю і має свій *формат*. Послідовність бітів у форматі, що має певний сенс, називається *полем*. У числовій інформації зазвичай виділяють *поле знака* і *поле значущих розрядів*. У форматі команди можна виділити два поля (рис. 1.1): *поле коду операції* (КОп) і *поле адрес* (адресну частину – АЧ).

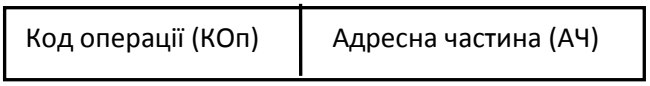


Рис. 1.1. Структура команди

Кодом операції є вказівка, яка операція повинна бути виконана, і задається за допомогою *r*-розрядної двійкової комбінації.

Вид адресної частини і число складових її адрес залежать від типу команди: в командах перетворення даних АЧ містить адреси об'єктів обробки (*операндів*) і результату; в командах зміни порядку обчислень – адреса наступної команди програми; в командах вводу/виводу – номер пристрою вводу/виводу. Адресна частина також подається двійковою послідовністю, довжину якої позначимо через *p.* Таким чином, команда в обчислювальній машині має вид ($r + p$) – розрядної двійкової комбінації.

Принцип програмного управління. Всі обчислення, передбачені алгоритмом розв'язання задачі, повинні бути подані у вигляді *програми*, що складається з послідовності управляючих слів – *команд*. Кожна команда передбачає деяку операцію з набору операцій, що реалізовуються обчислювальною машиною. Команди програми зберігаються в послідовних комірках

пам'яті обчислювальної машини і виконуються *в природній послідовності*, тобто в порядку їх розміщення в програмі. За необхідності, за допомогою спеціальних команд, ця послідовність може бути змінена. Рішення про зміну порядку виконання команд програми ухвалюється або на підставі аналізу результатів попередніх обчислень, або безумовно.

Принцип однорідності пам'яті. Команди і дані зберігаються в одній і тій же пам'яті і зовні не відрізняються. Розпізнати їх можна тільки за способом використання. Це дозволяє проводити над командами ті ж операції, що і над числами, і, відповідно, відкриває ряд можливостей. Так, циклічно змінюючи адресну частину команди, можна забезпечити звернення до послідовних елементів масиву даних. Такий прийом носить назву *модифікації команд* і з позицій сучасного програмування не є оптимальним. Більш корисним є інший наслідок принципу однорідності, коли команди однієї програми можуть бути отримані як результат виконання іншої програми. Ця можливість лежить в основі *трансляції* – перекладу тексту програми з мови високого рівня на мову конкретної ОМ.

Концепція обчислювальної машини, викладена у статті фон Неймана, припускає єдину пам'ять для зберігання команд і даних. Такий підхід був прийнятий в обчислювальних машинах, що створювалися в Прінстонському університеті, через що і отримав назву *прінстонської архітектури*. Практично одночасно в Гарвардському університеті запропонували іншу модель, в якій ОМ мала окрему пам'ять команд і окрему пам'ять даних. Цей вид архітектури називають *гарвардською архітектурою*. Довгі роки переважаючою була і залишається прінстонська архітектура, хоча вона породжує проблеми пропускної спроможності тракту «процесор - пам'ять». Останнім часом у зв'язку з широким використанням кеш-пам'яті розробники ОМ все частіше звертаються до гарвардської архітектури.

Принцип адресності. Структурно основна пам'ять складається з пронумерованих комірок, причому процесору в довільний момент доступною є будь-яка комірка. Двійкові коди команд і даних розділяються на одиниці інформації, звані *словами*, і зберігаються в комірках пам'яті, а для доступу до них використовуються номери відповідних комірок – *адреси*.

Фон-нейманівська архітектура. У статті фон Неймана визначені основні пристрої ОМ, за допомогою яких повинні бути реалізовані вищеперелічені принципи. Більшість сучасних ОМ за своєю структурою відповідають принципу програмного управління. Типова фон-нейманівська ОМ містить: пам'ять, пристрій управління, арифметико-логічний пристрій і пристрій вводу/виводу (рис. 1.2).

У будь-якій ОМ є засоби для введення програм і даних до них. Інформація поступає з приєднаних до ЕОМ *периферійних пристроїв* (ПП) введення.

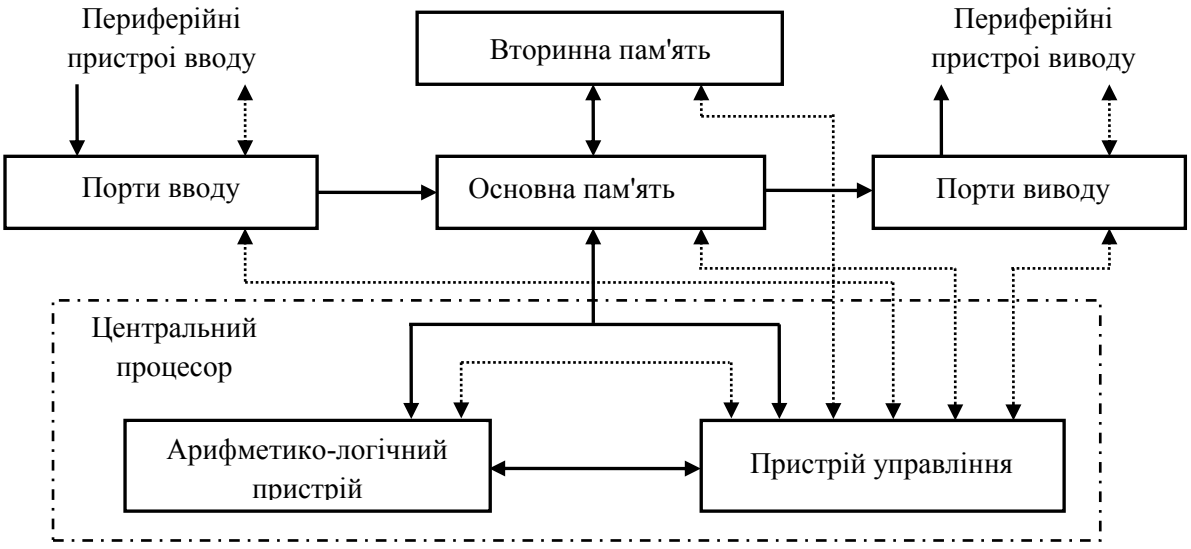


Рис. 1.2. Структура фон-нейманівської обчислювальної машини

Результати обчислень виводяться на периферійні пристрої виводу. Зв'язок і взаємодія ОМ і ПП забезпечують *порти вводу* і *порти виводу*. Терміном *порт* позначають апаратуру сполучення периферійного пристрою з ОМ і управління ним. Сукупність портів вводу і виводу називають *пристроєм вводу/виводу* (ПВВ) або *модулем вводу/виводу* ОМ (МВВ).

Введена інформація спочатку запам'ятовується в основній пам'яті, а потім переноситься у вторинну пам'ять, для тривалого зберігання. Щоб програма могла виконуватися, команди і дані повинні розташовуватися в *основній пам'яті* (ОП), організованій таким чином, що кожне двійкове слово зберігається в окремій комірці, що ідентифікується адресою, причому сусідні комірки пам'яті мають наступні за порядком адреси. Доступ до будь-яких комірок запам'ятовуючого пристрою (ЗП) основної пам'яті може проводитися в довільній послідовності. Такий вид пам'яті відомий як пам'ять *з довільним доступом*. ОП сучасних ОМ в основному складається з напівпровідникових *оперативних запам'ятовуючих пристроїв* (ОЗП), які забезпечують як зчитування, так і запис інформації. Для таких ЗП характерна енергозалежність – інформація, що зберігається, втрачається при відключенні електроживлення. Якщо необхідно, щоб частина основної пам'яті була енергонезалежною, в склад ОП включають *постійні запам'ятовуючі пристрої* (ПЗП), які також забезпечують довільний доступ. Інформація, що зберігається в ПЗП, може тільки зчитуватися (але не записуватися).

Розмір комірки основної пам'яті зазвичай приймається рівним 8 двійковим розрядам – *байту*. Для зберігання великих чисел використовуються 2, 4 або 8 байтів, що розміщуються в комірках з послідовними адресами.

Для довготривалого зберігання великих програм і масивів даних в ОМ зазвичай є додаткова пам'ять, відома як *вторинна*. Вторинна пам'ять енергонезалежна і найчастіше реалізується на базі магнітних дисків. Інформація

в ній зберігається у вигляді спеціальних програмно підтримуваних об'єктів – *файлів* (згідно зі стандартом ISO, файл – це «ідентифікована сукупність екземплярів повністю описаного в конкретній програмі типу даних, що знаходяться поза програмою в зовнішній пам'яті і доступних програмі за допомогою спеціальних операцій») [25].

Пристрій управління (ПУ) – найважливіша частина ОМ, що організує автоматичне виконання програм (шляхом реалізації функцій управління) і забезпечує функціонування ОМ як єдиної системи. Для пояснення функцій ПУ ОМ слід розглядати як сукупність елементів, між якими відбувається пересилання інформації, в ході якої ця інформація може піддаватися певним видам обробки. Пересилання інформації між будь-якими елементами ОМ ініціюється своїм *сигналом управління* (СУ) в потрібній тимчасовій послідовності. Ланцюги СУ показані на рис. 1.2 півтоновими лініями. Основною функцією ПУ є формування управляючих сигналів, що відповідають за витягання команд з пам'яті в порядку, який визначається програмою, і подальше виконання цих команд. Крім того, ПУ формує СУ для синхронізації і координації внутрішніх і зовнішніх пристроїв ОМ.

Ще однією невід'ємною частиною ОМ є *арифметико-логічний пристрій* (АЛП). АЛП забезпечує арифметичну і логічну обробку двох вхідних змінних, в результаті якої формується вихідна змінна. Функції АЛП зазвичай зводяться до простих арифметичних і логічних операцій, а також операцій зсуву. Крім результату операції АЛП формує ряд *ознак результату* (прапорів), що характеризують отриманий результат і події, які сталися в процесі його отримання (рівність нулю, знак, парність, перенесення, переповнення і т. д.). Прапори можуть аналізуватися в ПУ з метою ухвалення рішення про подальшу послідовність виконання команд програми.

ПУ і АЛП тісно взаємозв'язані і їх зазвичай розглядають як єдиний пристрій, відомий як *центральний процесор* (ЦП) або просто *процесор*. Крім ПУ і АЛП в процесор входить також набір *регістрів загального призначення* (РЗП), які виконують функцію проміжного зберігання інформації в процесі її обробки.

1.4. ПРИНЦИП ДІЇ ФОН-НЕЙМАНІВСЬКОЇ ЕЛЕКТРОННОЇ ОБЧИСЛЮВАЛЬНОЇ МАШИНИ

Програма у фон-нейманівській ЕОМ реалізується центральним процесором (ЦП) за допомогою послідовного виконання створюючих цю програму команд. Дії, потрібні для вибірки і виконання команди, називають *циклом команди*. В загальному випадку цикл команди включає декілька складових (етапів):

- вибірку команди;
- формування адреси наступної команди;
- декодування команди;
- обчислення адрес операндів;
- вибірку операндів;
- виконання операції;
- запис результату.

Перераховані етапи виконання команди надалі називатимемо стандартним циклом команди. Відзначимо, що не всі з етапів присутні під час виконання будь-якої команди (залежить від типу команди), проте етапи вибірки, декодування, формування адреси наступної команди і виконання мають місце завжди.

Стисло охарактеризуємо кожен з вищеперелічених етапів стандартного циклу команди.

Етап вибірки команди. Цикл будь-якої команди починається з того, що центральний процесор витягує команду з пам'яті, використовуючи адресу, що зберігається в лічильнику команд (ЛК). Двійковий код команди поміщається в регістр команди (РК) і з цієї миті стає «видимим» для процесора. Лічильник команд і регістр команд розташовані в пристрої управління. Система команд багатьох ОМ припускає декілька форматів команд, причому в різних форматах команда може займати 1, 2 або більше комірок. У цьому випадку етап вибірки команди можна вважати завершеним лише після того, як в РК буде поміщений повний код команди. Інформація про фактичну довжину команди міститься в полях коду операції і способу адресації. Зазвичай ці поля розташовують у першому слові коду команди, і для з'ясування необхідності продовження процесу вибірки необхідне попереднє декодування їх вмісту. Таке декодування може бути проведене після того, як перше слово коду команди опиниться в РК. У разі багатослівного формату команди процес вибірки продовжується аж до занесення в РК всіх слів команди.

Етап формування адреси наступної команди. Для фон-нейманівських машин характерне розміщення сусідніх команд програми в суміжних комірках пам'яті. Якщо вибрана з пам'яті команда не порушує природного порядку виконання програми, то для обчислення адреси наступної виконуваної команди досить збільшити вміст лічильника команд на довжину поточної команди, яка подана кількістю займаних кодом команди комірок пам'яті. Довжина команди, а також те, чи здатна вона змінити природний порядок виконання команд програми, з'ясовуються в ході раніше згадуваного попереднього декодування. Якщо вибрана з пам'яті команда здатна змінити послідовність виконання програми (команда умовного або безумовного переходу, виклику процедури і т. п.),

процес формування адреси наступної команди переноситься на етап виконання операції. В силу сказаного, в ряді ОМ даний етап циклу команди йде не за вибіркою команди, а знаходиться в кінці циклу.

Етап декодування команди. Після вибірки команди вона повинна бути декодована, для чого ЦП розшифровує код команди, що знаходиться в РК. В результаті декодування з'ясовуються такі моменти:

- чи знаходиться в РК повний код команди або потрібне дозавантаження решти слів команди;
- які подальші дії потрібні для виконання даної команди;
- якщо команда використовує операнди, то звідки вони повинні бути узяті (номер регістра або адреса комірки основної пам'яті);
- якщо команда формує результат, то куди цей результат повинен бути направлений.

Відповіді на два перші питання дає розшифровка коду операції, результатом якої може бути унітарний код, де кожен розряд відповідає одній з команд. На практиці замість унітарного коду можуть зустрітися найрізноманітніші форми подання результатів декодування, наприклад адреса комірки спеціальної управляючої пам'яті, де зберігається перша мікрокоманда мікропрограми для реалізації вказаної в команді операції.

Повне з'ясування всіх аспектів команди, крім розшифровки коду операції, вимагає також аналізу адресної частини команди, включаючи поле способу адресації.

За наслідками декодування проводиться підготовка електронних схем ОМ до виконання наказаних командою дій.

Етап обчислення адрес операндів. Етап має місце, якщо в процесі декодування команди з'ясовується, що команда використовує операнди. Якщо операнди розміщуються в основній пам'яті, здійснюється обчислення їх виконавчих адрес, з урахуванням вказаного в команді способу адресації. Так, у разі індексної адресації для отримання виконавчої адреси проводиться підсумовування вмісту адресної частини команди і вмісту індексного регістра.

Етап вибірки операндів. Обчислені на попередньому етапі виконавчі адреси використовуються для зчитування операндів з пам'яті і занесення в певні регістри процесора. Наприклад, у разі арифметичної команди операнд після вибору з пам'яті може бути завантажений у вхідний регістр АЛП. Проте частіше операнди заздалегідь заносяться в спеціальні допоміжні регістри процесора, а їх пересилка на вхід АЛП відбувається на етапі виконання операції.

Етап виконання операції. На цьому етапі в АЛП реалізується вказана в команді операція. Через відмінність суті кожної з команд ОМ, зміст цього етапу також суто індивідуальний.

Етап запису результату. Етап запису результату присутній в циклі тих команд, які припускають занесення результату в регістр або комірку основної пам'яті. Якщо отриманий результат буде використаний в наступній команді, то він може залишатися в регістрі АЛП.

1.5. ЕТАПИ РОЗВИТКУ АРХІТЕКТУРИ ФОН НЕЙМАНА

У 1946 році відомий американський математик Дж. фон Нейман вперше сформулював основні принципи побудови ЕОМ, що програмно управляються:

1) принцип програмного управління – ЕОМ може автоматично перетворювати вихідні дані відповідно до заданої програми;

2) принцип умовного переходу – надає гнучкість та універсальність програм за рахунок забезпечення можливості переходу в процесі розв'язання задачі на певну ділянку програми залежно від результатів проміжних обчислень або вихідних даних;

3) принцип збереженої програми – програму розміщують у запам'ятовуючий пристрій ЕОМ;

4) принцип довільного доступу до елементів пам'яті;

5) принцип використання двійкової системи числення;

6) принцип багаторівневої (ієрархічної) пам'яті.

Ці принципи актуальні і для сучасних ЕОМ, але зі створенням нових поколінь та сімейств машин вони доповнювались та уточнювались.

В ЕОМ, починаючи з **третього покоління**, додатково застосовуються такі принципи:

- мультипрограмування – сумісне використання різних команд однієї й тієї ж або різних, незалежно одна від одної, програм, які зберігаються в оперативній пам'яті;
- інформаційна та програмна сумісність – дає змогу виконувати наявні програми на різних моделях сімейства;
- високий рівень технічної стандартизації – єдина для всіх машин номенклатура зовнішніх та інших пристроїв;
- можливість організації багатоетапної роботи зі створення та удосконалення ЕОМ.

Комп'ютери четвертого покоління будуються за принципами:

- багатопроцесорності – комутація декількох процесорів під час роботи зі спільною пам'яттю;
- організації віртуальної пам'яті – забезпечує практично необмежений об'єм адресного простору ОЗП;

- широкого використання ВІС та НВІС і макромодульної структури, в основі якої – ідея побудови з великих стандартизованих блоків (макромодулів) функціонально гнучких обчислювальних систем;
- використання внутрішніх мов високого рівня.

Комп'ютери п'ятого покоління відрізняються:

- подальшим розвитком функції введення-виведення графіки, зображень, документів, мови;
- можливістю діалогової обробки інформації за допомогою природної мови;
- здатністю до самовдосконалення, до асоціативних побудов та отримання висновків.

Мови програмування в процесі форматування програм можуть реалізовувати безпосередній інтерфейс між людиною і машиною. Це мови надвисокого рівня, які забезпечують цю можливість:

- підтриманням засобів верифікації та підвищенням загальної надійності програм;
- забезпеченням розумної взаємодії користувачів з обчислювальною системою на різному рівні доступу до бази даних для вибору потрібної інформації та до бази знань для отримання нових уявлень, необхідних для розв'язання незнайомих задач;
- використанням існуючих програмних фондів, орієнтованих на традиційну архітектуру ЕОМ.

1.6. СТРУКТУРНА СХЕМА IBM PC – СУМІСНОГО ПЕРСОНАЛЬНОГО КОМП'ЮТЕРА

Персональний комп'ютер (ПК) є сукупністю двох важливих частин: *апаратного* та *програмного забезпечення*. На сьогоднішній день найбільше розповсюдження отримали IBM PC – сумісні персональні комп'ютери, структурна схема яких подана на рис. 1.3.

Архітектура PC – сумісного комп'ютера визначається рядом властивостей, які забезпечують можливість функціонування програмного забезпечення, що управляє периферійним обладнанням. Програми можуть взаємодіяти з пристроями різними способами [5]:

- через виклики функцій операційної системи;
- через виклики функцій базової системи вводу/виводу (BIOS);
- безпосередньо взаємодіючи з відомими їм пристроями.

Апаратне забезпечення персонального комп'ютера складається з таких частин:

системний блок комп'ютера – пристрій розміщений в компактному металевому чи пластмасовому корпусі, в якому містяться всі електронні компоненти ПК, а також блок живлення;

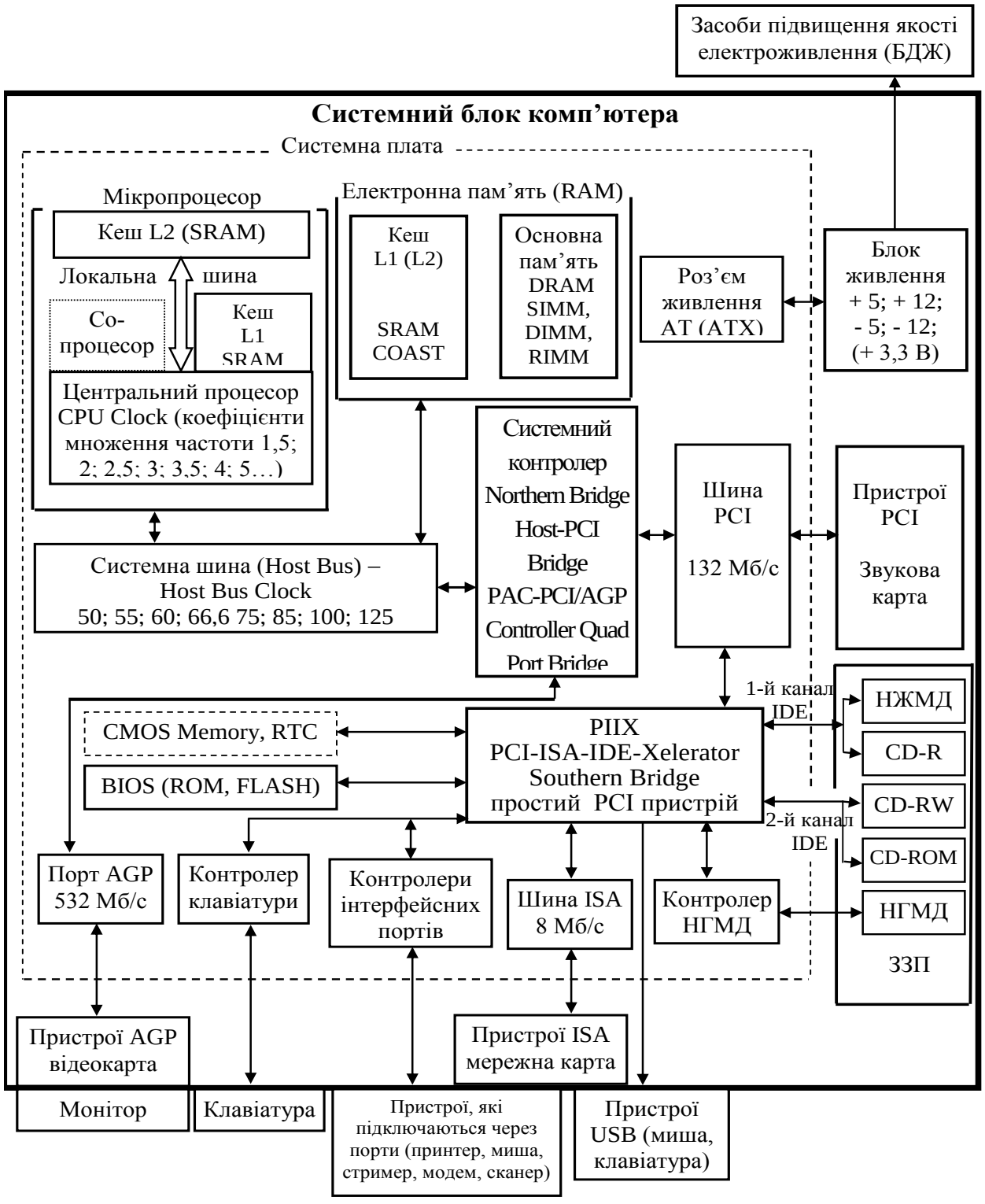


Рис. 1.3. Структурна схема IBM PC – сумісного комп'ютера

клавіатура – основний пристрій ручного введення інформації в ПК;

монітор – найголовніший пристрій вводу/виводу візуальної інформації ПК.

Крім того, в ПК можна використовувати додаткові пристрої вводу/виводу інформації (ПВВ) і засоби покращання якості електроживлення (наприклад,

БДЖ – безперебійне джерело живлення для захисту від раптового зникнення напруги в мережі).

У **системному блоці** (рис. 1.3) розташовуються такі компоненти:

Системна плата (об'єднуюча, материнська) є основним компонентом комп'ютера. Вона, як правило, містить:

- сокет (слот) процесора;
- сокети (слоти) пам'яті SIMM і DIMM;
- слоти шини;
- ПЗП BIOS (ROM BIOS);
- кеш 2-го рівня (кеш L2, Level 2 cache);
- набір мікросхем системної логіки (чіпсет) системної плати;
- чіп вводу/виводу;
- перетворювачі напруги живлення процесора;
- батарею для живлення годинника і CMOS.

Набір мікросхем системної логіки містить усі схеми, які входять до складу системної плати. Він управляє центральним процесором, системною шиною процесора, кешем L2, оперативною пам'яттю, шиною PCI (*Peripheral Component Interconnect*), шиною ISA (*In-dustry Standard Architecture)* чи іншою, ідентичною за функціональним призначенням, ресурсами системи та ін.

Набір мікросхем системної логіки визначає також первинні можливості та специфікації системної плати – типи підтримуваних процесорів, пам’яті, плат розширення, дисководів та ін.

Системні плати випускають у кількох варіантах, які відрізняються розмірами та форм-факторами. *Форм-фактор* системної плати визначає тип корпусу, в який її можна вставити.

Мікропроцесор (МП) – це основний «мозковий» вузол ПК, в задачу якого входить виконання програмного коду, що знаходиться в пам'яті, і керування іншими пристроями. До складу мікропроцесора сучасного ПК крім центрального мікропроцесора – пристрою обробки CPU (*Central Processing Unit*) – входить математичний співпроцесор CoP (*CoProcessor*), який ефективно обробляє числові дані у форматі з плаваючою комою (крапкою), і невелика за обсягом швидкодіюча кэш-пам'ять, реалізована за одно- або дворівневою схемою (від англ. *cache* – склад, схованка; кеш-пам'ять призначена для пом'якшення наслідків, що викликані розузгодженням швидкості роботи швидкого МП і повільної оперативної пам'яті).

Мікропроцесор – це найчастіше одна надвелика інтегральна схема (НВІС), реалізована в єдиному напівпровідниковому кристалі. Ступінь інтеграції визначається розміром кристала і кількістю реалізованих у ньому транзисторів. Деякі МП, строго кажучи, не є однокристальними – кристал центрального

процесора зі співпроцесором і кілька кристалів вторинного кеша зібрані на загальному картриджі.

У багатопроцесорній системі для підвищення загальної продуктивності системи функції центрального процесора розподіляються між декількома, звичайно ідентичними мікропроцесорами, один із яких призначається головним.

Співпроцесор – це спеціалізований процесор, призначений для розвантаження центрального процесора від складних обчислень з плаваючою комою. Співпроцесор в МП класу 486 і старше вбудований у середину кристала мікропроцесора.

Кеш–пам'ять першого рівня *(Level 1 – L 1*) у процесорів класу 486 і старше вбудована у середину кристала і працює на однаковій з ним частоті. Якщо МП використовує дворівневу модель кеш-пам'яті (*L1, L2*), то застосовується архітектура подвійної незалежної шини (ПНШ – *Dual Independent Bus*). Одна з шин МП архітектури ПНШ – локальна шина – використовується тільки для зв'язку з кристалами вторинного кеша, які розташовані у тому ж корпусі мікросхеми або на загальному картриджі. Ця шина є локальною і у геометричному значенні – провідники мають довжину одиниць сантиметрів, що дозволяє її використовувати навіть на частоті ядра процесора.

Друга шина МП виходить на зовнішні виводи мікросхеми, вона є системною шиною МП. Ця шина працює на зовнішній частоті незалежно від внутрішньої шини.

МП керує роботою ПК, отримуючи і посилаючи керуючі сигнали, адреси пам'яті і дані від одних компонентів ПК до інших компонентів, використовуючи для цього групу сполучаючих електронних шляхів, які називаються системною шиною (СШ).

Системна шина - це електронний шлях на системній платі, що спільно використовується, до якого підключені всі керовані компоненти ПК. Коли дані передаються від одного компонента до іншого, вони переміщаються вздовж цього загального шляху до місця призначення. СШ поділяється на чотири складові частини: лінії передачі електроживлення, керуюча шина (для передачі керуючої інформації, наприклад, такої, як тактові сигнали від системного генератора тактових сигналів, сигнали переривання і т.д.), адресна шина (виконує передачу адрес комірок пам'яті і пристроїв, приєднаних до шини) і шина даних (разом із шиною адреси здійснює перенесення даних в середині ПК).

Оскільки швидкодія різних компонентів ПК (мікропроцесора, пам'яті й інших пристроїв) істотно розрізняється, в комп'ютерах на МП класу 486 і старше застосовується внутрішнє множення частоти. Розрізняють такі частоти:

- *Host Bus Clock* – частота системної шини (зовнішня частота процесора), опорна для всіх інших частот. МП класу Pentium і старше використовують частоти 50, 55, 60, 66,6, 75, 83, 100, 125 МГц і вище.
- *CPU Clock* чи *Core Speed* - внутрішня частота МП, на якій працює його обчислювальне ядро (центральний процесор, співпроцесор, кэш-пам'ять L1). Сучасні технології дозволили істотно підвищити граничні частоти інтегральних компонентів, у зв'язку з чим широко застосовується внутрішнє множення частоти на 1,5, 2, 2,5, 3, 3,5, 4 і деякі інші значення.

Основна чи оперативна пам'ять ПК призначена для оперативного обміну (збереження, запису і зчитування) інформацією (кодами і даними) між МП, зовнішньою пам'яттю і периферійними пристроями. Для побудови основної пам'яті в ПК використовують мікросхеми динамічної пам'яті (мікросхеми DRAM – пам'яті – *Dynamic Random Access Memory*), що мають найкраще сполучення обсягу, щільності упакування, енергоспоживання і ціни. Мікросхеми DRAM – пам'яті в сучасному ПК установлюють на спеціальні модулі пам'яті SIMM (*Single In – Line Memory Module*), DIMM (*Dual In – Line Memory Module*), RIMM (*Rambus In – Line Memory Module*) у відповідні гнізда системної плати.

Важливою особливістю основної пам'яті ПК є її ієрархічний спосіб побудови, що прийшов в архітектуру ПК з появою процесора 386, і полягає в сполученні основної пам'яті великого обсягу на мікросхемах динамічної пам'яті з відносно невеликою кеш-пам'яттю на швидкодіючих мікросхемах статичної пам'яті SRAM (*Static RAM*).

Кеш-пам'ять є додатковим і швидкодіючим сховищем копій блоків інформації основної пам'яті, до яких, імовірно, найближчим часом буде звернення. У сучасному комп'ютері кеш-пам'ять побудована за трирівневою схемою:

- *кеш L1* – кеш на системній платі 386 процесорів, працює на *Host Bus Clock*; кеш, вбудований в кристал мікропроцесора класу 486 і старше, працює на *CPU Clock*;
- *кеш L2* – кеш на системній платі мікропроцесора класу 486 і старше (за винятком МП Pentium Pro, Pentium II Xeon, Pentium II, Celeron 300A і старше, K6–3 і їхніх мобільних варіантів) працює на *Host Bus Clock*; кеш вбудований в корпус МП чи встановлений на загальному картриджі (для МП Pentium Pro, Pentium II Xeon, Pentium II, Celeron 300A і старше, K6–3) працює на *CPU Clock* чи на половині цієї частоти;
- *кеш L3* – кеш на системній платі мікропроцесора K6–3 працює на *Host Bus Clock*.

Кеш на системній платі сучасного ПК набирається мікросхемами статичної пам'яті фіксованого обсягу, запаяних на плату без застосування додаткових

модулів і роз'ємів (для старих системних плат із МП Pentium широке поширення отримали модулі COAST (*Cache On A Stick)* – «кеш на полиці»; модуль із двостороннім друкованим роз'ємом, який встановлений у спеціальний слот).

CMOS Memory *(Complementary Metal-Oxide-Semiconductor),* RTC *(Real Time Clock)* – спеціальна мікросхема напівпостійної пам'яті (КМОН – пам'ять) невеликого обсягу для збереження інформації про конфігурацію ПК разом з годинником і календарем. Живлення CMOS Memory, RTC при виключеному ПК здійснюється від батарейки.

BIOS *(Basic Input Output System)* – ключовий елемент системної плати, призначений для енергонезалежного збереження системної інформації. BIOS, користуючись засобами, наданими чипсетом, керує всіма компонентами і ресурсами системної плати. Код BIOS зберігається в мікросхемі енергонезалежної постійної пам'яті *(Read Only Memory (ROM BIOS)* чи флеш-пам'яті *(Flash BIOS).*

Шини розширення призначені для підключення різних адаптерів чи контролерів периферійних пристроїв, що розширюють можливості ПК. Під адаптером звичайно розуміють засіб сполучення якого-небудь пристрою з шиною ПК (контролер служить тим же цілям сполучення, але при цьому мається на увазі його деяка активність, тобто здатність до самостійних дій після отримання команд від обслуговуючої його програми).

В сучасному ПК застосовують такі шини, розташовані на системній платі у виді слотів чи секцій розширення:

- *PCI – Peripheral Component Interconnect Local Bus*;
- *AGP – Accelerated Graphic Port* – прискорений графічний порт;
- *ISA – Industry Standard Architecture.*

Зовнішні інтерфейси ПК (рівнобіжний, послідовний і USB-шина), як і шини розширення, дозволяють значно розширити функціональні можливості ПК. Однак, на відміну від шин розширення, зовнішні інтерфейси дозволяють підключати різні периферійні пристрої прямо, без використання додаткових адаптерів.

Сучасні системні плати будують на основі чипсетів *(Chipset)* – набір з декількох надвеликих інтегральних схем, що реалізують усі необхідні функції зв'язку основних компонентів: МП, пам'яті і шин розширення. Майже всі сучасні чипсети є набором із двох мікросхем, що прийнято називати *Northern Bridge* (Північний міст) і *Southern Bridge* (Південний міст).

Northern Bridge відповідає за роботу з МП, системною шиною, PCI-шиною, AGP-портом, пам'яттю і кешем (які не входять до складу МП).

Southern Bridge – це фактично простий PCI – пристрій, що містить всередині себе контролер *Bus Master IDE* (який дозволяє пристроям, що

знаходяться на такій шині, самим керувати процесом передачі даних по ній без участі процесора), міст PCI–ISA.

Периферія досить стабільна, тому архітектура *Southern Bridge* набагато менше піддається змінам, чим *Northern Bridge,* що дозволяє в нових чипсетах використовувати його попередні версії.

Міст PCI–IDE потрібний для роботи з існуючими накопичувачами з жорсткими магнітними дисками (НЖМД), міст PCI–ISA є рудиментом, що відмирає. Справа в тому, що дотепер BIOS, контролер гнучких дисків, послідовні і рівнобіжні порти, інші пристрої, яким цілком достатньо невеликої пропускної здатності, знаходяться на шині ISA. Однак уже відбувається відмова від старої шини ISA і повний перехід на PCI.

Контролери накопичувачів на гнучких магнітних дисках (НГМД), інтерфейсних портів, клавіатури, CMOS Memory, RTC можуть входити власне в чипсет, а можуть бути реалізовані на окремих «сторонніх» мікросхемах.

Зовнішні запам'ятовуючі пристрої комп'ютера (ЗЗП) – це пристрої, що дозволяють автономно зберігати інформацію для наступного її використання незалежно від стану ПК (включений чи виключений). В сучасному ПК в ЗЗП входять пристрої магнітної, оптичної і магнітооптичної пам'яті. ЗЗП можна розміщувати як у системному блоці комп'ютера, так і в окремому корпусі.

Блок живлення призначений для перетворення змінного електричного струму в постійний, який стабілізований у невеликих межах, а також для захисту електричних ланцюгів блока живлення і комп'ютера від впливу різних перешкод і несправностей.

1.7. ОСНОВНІ ТИПИ ТА ХАРАКТЕРИСТИКИ ОБЧИСЛЮВАЛЬНИХ МАШИН

Обчислювальні машини можуть бути класифіковані за рядом ознак, зокрема: принцип дії; етапи створення та елементна база; призначення; спосіб організації обчислювального процесу; розмір, обчислювальна потужність; функціональні можливості; спроможність до паралельного виконання програм і т.д. [3].

За принципом дії обчислювальні машини діляться на три великих класи: аналогові, цифрові, гібридні.

Критерієм розподілу обчислювальних машин на ці три класи являється форма подання інформації, з якою вони працюють.

За етапами створення і елементної бази ОМ умовно діляться на покоління:

- 1-ше покоління, 50-ті роки: ЕОМ на електронних вакуумних лампах;
- 2-ге покоління, 60-ті роки: ЕОМ на дискретних напівпровідникових пристроях (транзисторах);

- 3-тє покоління, 70-ті роки: ЕОМ на напівпровідникових інтегральних схемах з малим і середнім ступенем інтеграції(сотні-тисячі транзисторів в одному корпусі);
- 4-те покоління, 80-90-ті роки: ОМ на великих та надвеликих інтегральних схемах, основна з яких – мікропроцесор (сотні тисяч-десятки мільйонів активних елементів в одному кришталі).
- 5-те покоління, теперішній час: ОМ з багатьма десятками паралельно працюючих мікропроцесорів, що дозволяють будувати ефективні системи обробки знань; комп'ютери на надскладних мікропроцесорах з паралельно-векторною структурою, одночасно виконуючих десятки послідовних інструкцій програми.
- 6-те і наступні покоління: оптоелектронні ОМ з масовим паралелізмом та нейронною структурою, з розподіленою сіткою великого числа (десятки і тисячі) простих мікропроцесорів, що моделюють архітектуру нейронних біологічних систем.

Кожне наступне покоління обчислювальних машин має порівняно з попереднім суттєво кращі характеристики. Так, продуктивність ОМ та ємність усіх запам'ятовуючих пристроїв збільшується, як правило, більше, ніж на порядок.

За призначенням ОМ можна розділити на три групи: універсальні (загального призначення), проблемно-орієнтовані, спеціалізовані.

Універсальні ОМ призначені для вирішення різноманітних інженерно-технічних, економічних, математичних, інформаційних та інших задач, що відрізняються складністю алгоритмів і великим об'ємом оброблюваних даних. Вони широко використовуються в обчислювальних центрах колективного використання та інших могутніх обчислювальних комплексах.

Характерними рисами універсальних ОМ є:

- висока продуктивність;
- різноманітність форм оброблюваних даних: двійкових, десяткових, символьних, при великому діапазоні їх змінення і високій точності їх подання;
- широка номенклатура вироблених операцій як арифметичних, логічних, так і спеціальних;
- велика ємність оперативної пам'яті;
- розвинута організація системи вводу/виводу інформації, що забезпечує підключення різних видів зовнішніх пристроїв.

Проблемно-орієнтовані ОМ призначені для вирішення більш вузького кола задач, пов'язаних, як правило, з управлінням технологічними об'єктами; реєстрацією, накопиченням і обробкою відносно невеликих об'ємів даних; виконанням розрахунків за порівняно нескладними алгоритмами. Вони володіють

обмеженими, порівняно з універсальними ОМ, апаратними і програмними ресурсами.

Спеціалізовані ОМ призначені для вирішення певного вузького кола задач або реалізації строго певної групи функцій.

Така вузька орієнтація ОМ дозволяє чітко спеціалізувати їх структуру, істотно понизити їх складність і вартість при збереженні високої продуктивності і надійності їх роботи. До спеціалізованих комп'ютерів можна віднести, наприклад, програмуючі мікропроцесори спеціального призначення; адаптери і контролери, виконуючі логічні функції управління окремими нескладними технічними пристроями, агрегатами і процесами; пристрої узгодження і сполучення роботи вузлів обчислювальних систем.

За розмірами і обчислювальною потужністю (рис. 1.4) обчислювальні машини можна розділити на надвеликі (суперкомп'ютери, супер-ЕОМ), великі, малі, надмалі (мікрокомп'ютери або мікро-ЕОМ).

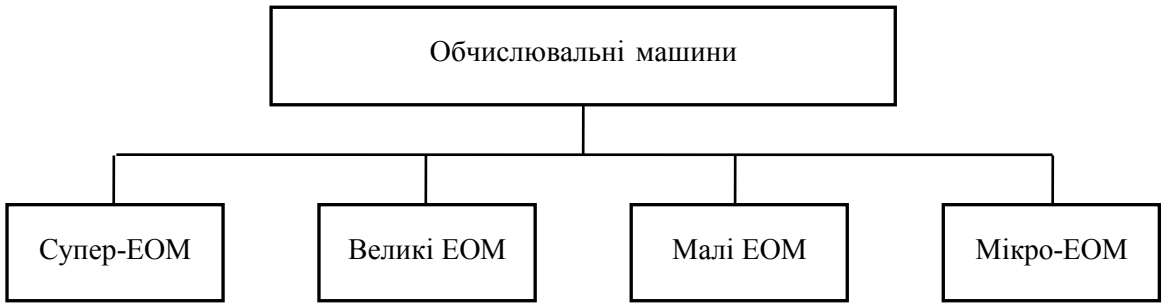


Рис. 1.4. Класифікація ОМ за розмірами і обчислювальною потужністю

Історично першими з'явилися *великі* ЕОМ, елементна база яких пройшла шлях від електронних ламп до інтегральних схем з надвисоким ступенем інтеграції.

Продуктивність великих ЕОМ виявилася недостатньою для ряду задач (метеопрогнозування, управління складними оборонними комплексами, біологічні дослідження, моделювання екологічних систем та ін.) Це виявилося передумовою для розробки і створення *супер-ЕОМ*, наймогутніших обчислювальних систем, що інтенсивно розвиваються і в даний час. Поява в 70-х роках *малих* комп'ютерів обумовлена, з одного боку, прогресом в області електронної елементної бази, а з другого – надмірністю ресурсів великих ЕОМ для ряду додатків. Малі комп'ютери використовуються частіше за все для управління технологічними процесами. Вони більш компактні і істотно дешевші за великі комп'ютери. Подальші успіхи в області елементної бази і архітектурних рішень привели до виникнення *суперміні-комп'ютера* – обчислювальної машини, що відноситься за архітектурою, розмірами і вартістю до класу малих комп'ютерів, але за продуктивністю порівняні з великою ОМ.

Винахід у 1969 році мікропроцесора (МП) привів до появи в 70-х роках ще одного класу комп'ютерів – *мікрокомп'ютерів*. Саме наявність МП послужило спочатку визначальною ознакою мікрокомп'ютерів. Зараз мікропроцесори використовуються у всіх без виключення класах комп'ютерів.

Розглянемо стисло сучасний стан деяких класів комп'ютерів.

Великі ЕОМ за кордоном часто називають мейнфреймами (main-frame); до них відносять, як правило, ОМ, що мають:

- продуктивність не менше 100 M1PS (M1PS – мільйон операцій в секунду над числами з фіксованою комою)
- основну пам'ять ємністю від 512 до 10 000 Мбайт;
- зовнішню пам'ять не менше 100 Гбайт;
- розрахований на багато користувачів режим роботи (обслуговують одночасно від 16 до1000 користувачів).

Основні напрями ефективного застосування мейнфреймів – рішення науково-технічних задач, робота в обчислювальних системах з пакетної обробки інформації, робота з великими базами даних, управління обчислюваними мережами і їх ресурсами. Останній напрям – використання мейнфреймів як великих серверів обчислювальних мереж – часто наголошується спеціалістами як найактуальніший.

Малі комп'ютери (міні-ЕОМ) – надійні, недорогі і зручні в експлуатації комп'ютери, що володіють дещо більш низькими в порівнянні з мэйнфреймами можливостями.

Міні-комп'ютери (і найбільш могутні з них суперміні-комп'ютери) мають такі характеристики:

- продуктивність – до 1000 M1PS;
- ємність основної пам'яті – до 8000 Мбайт;
- ємність дискової пам'яті – до 1000 Гбайт;
- число підтримуваних користувачів – 16-1024.

Міні-комп'ютери орієнтовані на використання як управляючі обчислювані комплекси. Традиційна для подібних комплексів широка номенклатура периферійних пристроїв доповнюється блоками міжпроцесорного зв'язку, завдяки чому забезпечується реалізація обчислювальних систем із змінною структурою. Наряду з використанням міні-комп'ютерів для управління технологічними процесами, вони успішно застосовуються для обчислень в обчислювальних системах, які розраховані на багато користувачів.

Мікрокомп'ютери досить численні і різноманітні. Серед них можна виділити декілька підкласів.

Багатокористувальні мікрокомп'ютери – це могутні мікрокомп'ютери, що устатковані декількома відеотерміналами і працюють в режимі розділення часу, що дозволяє ефективно працювати на них відразу декільком користувачам.

Робочі станції (work station) - мікрокомп'ютери, які розраховані на одного користувача, часто спеціалізовані для виконання певного виду робіт (графічних, інженерних, видавничих і т. д.).

Сервери (server) розраховані на багато користувачів, могутні мікрокомп'ютери в обчислювальних мережах та призначені для обробки запитів від всіх робочих станцій мережі.

Мережні комп'ютери (network computer) – спрощені мікрокомп'ютери, що забезпечують роботу в мережі і доступ до мережних ресурсів, часто спеціалізовані на виконанні певного виду робіт (захист мережі від несанкціонованого доступу, організація перегляду мережних ресурсів, електронної пошти і т. д.).

Персональні комп'ютери (ПК) належать до класу мікрокомп'ютерів, але через їх масове поширення заслуговують особливої уваги. ПК для задоволення вимог загальнодоступності і універсальності застосування повинні відповідати таким якостям:

- низька вартість, що знаходиться в межах доступності для індивідуального покупця;
- автономність експлуатації без спеціальних вимог до умов навколишнього середовища;
- гнучкість архітектури, що забезпечує її адаптованість до різноманітних застосувань у сфері управління, науки, освіти, в побуті;
- сумісність операційної системи та іншого програмного забезпечення, обумовлююча можливість роботи з нею користувача без спеціальної професійної підготовки;
- висока надійність роботи (більше 5000 годин напрацювання на відмову).

Функціональні можливості обчислювальних машин обумовлені такими найважливішими техніко-експлуатаційними характеристиками:

- швидкодія, яка вимірюється усередненою кількістю операцій, що виконує ОМ за одиницю часу;
- розрядність і форми подання чисел, з якими оперує комп'ютер;
- номенклатура, ємність і швидкодія всіх запам'ятовуючих пристроїв;
- номенклатура і техніко-економічні характеристики зовнішніх пристроїв зберігання, обміну і вводу/виводу інформації;
- типи і пропускна здатність пристроїв зв'язку і сполучення вузлів комп'ютера між собою (тип внутрішньомашинного інтерфейсу);
- здатність комп'ютера одночасно працювати з декількома користувачами і виконувати одночасно декілька програм (багатопрограмність);
- типи і техніко-експлуатаційні характеристики операційних систем, що використовуються в ОМ;

- наявність і функціональні можливості програмного забезпечення;
- здатність виконувати програми, написані для інших типів комп'ютерів (програмна сумісність з іншими типами комп'ютерів);
- система і структура машинних команд;
- можливість підключення до каналів зв'язку і обчислювальної мережі;
- експлуатаційна надійність комп'ютера;
- коефіцієнт корисного використання комп'ютера в часі, визначений співвідношенням часу корисної роботи і часу профілактики.

1.8. ТИПИ СТРУКТУР ОБЧИСЛЮВАЛЬНИХ МАШИН І СИСТЕМ

Переваги і недоліки архітектури обчислювальних машин і систем залежать від способу з'єднання компонентів. Загалом можна говорити про два основні типи структур обчислювальних машин і два типи структур обчислювальних систем.

1.8.1. Структури обчислювальних машин

В даний час приблизно однакове розповсюдження отримали два способи побудови обчислювальних машин: *з безпосередніми зв'язками* і *на основі шини*.

Типовим представником першого способу може служити класична фон-нейманівська ОМ (див. рис. 1.2). В ній між взаємодіючими пристроями (процесор, пам'ять, пристрій вводу/виводу) є безпосередні зв'язки. Особливості зв'язків (число ліній в шинах, пропускна здатність і т. п.) визначаються видом інформації, характером та інтенсивністю обміну. Перевагою архітектури з безпосередніми зв'язками можна вважати можливість розв'язки «вузьких місць» шляхом покращання структури і характеристик тільки певних зв'язків, що економічно може бути найбільш вигідним рішенням. У фон-нейманівських ОМ таким «вузьким місцем» є канал пересилки даних між ЦП і пам'яттю, і «розв'язати» його досить непросто. Крім того, ОМ з безпосередніми зв'язками погано піддаються реконфігурації.

У варіанті із загальною шиною всі пристрої обчислювальної машини підключені до магістральної шини, яка служить єдиним трактом для потоків команд, даних і управління (рис. 1.5).

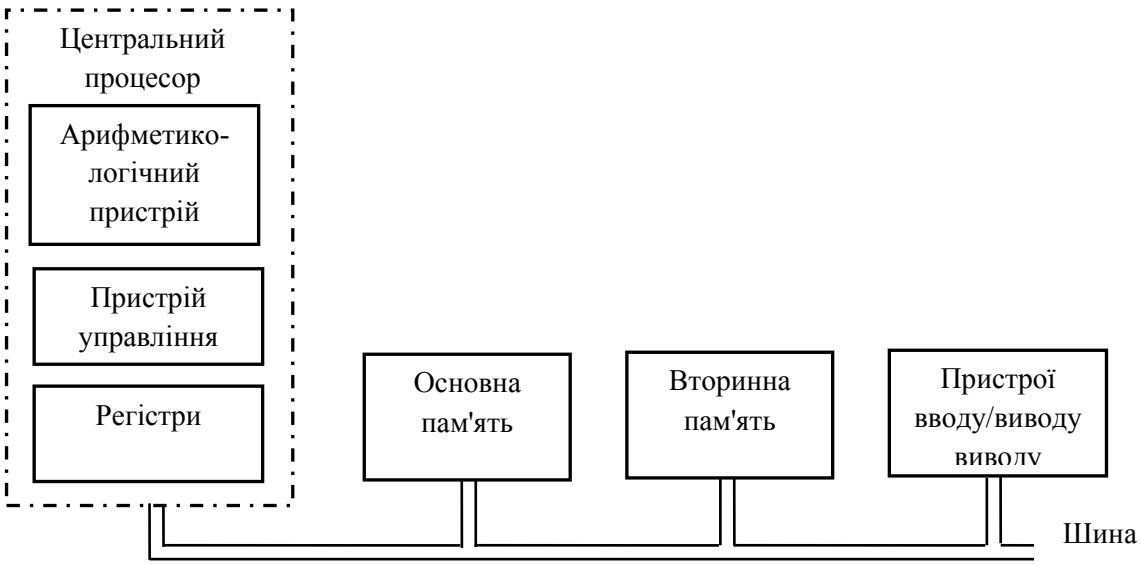


Рис. 1.5. Структура обчислювальної машини на базі загальної шини

Наявність загальної шини істотно спрощує реалізацію ОМ, дозволяє легко міняти склад і конфігурацію машини. Завдяки цим властивостям шинна архітектура набула широкого поширення в міні- і мікро-ЕОМ. Разом з тим, саме з шиною зв'язаний і основний недолік архітектури: у кожен момент передавати інформацію по шині може тільки один пристрій. Основне навантаження на шину створюють обміни між процесором і пам'яттю, пов'язані з витягуванням із пам'яті команд і даних і записом в пам'ять результатів обчислень. На операції вводу/виводу залишається лише частина пропускної здатності шини.

В цілому слід визнати, що при збереженні фон-нейманівської концепції послідовного виконання команд програми шинна архітектура в чистому її вигляді виявляється недостатньо ефективною. Більш поширеною є *архітектура з ієрархією шин*, де крім магістральної шини є ще декілька додаткових шин. Вони можуть забезпечувати безпосередній зв'язок між пристроями з найбільш інтенсивним обміном, наприклад процесором і кеш-пам'яттю. Інший варіант використання додаткових шин – об'єднання однотипних пристроїв вводу/виводу з подальшим виходом з додаткової шини на магістральну. Всі ці заходи дозволяють понизити навантаження на загальну шину і ефективніше витрачати її пропускну здатність.

1.8.2. Структури обчислювальних систем

Поняття «обчислювальна система» допускає наявність багатьох процесорів або закінчених обчислювальних машин, під час об'єднання яких використовується один з двох підходів.

В обчислювальних *системах із загальною пам'яттю* (рис. 1.6) є загальна основна пам'ять, спільно використовувана всіма процесорами системи. Зв'язок процесорів з пам'яттю забезпечується за допомогою комунікаційної мережі, найчастіше вироджуваною в загальну шину.

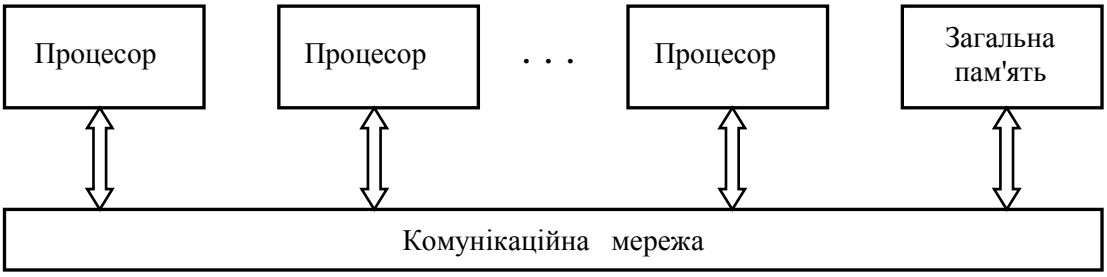


Рис. 1.6. Структура обчислювальної системи із загальною пам'яттю

Таким чином, структура ОС із загальною пам'яттю аналогічна розглянутій вище архітектурі із загальною шиною, через що їй властиві ті ж недоліки. Стосовно обчислювальних систем дана схема має додаткову перевагу: обмін інформацією між процесорами не пов'язаний з додатковими операціями і забезпечується за рахунок доступу до загальних областей пам'яті.

Альтернативний варіант організації – *розподілена* система, де загальна пам'ять взагалі відсутня, а кожен процесор володіє власною локальною пам'яттю (рис. 1.7).

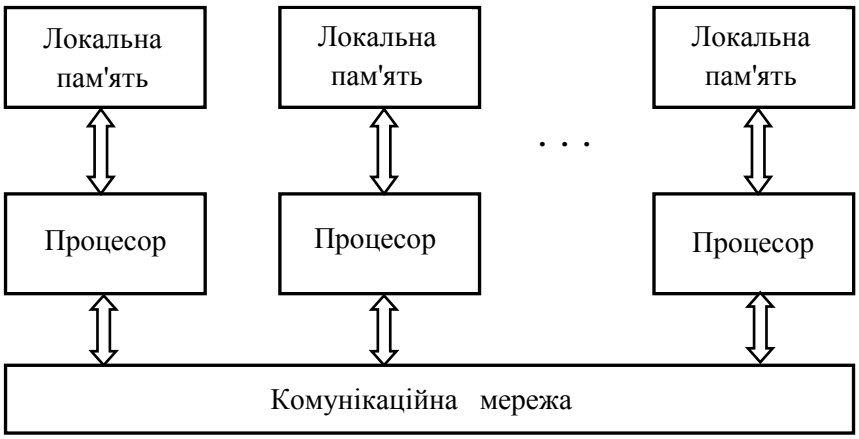


Рис. 1.7. Структура розподіленої обчислювальної системи

Часто такі системи об'єднують окремі ОМ. Обмін інформацією між складовими системи забезпечується за допомогою комунікаційної мережі шляхом обміну повідомленнями.

Подібна побудова ОС знімає обмеження, властиві для загальної шини, але приводить до додаткових витрат на пересилку повідомлень між процесорами або машинами.

Обчислювальні системи бувають однорідні і неоднорідні.

Однорідна обчислювальна система будується на базі однотипних комп'ютерів або процесорів. Однорідні системи дозволяють використовувати стандартні набори технічних, програмних засобів, стандартні протоколи поєднання пристроїв. Тому їх організація значно простіша, полегшується обслуговування систем та їх модернізація.

Неоднорідна обчислювальна система включає у свій склад різні типи комп'ютерів або процесорів. Під час побудови системи приходиться

враховувати їх різні технічні і функціональні характеристики, що значно ускладнює створення та обслуговування неоднорідних систем.

Розрізняють обчислювальні системи з *централізованим* і *децентралізованим* управлінням. У першому випадку управління виконує виділений комп'ютер або процесор, в другому – ці компоненти рівноправні і можуть брати управління на себе.

Крім того, обчислювальні системи можуть бути:

- *територіально-зосередженими* (усі компоненти розташовуються у безпосередній близькості один від одного);
- *розподіленими* (компоненти розташовуються на значній відстані, наприклад, обчислювальні мережі);
- *структурно-однорівневими* (є тільки один загальний рівень обробки даних);
- *багаторівневими* (ієрархічними) структурами. В ієрархічних обчислювальних системах комп'ютери або процесори розподілені по різних рівнях обробки інформації, деякі з них можуть бути орієнтовані на виконання певних функцій.

Нарешті, як уже відмічалося, обчислювальні системи діляться на такі: одномашинні, багатомашинні, багатопроцесорні.

Слід відзначити, що архітектура обчислювальних систем постійно вдосконалюється. Еволюцію архітектур визначають різні фактори, головні з яких показані на рис. 1.8.

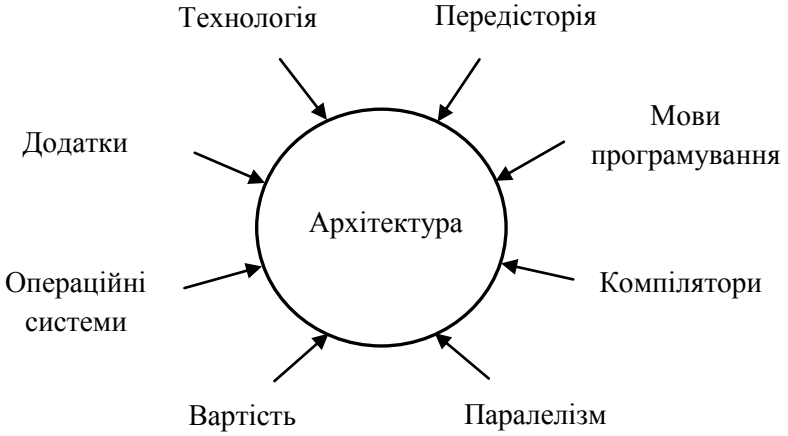


Рис. 1.8. Фактори, які визначають розвиток архітектури обчислювальних систем

Кожна зміна в архітектурі спрямована на абсолютне підвищення продуктивності або принаймні на більш ефективне розв'язання задач певного класу. Не зменшуючи ролі жодного з них, слід визнати, що найбільш очевидні успіхи в галузі засобів обчислювальної техніки зв'язані з технологічними досягненнями.

Основні напрямки досліджень у галузі архітектури обчислювальних систем можна умовно розділити на дві групи: еволюційні і революційні.

До *першої групи* слід віднести дослідження, метою яких є вдосконалення методів реалізації вже досягнутих відомих ідей. В основному вони зв'язані з вдосконаленням архітектури мікропроцесорів (МП). Найбільш значимі із змін в архітектурі МП зв'язані з підвищенням рівня паралелізму на рівні команд (можливості одночасного виконання декількох команд). Тут у першу чергу слід відзначити конвеєризацію, суперскалярну обробку і архітектуру з командними словами надвеликої довжини (VLIW).

Друга група досліджень спрямована на створення абсолютно нових архітектур, які принципово відрізняються від традиційної фон-нейманівської архітектури (нетрадиційні архітектури).

КОНТРОЛЬНІ ПИТАННЯ

1. За якими ознаками виділяють покоління обчислювальних машин?
2. Поясніть визначальні ідеї для кожного з етапів еволюції обчислювальної техніки.
3. Охарактеризуйте основні принципи фон-нейманівської концепції обчислювальної машини.
4. Поясніть призначення та структуру команди.
5. Охарактеризуйте кожний такт циклу команди.
6. Які вузли ВМ беруть участь в реалізації етапу вибірки команди?
7. Дайте коротку характеристику основних компонентів структури IBM PC-сумісного комп'ютера.
8. Які пристрої є ядром комп'ютера? Назвіть їх призначення.
9. Яку функцію виконує системна шина?
10. Назвіть призначення та склад системної плати.
11. Дайте характеристику шині розширення.
12. Яку функцію виконує Northern Bridge (Північний міст)?
13. Яку функцію виконує Southern Bridge (Південний міст)?
14. Які ознаки покладені в основу класифікації обчислювальних машин?
15. Наведіть найважливіщі характеристики різних класів ОМ.
16. Оцініть переваги і недоліки архітектури обчислювальних машин з безпосередніми зв'язками і загальною шиною.
17. Дайте характеристику різних типів обчислювальних систем.

2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНИХ СИСТЕМ

2.1. ТИПИ І ФОРМАТИ КОМАНД

2.1.1. Типи команд

Незважаючи на відмінність у системах команд різних ОМ, деякі основні типи операцій можуть бути знайдені в будь-якій з них. Для опису цих типів приймемо таку класифікацію:

- команди пересилки даних;
- команди арифметичної і логічної обробки;
- команди роботи з рядками;
- команди SIMD;
- команди перетворення;
- команди вводу/виводу;
- команди управління потоком команд.

Команди пересилки. Дана група команд забезпечує передачу інформації між процесором і оперативною пам'яттю (ОП), усередині процесора і між елементами пам'яті. Пересильні операції усередині процесора мають тип «регістр-регістр». Передачі між процесором і пам'яттю відносяться до типу «регістр-пам'ять», а пересилки в пам'яті – до типу «пам'ять-пам'ять».

Команди арифметичної і логічної обробки. До даної групи входять команди, що забезпечують арифметичну і логічну обробку інформації в різних формах її уявлення. Для кожної форми подання чисел в архітектурі системи команд (АСК) зазвичай передбачається якийсь стандартний набір операцій.

Крім обчислення результату виконання арифметичних і логічних операцій супроводжується формуванням в АЛП ознак (прапорів), що характеризують цей результат. Найчастіше фіксуються такі ознаки: **Z** (**Z**ero) – нульовий результат; **N** (**N**egative) – негативний результат; **V** (o**V**erflow) – переповнювання розрядної сітки; **C** (**C**arry) – наявність перенесення.

Команди для роботи з рядками. Для роботи з рядками в АСК зазвичай передбачаються команди, що забезпечують переміщення, порівняння і пошук рядків. У більшості машин перераховані операції просто імітуються за рахунок інших команд.

SIMD-команди. Назва даного типу команд є абревіатурою від Single Instruction Multiple Data – буквально «одна інструкція – багато даних». На відміну від звичайних команд, що оперують двома числами, SIMD – команди

обробляють відразу дві групи чисел (у принципі їх можна називати груповими командами). Операнди таких команд зазвичай подані в одному з упакованих форматів.

Ідея SIMD – обробки була висунута в Інституті точної механіки і обчислювальної техніки ім. С.А. Лебєдєва в 1978 році в рамках проекту «Ельбрус-1». З 1992 року команди типу SIMD стають невід'ємним елементом АСК мікропроцесорів фірм Intel і AMD. Приводом послужило широке розповсюдження мультимедійних застосувань. Відео, тривимірна графіка і звук у ОМ подаються великими масивами даних, елементи яких найчастіше обробляються ідентично. Так, у процесі стиснення відео і перетворення його у формат MPEG один і той же алгоритм застосовується до тисяч бітів даних. У тривимірній графіці часто зустрічаються операції, які можна виконати за один такт (інтерполяція і нормування векторів і так далі). Включення SIMD – команд в АСК дозволяє істотно прискорити подібні обчислення.

Команди перетворення. Команди перетворення здійснюють зміну формату подання даних. Прикладом може служити перетворення з десяткової системи числення в двійкову або переклад 8-розрядного коду символу з кодування ASCII в кодування EBCDIC, і навпаки.

Команди вводу/виводу. Команди цієї групи можуть бути підрозділені на команди управління периферійним пристроєм (ПП), перевірки його стану, вводу і виводу.

Команди *управління периферійним пристроєм* служать для запуску ПП і вказівки йому необхідної дії. Наприклад, накопичувачу на магнітній стрічці може бути наказано на необхідність перемотування стрічки або її просування вперед на один запис. Трактування подібних інструкцій залежить від типу ПП.

Команди *перевірки стану вводу/виводу* застосовуються для тестування різних ознак, що характеризують стан модуля Вв/Вив і підключених до нього ПП. Завдяки цим командам центральний процесор може з'ясувати, чи включено живлення ПП, чи завершена попередня операція вводу/виводу, чи виникли в процесі вводу/виводу які-небудь помилки і тому подібне. Власне обмін інформацією з ПП забезпечують команди вводу і виводу. Команди вводу наказують модулю Вв/Вив отримати елемент даних (байт або слово) ПП і помістити його на шину даних, а команди виводу – примушують модуль Вв/Вив прийняти елемент даних з шини даних і переслати його ПП.

Команди управління системою. Команди, що входять до цієї групи, є привілейованими і можуть виконуватися, тільки коли центральний процесор ОМ знаходиться в привілейованому стані або виконує програму, що знаходиться в привілейованій області пам'яті (зазвичай привілейований режим використовується лише операційною системою).

Команди управління потоком команд. Концепція фон-нейманівської обчислювальної машини припускає, що команди програми, як правило, виконуються в порядку їх розташування в пам'яті. Для отримання адреси чергової команди досить збільшити вміст лічильника команд на довжину поточної команди. В той же час основні переваги ОМ полягають саме в можливості зміни ходу обчислень залежно від результатів, що виникають в процесі рахунку. З цією метою в АСК обчислювальної машини включаються команди, що дозволяють порушити природний порядок проходження, і передати управління в іншу точку програми. В адресній частині таких команд міститься адреса точки переходу. Перехід реалізується шляхом завантаження адреси точки переходу в лічильник команд.

В системі команд ОМ можна виділити три типи команд, здатних змінити послідовність обчислень:

- безумовні переходи;
- умовні переходи (галуження);
- виклики процедур і повернення з процедур.

Команда безумовного переходу забезпечує перехід за заданою адресою без перевірки яких-небудь умов. Незважаючи на те, що присутність у програмі великого числа *команд безумовного переходу* вважається ознакою поганого стилю програмування, такі команди обов'язково входять в АСК будь-яких ОМ.

Умовний перехід відбувається тільки за дотримання певної умови, інакше виконується наступна по порядку команда програми. Більшість виробників ОМ у своїх асемблерах позначають подібні команди словом *branch* (галуження).

Умовою, на підставі якої здійснюється перехід, найчастіше виступають ознаки результату попередньої арифметичної або логічної операції. Кожна з ознак фіксується в своєму розряді регістра прапорів процесора. Можливий і інший підхід, коли рішення про перехід приймається залежно від стану одного з регістрів загального призначення, куди заздалегідь поміщається результат операції порівняння. Третій варіант – це об'єднання операцій порівняння і переходу в одній команді.

У системі команд ОМ для кожної ознаки результату передбачається своя команда галуження (іноді – дві: перехід за наявності ознаки і перехід за його відсутності). Велика частина умовних переходів пов'язана з перевіркою взаємного співвідношення двох величин або з рівністю (нерівністю) деякої величини нулю. Останній вид перевірок використовується в програмах найінтенсивніше.

Однією з форм команд умовного переходу є *команди пропуску*. У них адреса переходу відсутня, а під час виконання умови відбувається пропуск наступної команди, тобто передбачається, що відсутня в команді адреса наступної команди еквівалентна адресі поточної команди, збільшеної на

довжину команди, що пропускається. Такий прийом дозволяє скоротити довжину команд передачі управління.

Для всіх мов програмування характерне інтенсивне використання механізму процедур.

Процедурний механізм базується на *командах виклику процедури*, що забезпечують перехід з поточної точки програми до початкової команди процедури, і на *командах повернення з процедури*, для повернення в точку, безпосередньо розташовану за командою виклику. Такий режим припускає наявність засобів для збереження поточного стану вмісту лічильника команд у момент виклику (запам'ятовування адреси точки повернення) і його відновлення під час виходу з процедури.

2.1.2. Формати команд

Типова команда, в загальному випадку, повинна указувати:

- операцію, що підлягає виконанню;
- адреси початкових даних (операндів), над якими виконується операція;
- адресу, по якій повинен бути поміщений результат операції.

Відповідно до цього команда складається з двох частин: операційної і адресної (рис. 2.1).

Рис. 2.1. Структура команди

Формат команди визначає її структуру, тобто кількість двійкових розрядів, що відводяться під всю команду, а також кількість і розташування окремих полів команди. *Полем* називається сукупність двійкових розрядів, що кодують складову частину команди. Вибір формату команди впливає на багато характеристик ОМ. Оцінюючи можливі формати, потрібно враховувати такі чинники:

- загальне число різних команд;
- загальну довжину команди;
- тип полів команди (фіксованої або змінної довжини) і їх довжина;
- простоту декодування;
- адресованість і способи адресації;
- вартість устаткування для декодування і виконання команд.

Довжина команди. Це найважливіша обставина, що впливає на організацію і ємність пам'яті, структуру шин, складність і швидкодію ЦП. З одного боку, зручно мати в розпорядженні могутній набір команд, тобто якомога більше кодів операцій, операндів, способів адресації, і максимальний

адресний простір. Проте все це вимагає виділення більшої кількості розрядів під кожне поле команди, що приводить до збільшення її довжини. Разом з тим, для прискорення вибірки з пам'яті бажано, щоб команда була якомога коротша, а її довжина була рівна або кратна ширині шини даних. Для спрощення апаратури і підвищення швидкодії ОМ довжину команди зазвичай вибирають кратною байту, оскільки в більшості ОМ основна пам'ять організована у вигляді 8-бітових комірок. В рамках системи команд одної ОМ можуть використовуватися різні формати команд. Зазвичай це зв'язано із застосуванням різних способів адресації. У такому разі в склад кода команди вводиться поле для завдання способу адресації (СА), і узагальнений формат команди набуває вигляд, показаний на рис. 2.2.

Рис. 2.2. Узагальнений формат команды

Загальна довжина команди $R_к$ може бути описана таким співвідношенням:

$$R_K = \sum_{i=1}^{l} R_{Ai} + R_{KOn} + R_{CA} \ , \qquad (2.1)$$

де l – кількість адрес у команді; R_{Ai} – кількість розрядів для запису *i-ї* адреси; R_{Kon} – розрядність поля коду операції; R_{CA} – розрядність поля способу адресації.

Розрядність полів команди. Як уже мовилося, в будь-якій команді можна виділити операційну і адресну частини. Довжини відповідних полів визначаються різними чинниками, які доцільно розглянути окремо.

Розрядність поля коду операції. Кількість двійкових розрядів, що відводяться під код операції, вибирається так, щоб можна було подати будь-яку з операцій. Якщо система команд припускає N_{KOn} операцій, то мінімальна розрядність поля коду операції R_{KOn} визначається таким чином:

$$R_{KOn} = \text{int}(\log_2 N_{KOn}), \qquad (2.2)$$

де *int* означає округлення у більшу сторону до цілого числа.

Коли задана довжина коду команди, доводиться шукати компроміс між розрядністю поля коду операції і адресного поля. Більшу кількість можливих операцій припускає довге поле коду операції, що веде до скорочення адресного поля, тобто до звуження адресного простору. Для усунення цієї суперечності іноді довжину поля коду операції варіюють. Спочатку під код операції відводиться якесь фіксоване число розрядів, проте для окремих команд це поле розширюється за рахунок декількох бітів, які віднімаються від адресного поля.

Так, наприклад, може бути збільшене число різних команд пересилки даних. Необхідно відзначити, що «урізування» частини адресного поля веде до скорочення можливостей адресації, і подібний прийом рекомендується тільки в тих командах, де подібне скорочення може бути виправданим.

Розрядність адресної частини. В адресній частині команди міститься інформація про місцезнаходження початкових даних і місце збереження результату операції. Звичайне місцезнаходження кожного з операндів і результату задається в команді шляхом вказівки адреси відповідної комірки основної пам'яті або номера регістра процесора. Принципи використання інформації з адресної частини команди визначає *система адресації*. Система адресації задає *число адрес* в команді і прийняті *способи адресації*.

Розрядності полів R_{Ai} і R_{CA} розраховуються за формулами:

$$R_{Ai} = \text{int}(\log_2 N_i); \quad (2.3)$$

$$R_{CA} = \text{int}(\log_2 N_{CA}), \quad (2.4)$$

де *Ni* – кількість комірок пам'яті, до якого можна звернутися за допомогою *i-ї* адреси; N_{CA} – кількість способів адресації.

Кількість адрес у команді. Для визначення кількості адрес, що включаються в адресну частину, використовуватимемо термін *адресність*. У «максимальному» варіанті необхідно вказати три компоненти: адреса першого операнда, адреса другого операнда і адреса комірки, куди заноситься результат операції. У принципі може бути додана ще одна адреса, що вказує місце зберігання наступної інструкції. У результаті має місце *чотириадресний формат команди* (рис. 2.3). Такий формат підтримувався у ОМ EDVAC, розробленою в 1940-х роках.

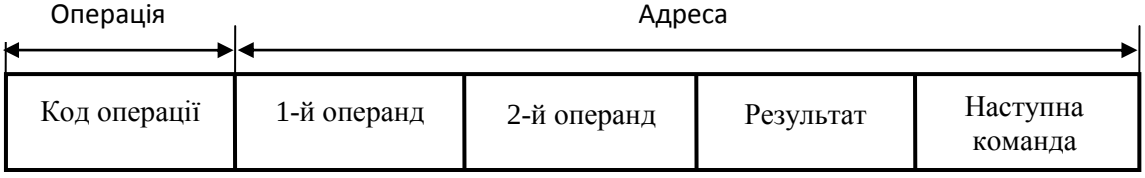


Рис. 2.3. Чотириадресний формат команди

У фон-нейманівських ОМ необхідність у четвертій адресі відпадає, оскільки команди розташовуються в пам'яті в порядку їх виконання, і адреса чергової команди може бути отримана за рахунок простого збільшення адреси поточної команди в лічильнику команд. Це дозволяє перейти до *триадресного формату команди* (рис. 2.4).

На жаль, і в триадресному форматі довжина команди може виявитися дуже великою. Так, якщо адреса комірки основної пам'яті має довжину 32 біта, а довжина коду операції – 8 біт, то довжина команди складе 104 біти (13 байт).

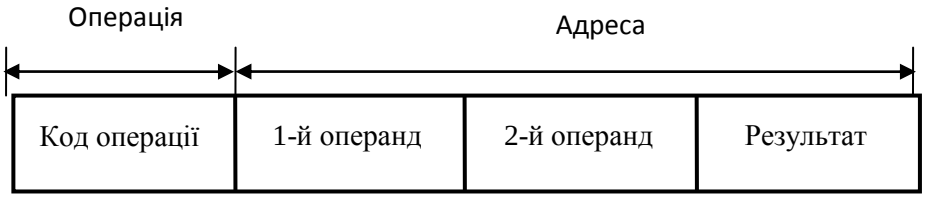


Рис. 2.4. Триадресний формат команди

Якщо за умовчанням взяти за адресу результату адресу одного з операндів (зазвичай другого), то можна обійтися без третьої адреси, і у результаті отримуємо *двоадресний формат команди* (рис. 2.5). Природно, що в цьому випадку відповідний операнд після виконання операції втрачається.

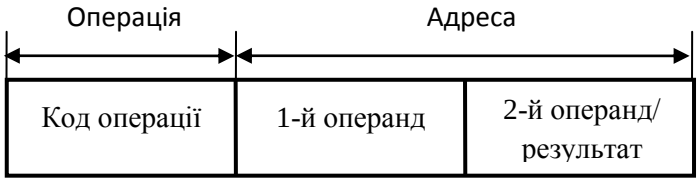


Рис. 2.5. Двоадресний формат команди

Команду можна ще більш скоротити, перейшовши до одноадресного формату (рис. 2.6), що можливо у випадку виділення певного стандартного місця для зберігання першого операнда і результату. Зазвичай для цієї мети використовується спеціальний регістр центрального процесора (ЦП), відомий під назвою акумулятора.

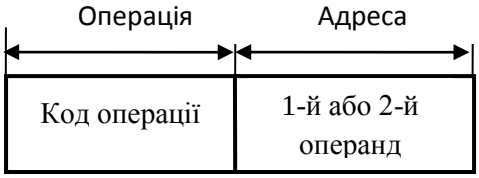


Рис. 2.6. Одноадресний формат команди

Застосування єдиного регістра для зберігання одного з операндів і результату є обмежуючим чинником, тому крім акумулятора часто використовують і інші регістри ЦП. Оскільки число регістрів в ЦП невелике, для вказівки одного з них у команді досить мати порівняно коротке адресне поле. Відповідний формат носить назву *півтораадресного* або *регістрового формату* (рис. 2.7).

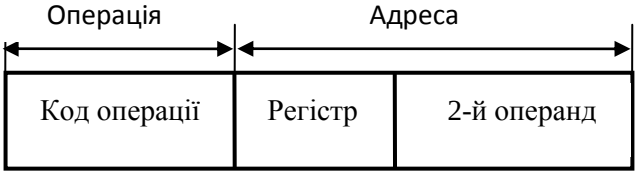


Рис. 2.7. Півтораадресний формат команди

Нарешті, якщо для обох операндів вказати чітко задане місцеположення, а також у разі команд, що не вимагають операнда, можна отримати *нульадресный формат команди* (рис. 2.8).

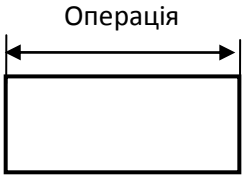


Рис. 2.8. Нульадресний формат команди

У такому варіанті адресна частина команди взагалі відсутня або не задіюється.

Вибір адресності команд. Під час вибору кількості адрес в адресній частині команди зазвичай керуються такими критеріями:

ємністю запам'ятовуючого пристрою, яка потрібна для зберігання програми; часом виконання програми; ефективністю використання комірок пам'яті під час зберігання програми.

Для оцінки впливу адресності на кожен з перерахованих елементів скористаємося методикою і виводами, викладеними в [25].

Адресність і ємність запам'ятовуючого пристрою. Якщо позначити кількість команд у програмі – N_A, розрядність команди, визначувану відповідно до формули (2.1) – R_{KA}, індекс, вказуючий адресність команд програми – A, то ємність запам'ятовуючого пристрою для зберігання програми E_A можна оцінити із співвідношення

$$E_A = N_A \times R_{KA}.$$

З цих позицій оптимальна адресність команди визначається шляхом розв'язання рівняння

$$\frac{\partial E_A}{\partial A} = 0$$

за умови, що знайдене значення забезпечує мінімум E_A. У [25] показано, що в середньому E_A монотонно зростає із збільшенням A. Таким чином, *під час вибору кількості адрес по критерію «ємність ЗП» перевагу слід віддавати одноадресним командам.*

Адресність і час виконання програми. Час виконання однієї команди складається з часу виконання операції і часу звернення до пам'яті.

Для триадресної команди останнє підсумовується з чотирьох складових часу:

- *вибірки команди;*
- *вибірки першого операнда;*
- *вибірки другого операнда;*
- *запису в пам'ять результату.*

Одноадресна команда вимагає двох звернень до пам'яті:

- *вибірки команди;*
- *вибірки операнда.*

Як видно, на виконання одноадресної команди витрачається менше часу, чим на обробку триадресної команди, проте для реалізації однієї триадресної команди, як правило, потрібно три одноадресних.

Цих міркувань проте недостатньо, щоб однозначно віддати перевагу тому або іншому варіанту адресності. Визначаючим під час вибору є тип алгоритмів, на переважну реалізацію яких орієнтована конкретна ОМ. Можливі типи алгоритмів умовно розділимо на три групи: *послідовні; паралельні; комбіновані.*

Для *послідовного алгоритму* результат попередньої команди використовується в подальшій. Потрібна всього одна команда попереднього засилання числа в суматор (акумулятор) на початку обчислення і одна команда пересилки результату в пам'ять в кінці обчислень. У послідовних алгоритмах найбільш вигідними виявляються одноадресні команди.

У *паралельному алгоритмі* результат попередньої команди не використовується в подальшій і повинен бути відісланий в пам'ять. У цьому випадку доцільно орієнтуватися на триадресні команди.

У *комбінованому алгоритмі* обчислювальний процес утворюють як послідовні, так і паралельні частини. Тут найбільш переважними є одноадресні команди.

Двоадресні команди в плані часу реалізації алгоритмів займають проміжне положення між одноадресними і триадресними. Дещо кращі показники дають півтораадресні команди, в яких, з одного боку, зберігаються переваги одноадресних команд для послідовних алгоритмів, а з другого – підвищується ефективність реалізації паралельних і комбінованих алгоритмів.

Враховуючи вищевикладене, перевагу слід віддати одноадресним командам.

2.2. ТИПИ І ФОРМАТИ ОПЕРАНДІВ

Машинні команди оперують даними, які в цьому випадку прийнято називати *операндами*. До найбільш загальних (базових) типів операндів можна віднести: адреси, числа, символи і логічні дані. Крім них ОМ забезпечує обробку і складніших інформаційних одиниць: графічних зображень, аудіо-, відео- і анімаційної інформації. Така інформація є похідною від базових типів даних і зберігається у вигляді файлів на зовнішніх запам'ятовуючих пристроях. Для кожного типу даних у ОМ передбачені певні формати.

2.2.1. Числова інформація

Серед цифрових даних можна виділити дві групи:

- цілі типи, використовувані для подання цілих чисел;
- дійсні типи для подання раціональних чисел.

В рамках першої групи є декілька форматів подання чисельної інформації, залежних від її характеру. Для подання дійсних чисел використовується форма з плаваючою комою.

Числа у формі з фіксованою комою

Подання числа *X* у формі з *фіксованою комою* (ФК), яку іноді називають також *природною формою*, включає знак числа і його модуль в *q*-ічному коді. Тут *q – основа системи числення* або *база*. Для сучасних ОМ характерна двійкова система (*q* = 2), але іноді використовуються також вісімкова (*q* = 8) або шістнадцяткова (*q* = 16) системи числення. Кому в записі числа називають відповідно двійковою, вісімковою або шістнадцятковою. Знак позитивного числа кодується двійковою цифрою 0, а знак негативного числа – цифрою 1.

Числам з ФК відповідає запис вигляду $X = \pm a_{n-1}\ldots a_1 a_0 a_{-1} a_{-2}\ldots a_{-r}$. Негативні числа зазвичай подаються в додатковому коді. Розряд коду числа, в якому розміщується знак, називається *знаковим розрядом коду*. Розряди, де розташовуються значущі цифри числа, називаються *цифровими розрядами коду*. Знаковий розряд розміщується лівіше старшого цифрового розряду. Положення коми однакове для всіх чисел і в процесі вирішення завдань не міняється. Хоча кома і фіксується, в коді числа вона ніяк не виділяється, а тільки мається на увазі. У загальному випадку розрядна сітка ОМ для розміщення чисел у формі з ФК має вигляд, показаний на рис. 2.9, де *n* розрядів використовуються для запису цілої частини числа і *r* розрядів – для дробової частини.

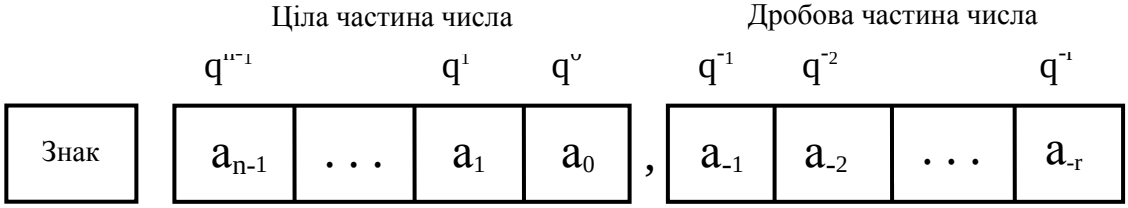


Рис. 2.9. Формат подання чисел з фіксованою комою

Якщо число є змішаним (містить цілу і дрібну частини), воно обробляються як ціле, хоча і не є таким (в цьому випадку застосовують термін *масштабоване ціле).* Обробка змішаних чисел у ОМ зустрічається украй рідко. Як правило, використовуються ОМ з дробовою(*n* = 0) або цілочисельною (*r* = 0) арифметикою.

Під час фіксації коми перед старшим цифровим розрядом можуть бути подані тільки правильні дроби.

Під час фіксації коми після молодшого розряду подаються лише цілі числа (рис. 2.10).

Це найбільш поширений спосіб, тому надалі поняття ФК зв'язуватиметься виключно з цілими числами, а операції над числами у формі з ФК характеризуватимуться як цілочисельні. Тут можливі числа із знаком і без знака.

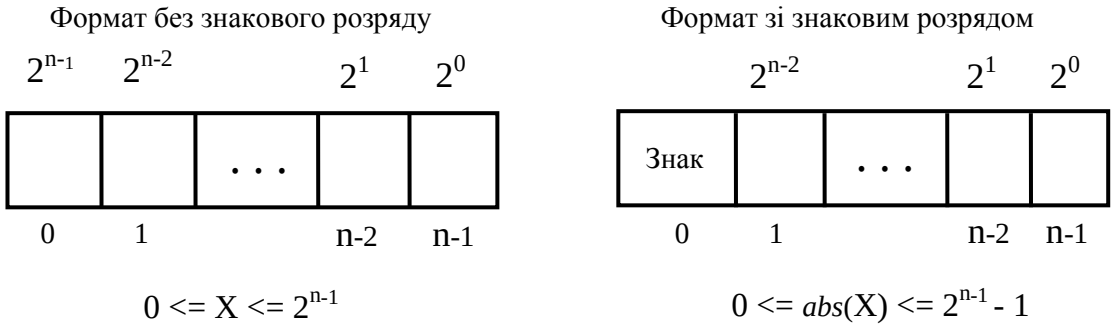


Рис. 2.10. Подання цілих чисел у форматі з ФК

На рис. 2.10 приведені цілочисельні формати з фіксованою комою, прийняті в мікропроцесорах фірми Intel.

Подання чисел у форматі з ФК спрощує апаратурну реалізацію ОМ і скорочує час виконання машинних операцій, проте під час вирішення завдань необхідно постійно стежити за тим, щоб усі початкові дані, проміжні і остаточні результати не виходили за допустимий діапазон формату, інакше можливе переповнення розрядної сітки і результат обчислень буде невірним.

Упаковані цілі числа. В АСК сучасних мікропроцесорів є команди, що оперують цілими числами, поданими в упакованому вигляді. Зв'язано це з обробкою мультимедійної інформації. Формат припускає упаковку в межах достатньо довгого слова (зазвичай 64-розрядного) декількох невеликих цілих чисел, а відповідні команди обробляють всі ці числа паралельно. Якщо кожне з чисел складається з чотирьох двійкових розрядів, то в 64-розрядне слово можна помістити до 16 таких чисел. Невикористані розряди заповнюються нулями.

У мікропроцесорах фірми Intel, починаючи з Pentium MMX, присутні спеціальні команди для обробки мультимедійної інформації (MMX - команди), що оперують цілими числами, упакованими в квадрослова (рис. 2.11).

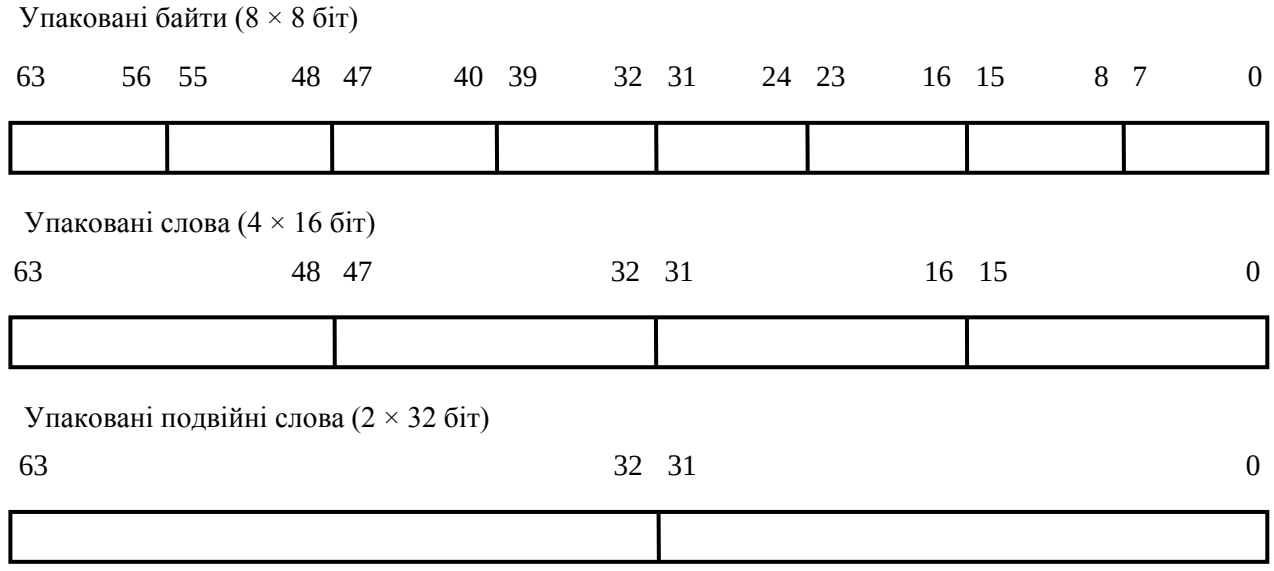


Рис 2.11. Формати упакованих цілих чисел у технологіях MMX і 3DNow!

Передбачено три формати квадрослів (64-розрядних слів): упаковані байти (вісім 8-розрядних чисел); упаковані слова (чотири 16-розрядні числа) і упаковані подвійні слова (два 32-розрядні числа).

Байти у форматі упакованих байтів нумеруються від 0 до 7, причому байт 0 розташовується в молодших розрядах квадрослова. Аналогічна система нумерації і розміщення упакованих чисел застосовується для упакованих слів (номери 0–3) і упакованих подвійних слів (номери 0–1).

Ідентичні формати упакованих даних застосовуються також в іншій технології обробки мультимедійної інформації, запропонованою фірмою AMD. Ця технологія носить назву 3DNow!, а реалізована в мікропроцесорах даної фірми.

Десяткові числа. В ряді завдань, головним чином обліково-статистичного характеру, доводиться мати справу із зберіганням, обробкою і пересилкою десяткової інформації. Особливість таких завдань полягає в тому, що оброблювані числа можуть складатися з різної і дуже великої кількості десяткових цифр. Традиційні методи обробки з перекладом початкових даних у двійкову систему числення і зворотним перетворенням результату часто зв'язані з істотними накладними витратами. З цієї причини у ОМ застосовуються інші спеціальні форми подання десяткових даних. У їх основу покладений принцип кодування кожної десяткової цифри еквівалентним двійковим числом з чотирьох бітів (тетрадою), тобто так званим двійково-десятковим кодом.

Використовуються два формати подання десяткових чисел (всі числа розглядаються як цілі): *зонний (розпакований)* і *ущільнений (упакований)*. В обох форматах кожна десяткова цифра подається двійковою тетрадою, тобто замінюється двійково-десятковим кодом (рис. 2.12).

Байт		Байт			Байт		Байт	
Зона	Цифра	Зона	Цифра	. . .	Зона	Цифра	Зона	Цифра

а

Байт		Байт			Байт		Байт	
Цифра	Цифра	Цифра	Цифра	. . .	Цифра	Цифра	Цифра	Знак

б

Рис. 2.12. Формати десяткових чисел: *а* – зонний; *б* – ущільнений

Зонний формат (рис. 2.12, а) застосовується в операціях вводу/виводу. У ньому під кожну цифру виділяється один байт, де молодші чотири розряди відводяться під код цифри, а в старшу тетраду (поле зони) записується

спеціальний код «зона», не співпадаючий з кодами цифр і знаків. Виняток становить байт, що містить молодшу цифру десяткового числа, де в полі зони зберігається знак числа.

Під час виконання операцій додавання і віднімання над десятковими числами зазвичай використовується упакований формат і в ньому ж виходить результат (множення і ділення можливе тільки в зонному форматі). В упакованому форматі (рис. 2.12, б) кожен байт містить коди двох десяткових цифр. Права тетрада останнього байта призначається для запису знака числа. Десяткове число повинне займати цілу кількість байтів. Якщо ця умова не виконується, то чотири старші двійкові розряди лівого байта заповнюються нулями.

Розміщення знака в молодшому байті, як в зонному, так і в упакованому поданнях, дозволяє задавати десяткові числа довільної довжини і передавати їх у вигляді ланцюжка байтів. В цьому випадку знак указує, що байт, в якому він міститься, є останнім байтом даного числа, а наступний байт послідовності – це старший байт чергового числа.

Числа у формі з плаваючою комою

Від недоліків ФК в значній мірі вільна форма подання чисел з плаваючою комою (ПК), відома також під назвами нормальної або напівлогарифмічної форми. У даному варіанті кожне число розбивається на дві групи цифр. Перша група цифр називається мантисою, друга – порядком. Число подається у вигляді добутку $X = \pm m\, q^{\pm p}$,

де m – мантиса числа X, p – порядок числа, q – основа системи числення.

Для подання числа у формі з ПК потрібно задати знаки мантиси і порядку, їх модулі в q-ічному коді, а також основу системи числення (рис. 2.13). Нормальна форма неоднозначна, оскільки взаємна зміна m і p приводить до «плавання» коми, чим і обумовлена назва цієї форми.

Рис. 2.13. Форма подання чисел з плаваючою комою

Діапазон і точність подання чисел з ПК залежать від числа розрядів, що відводяться під порядок і мантису.

У більшості обчислювальних машин для спрощення операцій над порядками останні приводять до цілих позитивних чисел, застосовуючи так званий зміщений порядок. Для цього до дійсного порядку додається ціле позитивне число – зсув (рис. 2.14). Наприклад, у системі із зсувом 128 порядок –3 подається як 125 (–3 + 128).

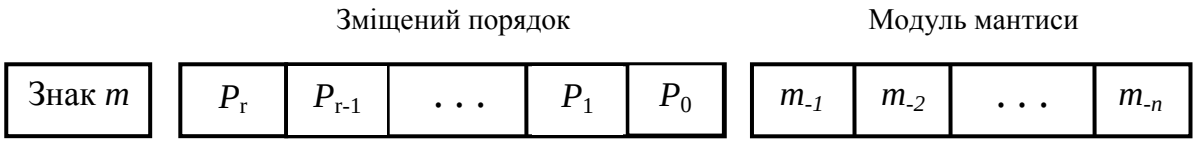


Рис. 2.14. Формат числа з ПК із зміщеним порядком

Звичайний зсув вибирається рівним половині уявного діапазону порядків. Відзначимо, що зміщений порядок займає всі біти поля порядку, у тому числі і той, який раніше використовувався для запису знака порядку.

Мантиса в числах з ПЗ зазвичай подається в *нормалізованій формі*. Це означає, що на мантису накладаються такі умови, щоб вона по модулю була менше одиниці ($|q| < 1$), а перша цифра після крапки відрізнялася від нуля. Отримана таким чином мантиса називається *нормалізованою*. Для двійкової системи числення можна записати: $X = q\ 2^{p}$, $(1 > |q| = S)$.

Якщо перші i цифр мантиси рівні нулю, для нормалізації її потрібно зсунути відносно коми на i розрядів вліво з одночасним зменшенням порядку на i одиниць.

Якщо для запису числа з ПК використовується основа 2 ($q = 2$), то часто застосовують спосіб підвищення точності подання мантиси, що називається *прийомом прихованої одиниці*. Суть його в тому, що в нормалізованій мантисі старша цифра завжди дорівнює одиниці (для подання нуля використовується спеціальна кодова комбінація), отже, ця цифра може не записуватися, а матися на увазі. Запис мантиси починають з її другої цифри, і це дозволяє задіювати додатковий значущий біт для точнішого подання числа. Слід зазначити, що значення порядку в даному випадку не міняється. Прихована одиниця перед виконанням арифметичних операцій відновлюється, а під час запису результату – віддаляється.

Для істотнішого збільшення точності обчислень під число відводять декілька машинних слів, наприклад два. Додаткові біти, як правило, служать для збільшення розрядності мантиси, проте у ряді випадків частина з них може відводитися і для розширення поля порядку. В процесі обчислень може виходити ненормалізоване число. У такому разі ОМ, якщо це наказано командою, автоматично нормалізує його.

Числа з плаваючою комою в різних ОМ мають декілька різних форматів. У даний час для всіх ОМ рекомендований стандарт, розроблений загальновизнаним міжнародним центром стандартизації IEEE (Institute of Electrical and Electronics Engineers) - IEEE 754 з метою полегшення переносу програм з одного процесора на інші.

Стандарт визначає 32-бітовий (одинарний) і 64-бітовий (подвійний) формати з 8- і 11-розрядним порядком відповідно. Основою системи числення є 2. На додаток стандарт передбачає два розширені формати, одинарний і

подвійний, фактичний вид яких залежить від конкретної реалізації. Розширені формати передбачають додаткові біти для порядку (збільшений діапазон) і мантиси (підвищена точність).

В останніх версіях АСК, що передбачають особливі команди для обробки мультимедійної інформації, крім упакованих цілих чисел використовуються і упаковані числа з плаваючою комою. Так, у вже згадуваній технології 3DNow! фірми AMD є команди, які служать для збільшення продуктивності систем під час обробки тривимірних застосувань, що описуються числами з ПК. Кожна така команда працює з двома операндами з плаваючою комою одинарної точності, які упаковуються в 64-розрядні групи.

У мікропроцесорах фірми Intel, починаючи з Pentium III, для аналогічних цілей введені команди, що реалізовують технологію SSE (Streaming SIMD Extension – потокова обробка за принципом «одна команда – багато даних»), також орієнтовану на паралельну обробку упакованих чисел з ПК. Тут числа об'єднуються в групи завдовжки 128 біт, і це дозволяє упакувати в групу чотири 32-розрядні числа з ПК (числа з одинарною точністю). Пізніше, в технології SSE2, яку можна вважати подальшим розвитком SSE, з'явився формат, де в групу з 128 біт упаковуються два 64-розрядні числа з ПК, тобто числа, подані з подвійною точністю.

Розрядність основних форматів числових даних і розміщення їх у пам'яті

Дані, що подають у ОМ числову інформацію, можуть мати *фіксовану* або *змінну* довжину. Операційні пристрої обчислювальних машин (цілочисельні арифметико-логічні пристрої, блоки обробки чисел з плаваючою комою, пристрою десяткової арифметики і т. п.), як правило, розраховані на обробку коду фіксованої довжини.

Найменшою одиницею даних у ОМ служить біт (BIT, BInary digiT - двійкова цифра). В більшості випадків ця одиниця інформації дуже мала. Однобітові операційні пристрої використовувалися в ОМ з послідовною обробкою інформації, а в сучасних машинах з паралельною обробкою розрядів вони практично не застосовуються. Побітову роботу з даними швидше можна зустріти в багатопроцесорних обчислювальних системах, побудованих з однорозрядних процесорів.

Реально найменшою оброблюваною одиницею вважається байт, що складається з восьми бітів. На практиці ця одиниця інформації також виявляється недостатньою, і значно частіше застосовуються числа, подані двома (півслово), чотирма (слово), вісьма (подвійне слово) або шістнадцятьма (зчетверене слово) байтами.

Розрядність цілочисельного АЛП зазвичай вибирається рівною ширині адреси (для більшості сучасних ОМ це 32 розряди). Отже, найбільш вигідними

в плані швидкодії є такі цілі числа, довжина яких збігається з розрядністю адреси. Використання коротших чисел дозволяє заощадити на пам'яті, але виграшу в продуктивності не дає.

Блоки операцій з плаваючою комою зазвичай узгоджені із стандартом IEEE 754 і розраховані на обробку чисел у форматі подвійної довжини (64 біти). В більшості ОМ реальна розрядність таких блоків навіть більше (80 біт). Таким чином, якнайкращим варіантом під час проведення обчислень з плаваючою комою можна вважати формат подвійного слова. Під час вибору формату меншої довжини (32 розряди) обчислення все одно ведуться з більшою точністю, після чого результат округляється. Таким чином, використання короткого формату чисел з плаваючою комою, як і у разі цілих чисел з фіксованою комою, крім економії пам'яті ніяких інших переваг також не дає.

В сучасних ОМ розрядність однієї комірки пам'яті, як правило, рівна одному байту (8 біт). В той же час реальна довжина кодів чисел складає 2, 4, 8 або 16 байт. У разі зберігання таких чисел у пам'яті послідовні байти числа розміщують в декількох комірках з послідовними адресами, при цьому для доступу до числа указується тільки найменша з адрес. Під час розробки архітектури системи команд необхідно визначити порядок розміщення байтів у пам'яті, тобто якому з байтів (старшому або молодшому) відповідатиме ця найменша адреса.

В обчислювальному плані обидва способи запису рівноцінні. Так, фірми DEC і Intel віддають переваги розміщенню в першій комірці молодшого байта, а IBM і Motorola орієнтуються на протилежний варіант. Вибір зазвичай пов'язаний з якимись іншими міркуваннями розробників ОМ. В даний час в більшості машин передбачається використання обох варіантів. Крім порядку розміщення байтів, істотним буває і вибір адреси, з якої може починатися запис числа. Зв'язано це з фізичною реалізацією напівпровідникових запам'ятовуючих пристроїв, де зазвичай передбачається можливість читання (запису) чотирьох байтів підряд. Причому дана операція виконується швидше, якщо адреса першого байта A відповідає умові $A \bmod S = 0$, (S = 2, 4, 8, 16). Числа, розміщені в пам'яті відповідно до цього правила, називаються *такими, що вирівнюються*.

2.2.2. Інші види інформації

Символьна інформація. У загальному об'ємі обчислювальних дій все більша частка приходиться на обробку символьної інформації, що містить букви, цифри, розділові знаки, математичні та інші символи. Кожному символу ставиться у відповідність певна двійкова комбінація. Сукупність можливих символів і призначених їм двійкових кодів утворює *таблицю кодування*. В даний час застосовується багато різних таблиць кодування. Об'єднує їх ваговий принцип, за якого ваги кодів цифр зростають у міру збільшення цифри, а ваги

символів збільшуються в алфавітному порядку. Так, вага букви «Б» на одиницю більше ваги букви «А». Це сприяє спрощенню обробки в ОМ. Найбільш поширеними є кодові таблиці, в яких символи кодуються за допомогою восьмирозрядних двійкових комбінацій (байтів), що дозволяють подати 256 різних символів:

- розширений двійково-кодований код EBCDIC (Extended Binary Coded Decimal Interchange Code);
- американський стандартний код для обміну інформацією ASCII (American Standard Code for Information Interchange).

Код EBCDIC використовується як внутрішній код в універсальних ОМ фірми IBM. Він же відомий під назвою ДКОІ (двійковий код для обробки інформації).

Стандартний код ASCII – 7-розрядний, восьма позиція відводиться для запису біта парності. Це забезпечує уявлення 128 символів, включаючи всі латинські букви, цифри, знаки основних математичних операцій і знаки пунктуації. Пізніше з'явилася європейська модифікація ASCII, звана Latin 1 (стандарт ISO 8859-1). У ній «корисно» використовуються всі 8 розрядів. Додаткові комбінації (коди 128-255) в новому варіанті відводяться для подання специфічних букв алфавітів західноєвропейских мов, символів псевдографіки, деяких букв грецького алфавіту, а також ряду математичних і фінансових символів. Саме ця кодова таблиця вважається світовим стандартом де-факто, який застосовується з різними модифікаціями у всіх країнах.

Хоча код ASCII достатньо зручний, він все ж таки дуже тісний і не вміщає багатьох необхідних символів. З цієї причини в 1993 році консорціумом компаній Apple Computer, Microsoft, Hewlett-Packard, DEC і IBM був розроблений 16-бітовий стандарт ISO 10646, що визначає універсальний набір символів (UCS, Universal Character Set). Новий код, відомий під назвою Unicode, дозволяє задати до 65 536 символів, тобто дає можливість одночасно подавати символи всіх основних «живих» і «мертвих» мов. Для букв російської мови виділені коди 1040-1093.

Логічні дані. Елементом логічних даних є логічна (булева) змінна, яка може приймати лише два значення: «істина» або «неправда». Кодування логічного значення прийнято здійснювати бітом інформації: одиницею кодують дійсне значення, нулем – помилкове. Як правило, в ОМ оперують наборами логічних змінних завдовжки в машинне слово. Обробляються такі слова за допомогою команд логічних операцій (І, АБО, НІ і т. д.), при цьому всі біти обробляються однаково, але незалежно один від одного, тобто ніяких перенесень між розрядами не виникає.

Рядки. Рядки – це безперервна послідовність бітів, байтів, слів або подвійних слів. *Бітовий рядок* може починатися в будь-якої позиції байта і

містити до 2^{32} біт. *Байтовий рядок* може складатися з байтів, слів або подвійних слів. Довжина такого рядка варіюється від нуля до 2^{32}-1 байт (4 Гбайт). Приведені цифри характерні для переважаючих у даний час 32-розрядних ОМ.

Якщо байтами байтового рядка є коди символів, то говорять про *текстовий рядок*. Оскільки довжина текстового рядка може мінятися в дуже широких межах, то для вказівки кінця рядка в останній байт заноситься код-обмежувач – звичайно це нулі у всіх розрядах байта. Іноді замість обмежувача довжину рядка указують числом, розташованим в першому байті (двох) рядка.

Відеоінформація. Відеоінформація буває як статичною, так і динамічною. Статична відеоінформація включає текст, малюнки, графіки, креслення, таблиці та ін. Малюнки діляться також на плоскі – двовимірні і об'ємні – тривимірні. Динамічна відеоінформація – це відео-, мульт- і слайд-фільми. В їх основі лежить послідовне експонування на екрані в реальному масштабі часу окремих кадрів відповідно до сценарію. Динамічна інформація використовується або для передачі рухомих зображень (анімація), або для послідовної демонстрації окремих кадрів (слайд-фільми).

В обчислювальній техніці існує два способи подання графічних зображень: *матричний (растровий)* і *векторний*. Матричні (bitmap) формати добре підходять для зображень із складною гамою кольорів, відтінків і форм, таких як фотографії, малюнки, відскановані дані. Векторні формати більш пристосовані для креслень і зображень з простими формами, тінями і забарвленням.

У матричних форматах зображення подається прямокутною матрицею точок – *пікселів* (picture element), положення яких у матриці відповідає координатам точок на екрані. Крім координат кожен піксел характеризується своїм кольором, кольором фону або градацією яскравості. Кількість бітів, що виділяються для вказівки кольору піксела, змінюється залежно від формату. У високоякісних зображеннях колір піксела описують 24 бітами, що дає близько 16 млн. кольорів. Основний недолік матричної (растрової) графіки полягає у великій ємності пам'яті, потрібної для зберігання зображення. У даний час існує множина форматів графічних файлів, що розрізняються алгоритмами стиснення і способами подання матричних зображень, а також сферою застосування.

Векторне подання, на відміну від матричної графіки, визначає опис зображення не пікселами, а кривими – *сплайнами*. Сплайн– це гладка крива, яка проходить через дві або більше опорні точки, що керують формою сплайна.

Основні переваги векторної графіки:

- опис об'єкта є простим і займає мало пам'яті;
- простота масштабування зображення без погіршення його якості;
- незалежність ємності пам'яті, потрібної для зберігання зображення, від вибраної колірної моделі.

Недоліком векторних зображень є їх деяка штучність, що полягає в тому, що будь-яке зображення необхідно розбити на кінцеву множину складових його примітивів. Примітив – ряд простих об'єктів, наприклад еліпс, прямокутник, лінія, які служать основою для створення складніших зображень.

Матрична і векторна графіка існують не відособлено одна від одної. Так, векторні малюнки можуть включати і матричні зображення. Крім того, векторні і матричні зображення можуть бути перетворені один в одного. Графічні формати, що дозволяють поєднувати матричний і векторний опис зображення, називаються *метафайлами*. Метафайли забезпечують достатню компактність файлів із збереженням високої якості зображення.

Аудіоінформація. Поняття *аудіо* пов'язане із звуками, які здатне сприймати людське вухо. Частоти аудіосигналів лежать в діапазоні від 15 Гц до 20 кГц, а сигнали за своєю природою є безперервними (аналоговими). Перш ніж бути поданою у ОМ, аудіоінформація повинна бути перетворена в цифрову форму (оцифрована). Для цього значення звукових сигналів (вибірки, samples), узяті через малі проміжки часу, за допомогою аналого-цифрових перетворювачів (АЦП) переводяться в двійковий код. Зворотна дія виконується цифро-аналоговими перетворювачами (ЦАП). Чим частіше проводяться вибірки, тим вище може бути точність подальшого відтворення початкового сигналу, але тим більша ємність пам'яті потрібна для зберігання оцифрованого звуку.

2.3. СПОСОБИ АДРЕСАЦІЇ ОПЕРАНДІВ

Питання про те, яким чином в адресному полі команди може бути вказане місцеположення операндів, вважається одним з центральних при розробці архітектури ОМ. З погляду скорочення апаратурних витрат очевидне прагнення розробників зменшити довжину адресного поля при збереженні можливостей доступу до всього адресного простору. З другого боку, спосіб завдання адрес повинен сприяти максимальному зближенню конструктив мов програмування високого рівня і машинних команд. Все це привело до того, що в архітектурі системи команд будь-якої ОМ передбачені різні способи адресації операндів.

Приступаючи до розгляду способів адресації, спочатку визначимо поняття «виконавчий» і «адресний код».

Виконавчою адресою операнда ($A_в$) називається двійковий код номера комірки пам'яті, яка служить джерелом або приймачем операнда. Цей код подається на адресні входи запам'ятовуючого пристрою (ЗП), і по ньому відбувається фактичне звернення до вказаної комірки. Якщо операнд зберігається не в основній пам'яті, а в регістрі процесора, його виконавчою адресою буде номер регістра.

Адресний код команди ($A_к$) – це двійковий код в адресному полі команди, з якого необхідно сформувати виконавчу адресу операнда.

У сучасних ОМ виконавча адреса і адресний код, як правило, не збігаються, і для доступу до даних потрібне відповідне перетворення. *Спосіб адресації* – це спосіб формування виконавчої адреси операнда за адресним кодом команди. Спосіб адресації істотно впливає на параметри процесу обробки інформації. Одні способи дозволяють збільшити ємність пам'яті, що адресується, без подовження команди, але знижують швидкість виконання операції, інші – прискорюють операції над масивами даних, треті – спрощують роботу з підпрограмами і так далі. В сьогоднішніх ОМ зазвичай є можливість застосування декількох різних способів адресації операндів до однієї і тієї ж операції.

Щоб пристрій управління обчислювальної машини міг визначити, який саме спосіб адресації прийнятий у даній команді, в різних ОМ використовуються різні прийоми. Часто різним способам адресації відповідають і різні коди операції. Інший підхід – це додавання до складу команди спеціального *поля способу адресації*, вміст якого визначає, який із способів адресації повинен бути застосований. Іноді в команді є декілька полів – поодинці на кожну адресу. Відзначимо, що можливий також варіант, коли в команді взагалі відсутня адресна інформація, тобто має місце *неявна адресація*. За неявної адресації адресного поля або взагалі немає, або воно містить не всі необхідні адреси – відсутня адреса мається на увазі кодом операції.

В даний час використовуються різні види адресації, найбільш поширені з яких розглядаються нижче.

Безпосередня адресація. Під час *безпосередньої адресації* (БА) в адресному полі команди замість адреси міститься безпосередньо сам операнд (рис. 2.15). Цей спосіб може застосовуватися під час виконання арифметичних операцій, операцій порівняння, а також для завантаження констант у регістри. Коли операндом є число, воно зазвичай подається в додатковому коді.

Під час запису в регістр, в якому розрядність перевищує довжину безпосереднього операнда, операнд розміщується в молодшій частині регістра, а позиції, що залишилися вільними, заповнюються значенням знакового біта операнда.

Рис. 2.15. Безпосередня адресація

Крім того, що в адресному полі можуть бути вказані тільки константи, ще одним недоліком даного способу адресації є те, що розмір безпосереднього

операнда обмежений довжиною адресного поля команди, яке в більшості випадків менше довжини машинного слова. Безпосередня адресація скорочує час виконання команди, оскільки не потрібне звернення до пам'яті за операндом. Крім того, економиться пам'ять, оскільки відпадає необхідність в комірці для зберігання операнда.

Пряма адресація. У випадку *прямої* або *абсолютної адресації* (ПА) адресний код прямо указує номер комірки пам'яті, до якої проводиться звернення (рис. 2.16).

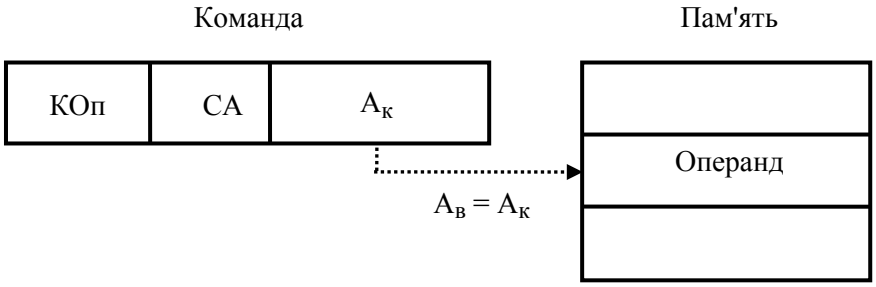


Рис. 2.16. Пряма адресація

При всій простоті використання спосіб має істотний недолік – обмежений розмір адресного простору, оскільки для адресації до пам'яті великої ємності потрібне «довге» адресне поле. Проте істотнішою недосконалістю можна вважати те, що адреса, вказана в команді, не може бути зміненою в процесі обчислень (в усякому разі, така зміна не рекомендується). Це обмежує можливості щодо довільного розміщення програми в пам'яті.

Непряма адресація. Одним із шляхів подолання проблем, властивих прямій адресації, може служити прийом, коли за допомогою обмеженого адресного поля команди вказується адреса комірки, що у свою чергу містить повнорозрядну адресу операнда (рис. 2.17). Цей спосіб відомий як *непряма адресація* (НА). Запис ($A_к$) означає вміст комірки, адреса якої вказана в дужках.

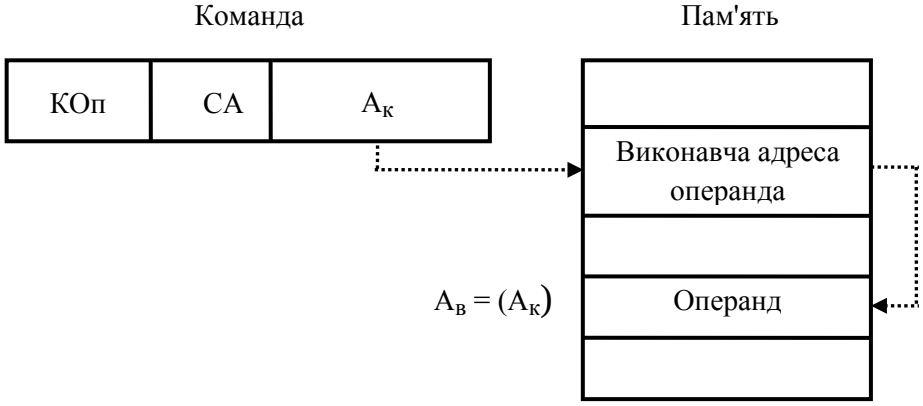


Рис. 2.17. Непряма адресація

У випадку непрямої адресації вміст адресного поля команди залишається незмінним, тоді як непряму адресу в процесі виконання програми можна змінювати. Це дозволяє проводити обчислення, коли адреси операндів

заздалегідь невідомі і з'являються лише в процесі розв'язання задачі. Додатково такий прийом спрощує обробку масивів і списків, а також передачу параметрів підпрограмам.

Недоліком непрямої адресації є необхідність у двократному зверненні до пам'яті: спочатку для витягання адреси операнда, а потім для звернення до операнда. Понад те задіюється зайва комірка пам'яті для зберігання виконавчої адреси операнда.

Регістрова адресація (РА) нагадує пряму адресацію. Відмінність полягає в тому, що адресне поле інструкції указує не на комірку пам'яті, а на регістр процесора (рис. 2.18). Ідентифікатор регістра надалі позначатимемо буквою *R*. Зазвичай розмір адресного поля в даному випадку складає три або чотири біти, що дозволяє вказати відповідно на один з 8 або 16 регістрів загального призначення (РЗП).

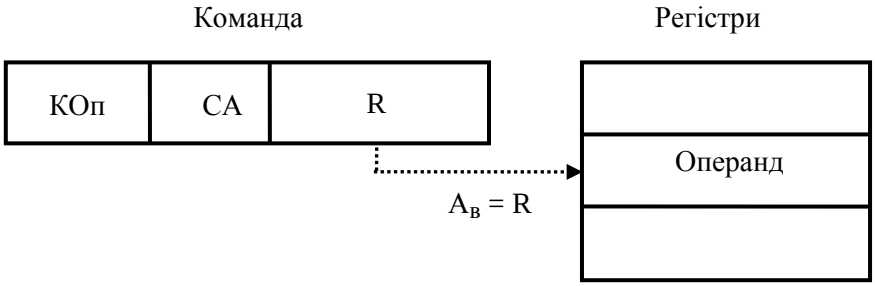


Рис. 2.18. Регістрова адресація

Двома основними перевагами регістрової адресації є: коротке адресне поле в команді і виключення звернень до пам'яті.

На жаль, можливості по використанню регістрової адресації обмежені малим числом РЗП у складі процесора.

Непряма регістрова адресація (НРА) є непрямою адресацією, де виконавча адреса операнда зберігається не в комірці основної пам'яті, а в регістрі процесора. Відповідно, адресне поле команди указує не на комірку пам'яті, а на регістр (рис. 2.19).

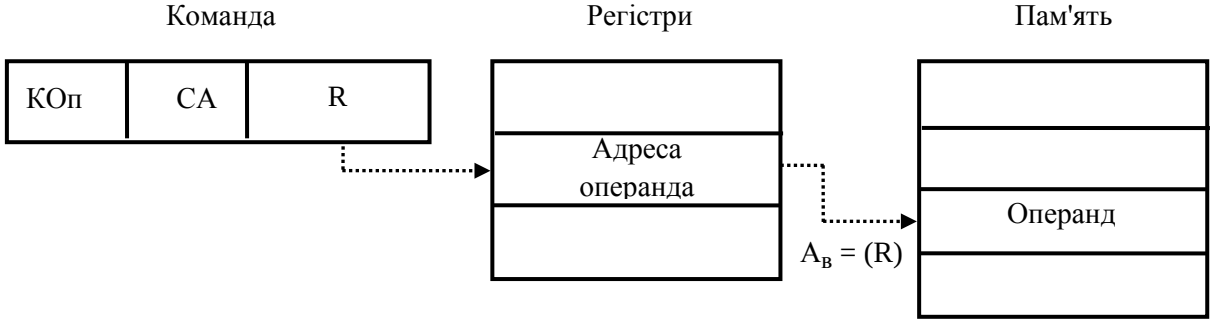


Рис. 2.19. Непряма регістрова адресація

Переваги і обмеження непрямої регістрової адресації ті ж, що і у звичайної непрямої адресації, але завдяки тому, що непряма адреса зберігається не в

пам'яті, а в регістрі, для доступу до операнда потрібно на одне звернення до пам'яті менше.

Адресація зі зміщенням. Під час адресації зі зміщенням виконавча адреса формується в результаті підсумовування вмісту адресного поля команди з вмістом одного або декількох регістрів процесора (рис. 2.20). *Адресація зі зміщенням* припускає, що адресна частина команди включає як мінімум одне поле ($A_к$). В ньому міститься константа, сенс якої в різних варіантах адресації зі зміщенням може мінятися. Константа може бути якоюсь базовою адресою, до якої додається зміщення, що зберігається в регістрі.

Допустимий і прямо протилежний підхід: базова адреса знаходиться в регістрі процесора, а в полі $A_к$ указується зміщення щодо цієї адреси.

В деяких процесорах для реалізації певних варіантів адресації зі зміщенням передбачені спеціальні регістри, наприклад базовий або індексний.

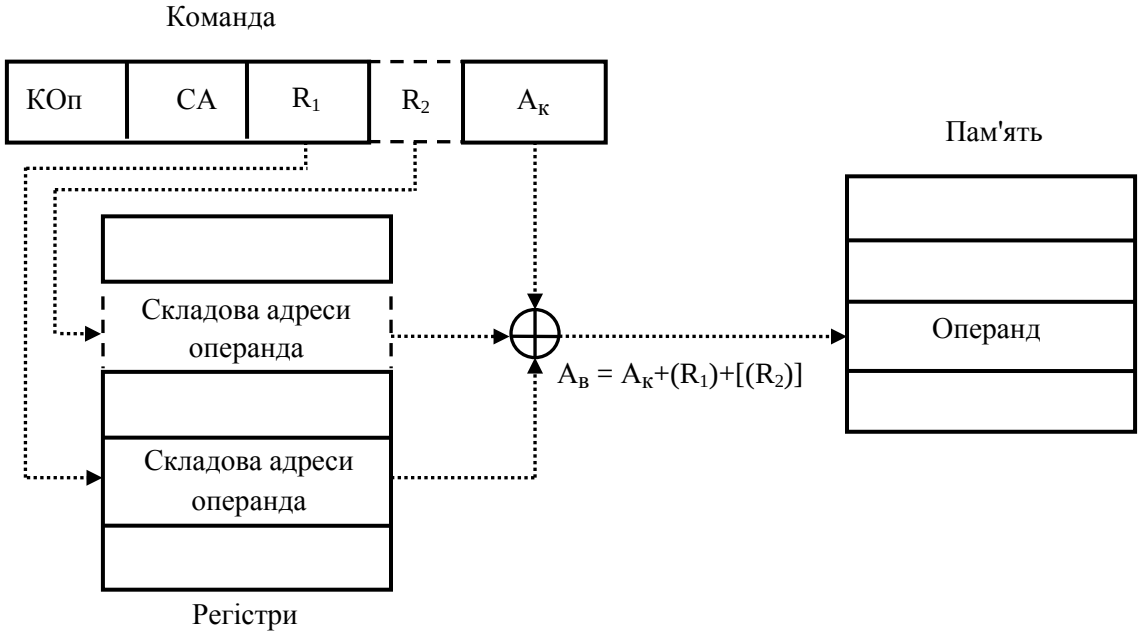


Рис. 2.20. Адресація зі зміщенням

Якщо ж складова адреси може розташовуватися в довільному регістрі загального призначення, то для вказівки конкретного регістра в команду включається додаткове поле *R* (у випадку складання адреси більш ніж з двох складових у команді буде декілька таких полів). У найбільш загальному випадку адресація зі зміщенням має на увазі наявність двох адресних полів: $A_к$ і *R*.

У рамках адресації зі зміщенням є ще один варіант, при якому виконавча адреса обчислюється не підсумовуванням, а конкатенацією (приєднанням) складових адреси.

Нижче розглядаються основні способи адресації зі зміщенням, кожен з яких має власну назву.

Відносна адресація. У випадку *відносної адресації* (ВА) для отримання виконавчої адреси операнда вміст підполя $A_к$ команди складається з вмістом лічильника команд (рис. 2.21). Таким чином, адресний код у команді є зміщенням щодо адреси поточної команди.

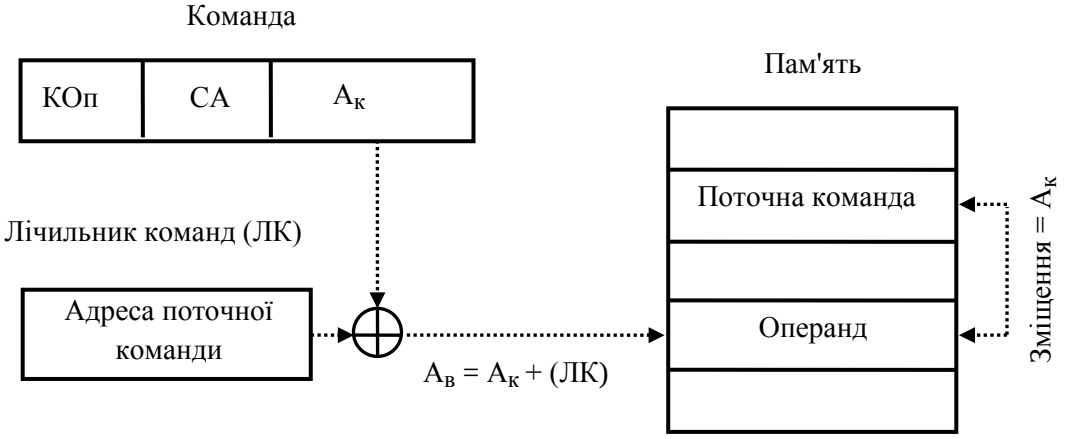


Рис. 2. 21. Відносна адресація

Слід зазначити, що у момент обчислення виконавчої адреси операнда в лічильнику команд може вже бути сформована адреса наступної команди, що потрібно враховувати під час вибору величини зміщення. Зазвичай підполе $A_к$ трактується як двійкове число в додатковому коді.

Адресація відносно лічильника команд базується на властивості локальності, що виражається в тому, що більша частина звернень відбувається до комірок, розташованих у безпосередній близькості від виконуваної команди. Це дозволяє заощадити на довжині адресної частини команди, оскільки розрядність підполя $A_к$ може бути невеликою. Головна перевага даного способу адресації полягає в тому, що він робить програму переміщуваною в пам'яті: незалежно від поточного розташування програми в адресному просторі взаємне положення команди і операнда залишається незмінним, тому адресація операнда залишається коректною.

Базова регістрова адресація. У разі *базової регістрової адресації* (БРА) регістр, званий базовим, містить повнорозрядну адресу, а підполе $A_з$ - зміщення щодо цієї адреси (рис. 2.22).

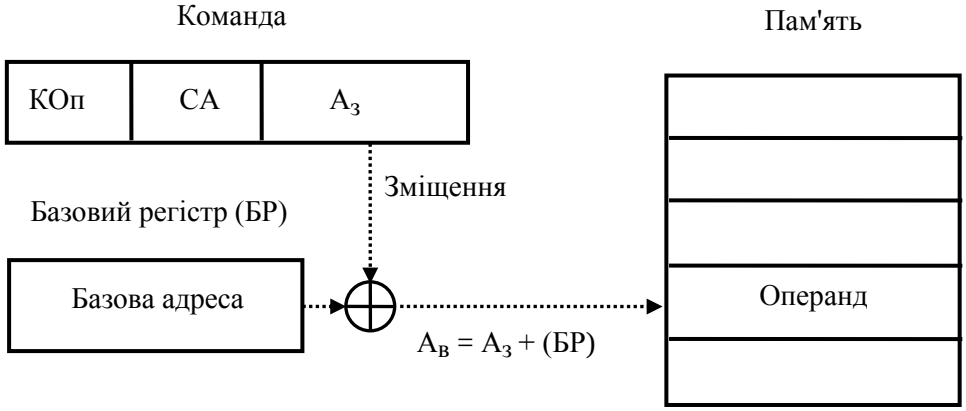


Рис. 2.22. Базова регістрова адресація з базовим регістром

Посилання на базовий регістр може бути явним або неявним. У деяких ОМ є спеціальний базовий регістр і його використання є неявним, тобто підполе *R* в команді відсутнє

Типовіший випадок, коли в ролі базового регістра виступає один з регістрів загального призначення (РЗП), тоді його номер явно вказується в підполі *R* команди (рис. 2.23).

Базову регістрову адресацію зазвичай використовують для доступу до елементів масиву, положення якого в пам'яті в процесі обчислень може мінятися. У базовий регістр заноситься початкова адреса масиву, а адреса елемента масиву вказується в підполі $A_з$ команди у вигляді зміщення щодо початкової адреси масиву.

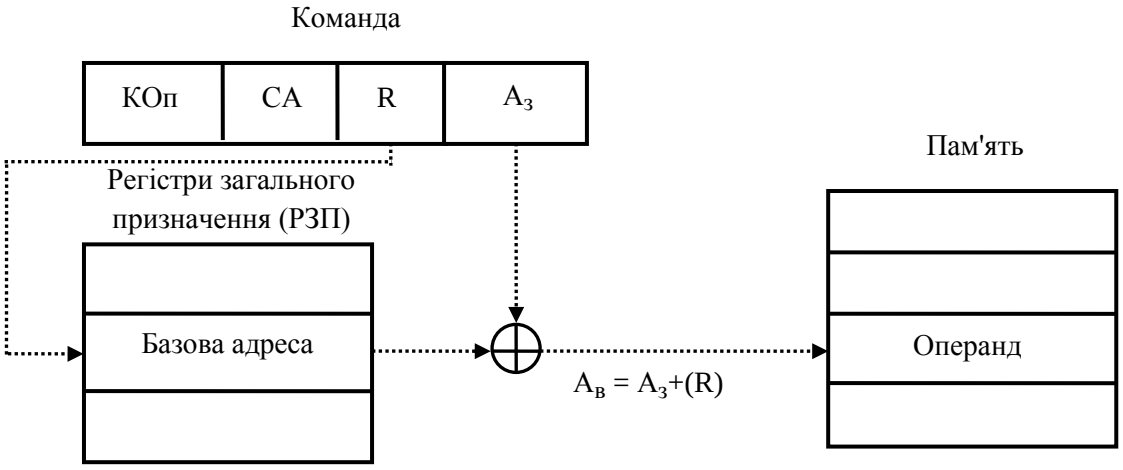


Рис. 2.23. Базова регістрова адресація з використанням одного з РЗП

Перевага даного способу адресації в тому, що зміщення має меншу довжину, чим повна адреса, і це дозволяє скоротити довжину адресного поля команди.

Індексна адресація. У разі *індексної адресації* (ІА) підполе $A_б$ містить адресу комірки пам'яті, а регістр (вказаний явно або неявно) – зміщення щодо цієї адреси (рис. 2.24).

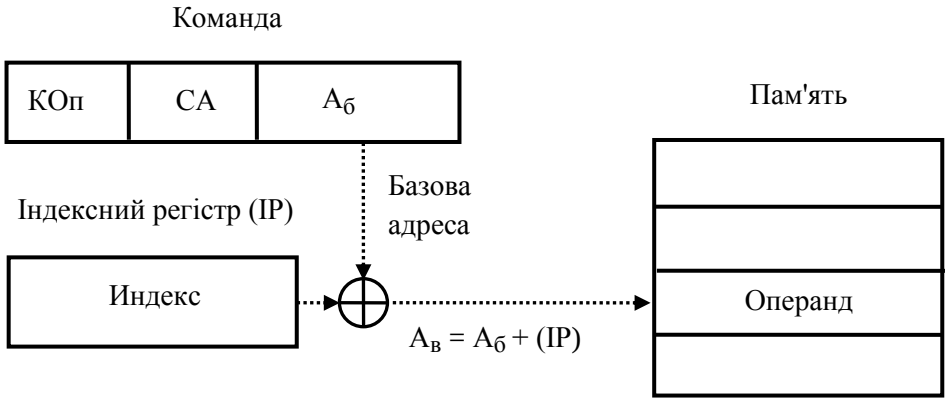


Рис. 2.24. Індексна адресація з індексним регістром

Як видно, цей спосіб адресації схожий на базову регістрову адресацію. Оскільки у разі індексної адресації в полі $A_б$ знаходиться повнорозрядна адреса комірки пам'яті, що грає роль бази, довжина цього поля більша, ніж у разі базової регістрової адресації. Проте обчислення виконавчої адреси операнда проводиться ідентично.

У разі, коли в ролі індексного регістра для зберігання зміщення виступає один з регістрів загального призначення (РЗП), тоді його номер явно вказується в підполі *R* команди (рис. 2.25).

Індексна адресація надає зручний механізм для організації ітеративних обчислень. Хай, наприклад, є масив чисел, розташованих у пам'яті послідовно, починаючи з адреси *N*, і ми хочемо збільшити на одиницю всі елементи даного масиву. Для цього потрібно витягувати кожне число з пам'яті, додати до нього 1. Послідовність виконавчих адрес буде такою: *N*, *N*+1, *N*+2 і т. д., аж до останньої комірки, займаної даним масивом. Значення *N* береться з підполя команди, а у вибраний регістр, званий *індексним регістром*, спочатку заноситься 0. Після кожної операції вміст індексного регістра збільшується на 1.

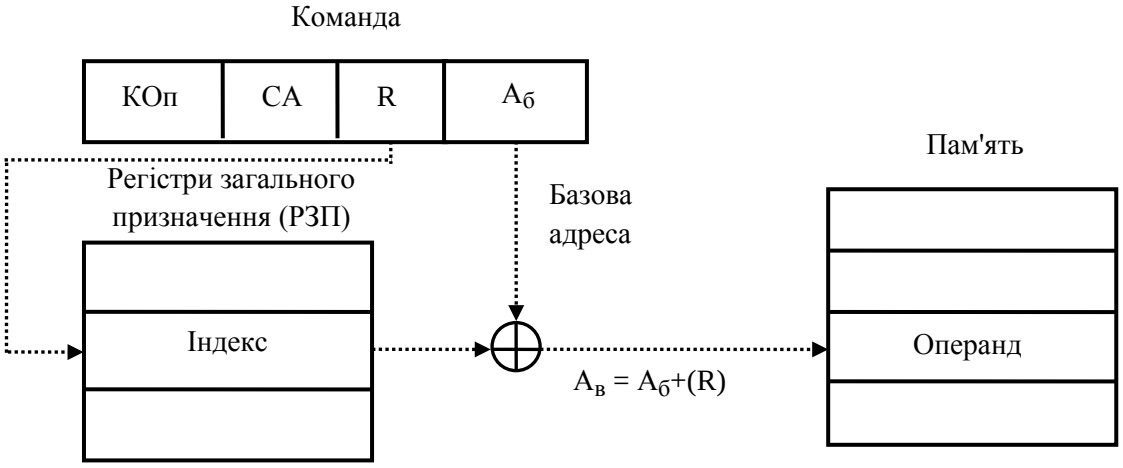


Рис. 2.25. Індексна адресація з використанням одного з РЗП

Оскільки це досить типовий випадок, у більшості ОМ збільшення або зменшення вмісту індексного регістра до або після звернення до нього здійснюється автоматично як частина машинного циклу. Такий прийом називається *автоіндексуванням*. Якщо для індексної адресації використовуються спеціально виділені регістри, автоіндексування може проводитися неявно і автоматично. У випадку задіювання для зберігання індексів регістрів загального призначення необхідність операції автоіндексування повинна указуватися в команді спеціальним бітом.

Автоіндексування із збільшенням вмісту індексного регістра носить назву *автоінкрементної адресації* і може бути описано таким чином:

$$A_в = A_б + (\mathrm{R}),\ R < (\mathrm{R}) + 1 \ \text{ або } \ R < (\mathrm{R}) + 1,\ A_в = A_б + (\mathrm{R}).$$

У першому варіанті збільшення вмісту індексного регістра відбувається після формування виконавчої адреси, і цей спосіб називається *постінкрементним автоіндексуванням*. У другому випадку спочатку проводиться збільшення вмісту індексного регістра, і вже нове значення використовується для формування виконавчої адреси. Тоді говорять про *преінкрементне автоіндексування.*

Автоіндексування із зменшенням вмісту індексного регістра носить назву *автодекрементної адресації* і може бути описано так:

$$A_в = A_б + (\mathrm{R}),\ R < (\mathrm{R}) - 1 \ \text{ або } \ R < (\mathrm{R}) - 1,\ A_в = A_б + (\mathrm{R}).$$

Тут також можливі два варіанти, що відрізняються послідовністю виконання операцій зменшення вмісту індексного регістра і обчислення виконавчої адреси: *постдекрементне автоіндексування* і *переддекрементне автоіндексування*.

Існує ще один варіант індексної адресації – *індексна адресація з масштабуванням і зсувом*: вміст індексного регістра множиться на масштабний коефіцієнт і підсумовується з $A_б$. Масштабний коефіцієнт може приймати значення 1, 2, 4 або 8, для чого в адресній частині команди виділяється додаткове поле.

Сторінкова адресація (СТА) припускає розбиття адресного простору на сторінки. Сторінка визначається своєю початковою адресою, що виступає як база. Старша частина цієї адреси зберігається в спеціальному регістрі – *регістрі адреси сторінки* (РАС). В адресному коді команди вказується зміщення усередині сторінки, що розглядається як молодша частина виконавчої адреси. Виконавча адреса утворюється конкатенацією (приєднанням, символ ||) $A_з$ до вмісту РАС, як показано на рис. 2.26.

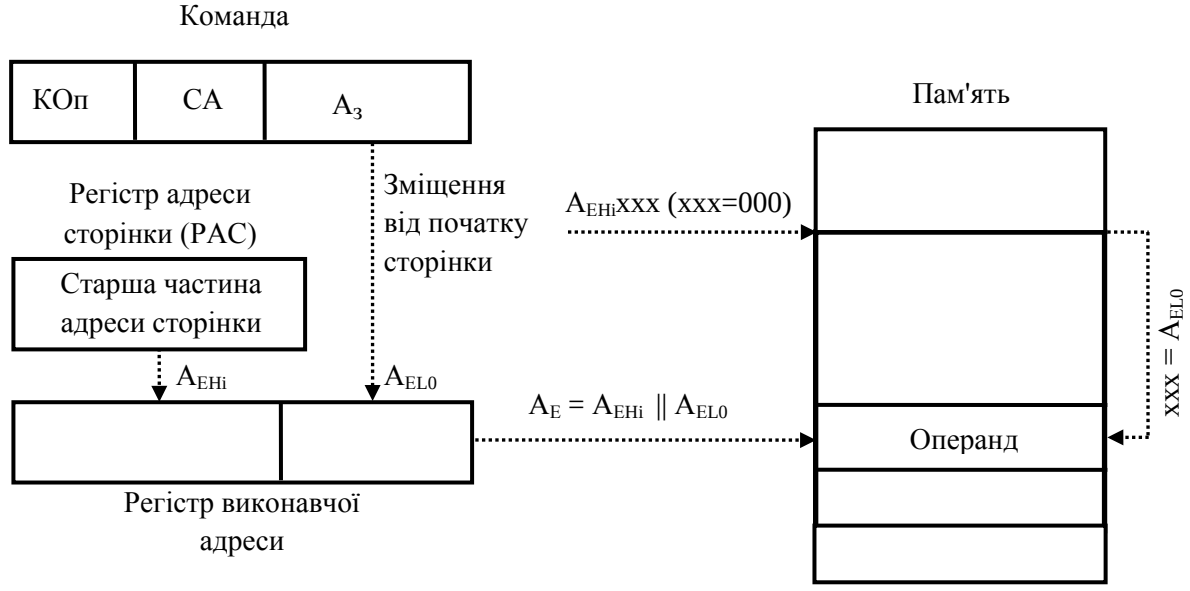


Рис. 2.26. Сторінкова адресація

Блочна адресація використовується в командах, для яких одиницею обробки служить блок даних, розташованих у послідовних комірках пам'яті. Цей спосіб дуже зручний під час роботи із зовнішніми запам'ятовуючими пристроями і в операціях з векторами. Для опису блока зазвичай береться адреса комірки, де зберігається перший або останній елемент блока, і загальна кількість елементів блока, яка задана числом байтів або комірок. Замість довжини блока може використовуватися спеціальна ознака «кінець блока», що поміщається за останнім елементом блока

Стекова адресація буде розглянута під час опису стекової архітектури системи команд.

2.4. АРХІТЕКТУРА СИСТЕМИ КОМАНД

Системою команд обчислювальної машини називають повний перелік команд, які здатна виконувати дана ОМ. У свою чергу, під *архітектурою системи команд* (АСК) прийнято визначати ті засоби обчислювальної машини, які видні і доступні програмістові.

Загальна характеристика архітектури системи команд обчислювальної машини складається з відповідей на такі питання [25]:

1. Якого вигляду дані будуть подані в обчислювальній машині і у якій формі?
2. Де ці дані можуть зберігатися крім основної пам'яті?
3. Яким чином здійснюватиметься доступ до даних?
4. Які операції можуть бути виконані над даними?
5. Скільки операндів може бути присутніми в команді?
6. Як визначатиметься адреса чергової команди?
7. Яким чином будуть закодовані команди?

В історії розвитку обчислювальної техніки відбиваються зміни, що відбувалися в поглядах розробників на перспективність тієї або іншої архітектури системи команд.

Серед мотивів, що найчастіше зумовлюють перехід до нового типу АСК, зупинимося на двох найбільш істотних. Перший – це склад операцій, що виконуються обчислювальною машиною, та їх складність. Другий – місце зберігання операндів, що впливає на кількість і довжину адрес, які вказуються в адресній частині команд обробки даних.

2.4.1. Класифікація АСК за складом і складністю команд

Сучасна технологія програмування орієнтована на мови високого рівня (МВР), головна мета яких – полегшити процес програмування. Перехід до МВР, однак, породив серйозну проблему: складні оператори, характерні для МВР, істотно відрізняються від простих машинних операцій, що

реалізовуються в більшості обчислювальних машин. Проблема отримала назву семантичного розриву, а її наслідком стає недостатньо ефективне виконання програм на ОМ. Намагаючись подолати семантичний розрив, розробники обчислювальних машин у даний час вибирають один з трьох підходів і, відповідно, один з трьох типів АСК:

- архітектуру з повним набором команд: CISC (Complex Instruction Set Computer);
- архітектуру зі скороченим набором команд: RISC (Reduced Instruction Set Computer);
- архітектуру з командними словами надвеликої довжини: VLIW (Very Long Instruction Word).

В обчислювальних машинах типу *CISC* проблема семантичного розриву вирішується за рахунок розширення системи команд, доповнення її складними командами, семантично аналогічними операторам МВР. Основоположником CISC-архітектури вважається компанія IBM, яка почала застосовувати даний підхід з сімейства машин IBM 360 і продовжує його в своїх могутніх сучасних універсальних ОМ, таких як IBM ES/9000. Аналогічний підхід характерний і для компанії Intel в її мікропроцесорах серії 8086 і Pentium. Для CISC-архітектури типові:

- наявність у процесорі порівняно невеликого числа регістрів загального призначення;
- велика кількість машинних команд, деякі з них апаратно реалізують складні оператори МВР;
- різноманітність способів адресації операндів;
- множина форматів команд різної розрядності;
- наявність команд, де обробка поєднується зі зверненням до пам'яті.

До типу CISC можна віднести практично всі ОМ, що випускалися до середини 1980-х років, і значну частину тих, що виробляються в даний час. Розглянутий спосіб вирішення проблеми семантичного розриву разом з тим веде до ускладнення апаратури ОМ, головним чином пристрою управління, що, у свою чергу, негативно позначається на продуктивності ОМ у цілому.

Відмічений недолік і його наслідки привели до серйозного перегляду традиційних рішень, наслідком чого стала поява *RISC-архітектури*. Ідея полягає в обмеженні списку команд ОМ найбільш часто використовуваними простими командами, оперуючими даними, що розміщені тільки в регістрах процесора. Звернення до пам'яті допускається лише за допомогою спеціальних команд читання і запису. Різко зменшена кількість форматів команд і способів указівки адрес операндів. Скорочення числа форматів команд і їх простота, використання обмеженої кількості способів адресації, відділення операцій обробки даних від операцій звернення до пам'яті дозволяє істотно спростити

апаратні засоби ОМ і підвищити їх швидкодію. RISC-архітектура розроблялася так, щоб зменшити час виконання програми за рахунок скорочення середньої кількості тактів процесора, що доводяться на одну команду, і тривалості тактового періоду. Як наслідок, реалізація складних команд за рахунок послідовності з простих, але швидких RISC-команд виявляється не менш ефективною, чим апаратний варіант складних команд в CISC-архітектурі.

Елементи RISC-архітектури вперше з'явилися в обчислювальних машинах CDC 6600 і супер-ЕОМ компанії Cray Research. Достатньо успішно реалізується RISC-архітектура і в сучасних ОМ, наприклад у процесорах Alpha фірми DEC, серії PA фірми Hewlett-Packard, сімействі PowerPC і т. п.

Відзначимо, що в останніх мікропроцесорах фірми Intel і AMD широко використовуються ідеї, властиві RISC-архітектурі, так що багато відмінностей між CISC і RISC поступово стираються.

Крім CISC- і RISC-архітектур у загальній класифікації був згаданий ще один тип АСК – архітектура з командними словами надвеликої довжини (VLIW). Концепція VLIW базується на RISC-архітектурі, де декілька простих RISC-команд об'єднуються в одну наддовгу команду і виконуються паралельно. В плані АСК архітектура VLIW порівняно мало відрізняється від RISC. З'явився лише додатковий рівень паралелізму обчислень, через що архітектуру VLIW логічніше адресувати не до обчислювальних машин, а до обчислювальних систем.

2.4.2. Класифікація АСК за місцем зберігання операндів

Кількість команд і їх складність, безумовно є найважливішими чинниками, проте не меншу роль при виборі АСК грає відповідь на питання про те, де можуть зберігатися операнди і яким чином до них здійснюється доступ.

З цих позицій розрізняють такі види архітектур системи команд:

- стекову;
- акумуляторну;
- регістрову;
- з виділеним доступом до пам'яті.

Вибір тієї або іншої архітектури впливає на принципові моменти: скільки адрес міститиме адресна частина команд, яка буде довжина цих адрес, наскільки просто буде відбуватися доступ до операндів і якою, зрештою, буде загальна довжина команд.

Стекова архітектура

Стеком називається пам'ять, що за своєю структурною організацією відмінна від основної пам'яті ОМ. Принципи побудови стекової пам'яті детально розглянемо пізніше, тут же виділимо тільки ті аспекти, які потрібні для пояснення особливостей АСК на базі стека.

Стек утворює множину логічно взаємозв'язаних комірок, що взаємодіють за принципом «останнім увійшов, першим вийшов» (LIFO - Last In First Out). Верхню комірку називають *вершиною стека*. Для роботи зі стеком передбачено дві операції: *push* (проштовхування даних у стек) і *pop* (виштовхування даних із стека). Запис можливий тільки у верхню комірку стека, при цьому вся інформація, що зберігається в стеку, заздалегідь проштовхується на одну позицію вниз. Читання допустиме також тільки з вершини стека. Інформація, що витягнута, видаляється із стека, а його вміст, що залишився, просувається вверх. У обчислювальних машинах, де реалізована АСК на базі стека (їх зазвичай називають стековими), операнди перед обробкою поміщаються в дві верхні комірки стекової пам'яті. Результат операції заноситься в стек.

Основні вузли та інформаційні тракти одного з можливих варіантів ОМ на основі стекової АСК показані на рис. 2.27.

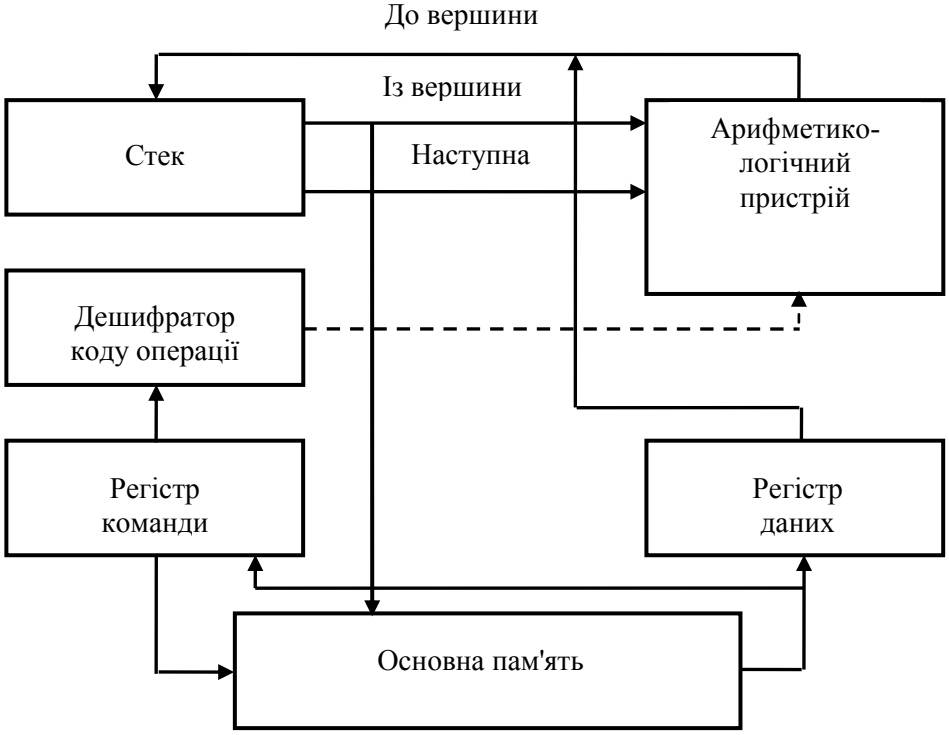


Рис. 2.27. Архітектура обчислювальної машини на базі стека

Інформація може бути занесена у вершину стека з пам'яті або з АЛП. Для запису в стек вмісту комірки пам'яті з адресою *x* виконується команда *push x*, за якою інформація зчитується з комірки пам'яті, заноситься в регістр даних, а потім проштовхується в стек. Результат операції з АЛП заноситься у вершину стека автоматично.

Збереження вмісту вершини стека в комірці пам'яті з адресою *x* проводиться командою *pop x*. За цією командою вміст верхньої комірки стека подається на шину, з якою і проводиться запис у комірку *x*, після чого вся інформація, що знаходиться в стеку, проштовхується на одну позицію вверх. Для виконання арифметичної або логічної операції на вхід АЛП подається

інформація, лічена з двох верхніх комірок стека (при цьому вміст стека просувається на дві позиції вверх, тобто операнди із стека видаляються). Результат операції заштовхується у вершину стека. Можливий варіант, коли результат відразу ж переписується в пам'ять за допомогою автоматично виконуваної операції *pop x*.

Верхні елементи стекової пам'яті, де зберігаються операнди і куди заноситься результат операції, як правило, робляться більш швидкодіючими і розміщуються в процесорі, тоді як решта частини стека може розташовуватися в основній пам'яті і частково навіть на магнітному диску.

До переваг АСК на базі стека слід віднести можливість скорочення адресної частини команд, оскільки всі операції проводяться через вершину стека, тобто адреси операндів і результату в командах арифметичної і логічної обробки інформації вказувати не потрібно. Код програми виходить компактним. Досить просто реалізується декодування команд.

З другого боку, стекова АСК за визначенням не припускає довільного доступу до пам'яті, через що компілятору важко створити ефективний програмний код, хоча створення самих компіляторів спрощується. Крім того, стек стає «вузьким місцем» ОМ в плані підвищення продуктивності. Через згадані причини, даний вигляд АСК довгий час вважався неперспективним і зустрічався, головним чином, в обчислювальних машинах 1960-х років, наприклад в ОМ фірми Burroughs (B5500, B6500) або фірми Hewlett-packard (HP2116B, HP3000/70).

Останні події в області обчислювальної техніки свідчать про відродження інтересу до стекової архітектури ОМ. Зв'язано це з популярністю мови Java і розширенням сфери застосування мови Forth, семантиці яких найбільш близька саме стекова архітектура. Серед сучасних ОМ із стековою АСК можна згадати машини JEM1 і JEM2 компанії Systems і Clip фірми Imsys. Особливо слід зазначити стекову машину IGNITE компанії Patriot Scientist, яку її автори вважають представником нового виду АСК – архітектурою *з безоперандним набором команд.* Для позначення таких ОМ вони пропонують абревіатуру ROSC (Removed Operand Set Computer).

Акумуляторна архітектура

Архітектура на базі акумулятора історично виникла одною з перших. У ній для зберігання одного з операндів арифметичної або логічної операції в процесорі є виділений регістр – *акумулятор*. У цей же регістр заноситься і результат операції. Оскільки адреса одного з операндів зумовлена, в командах обробки досить явно вказати місцеположення тільки другого операнда. Спочатку обидва операнди зберігаються в основній пам'яті, і до виконання операції один з них потрібно завантажити в акумулятор. Після виконання команди обробки результат знаходиться в акумуляторі і, якщо він не є

операндом для подальшої команди, його потрібно зберегти в комірці пам'яті. Для завантаження в акумулятор вмісту комірки *x* передбачена команда завантаження *load x*. За цією командою інформація зчитується з комірки пам'яті *x*, вихід пам'яті підключається до входів акумулятора і відбувається занесення зчитаних даних в акумулятор.

Запис вмісту акумулятора в комірку *x* здійснюється командою збереження *store x*, під час виконання якої виходи акумулятора підключаються до шини, після чого інформація з шини записується в пам'ять.

Типова архітектура ОМ на базі акумулятора показана на рис. 2.28.

Для виконання операції в АЛП проводиться зчитування одного з операндів з пам'яті в регістр даних. Другий операнд знаходиться в акумуляторі. Виходи регістра даних і акумулятора підключаються до відповідних входів АЛП. Після закінчення поточної операції результат з виходу АЛП заноситься в акумулятор.

Перевагами акумуляторної АСК можна вважати короткі команди і простоту декодування команд. Проте наявність всього одного регістра породжує багатократні звернення до основної пам'яті.

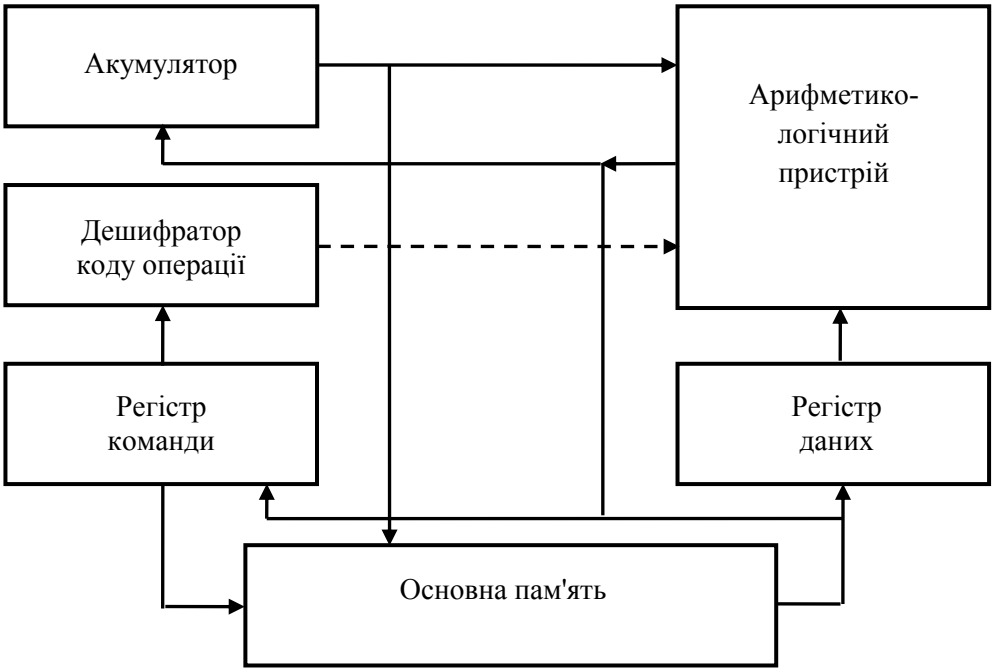


Рис. 2.28. Архітектура обчислювальної машини на базі акумулятора

АСК на базі акумулятора була популярна в ранніх ОМ, таких, наприклад, як IBM 7090, DEC PDP-8, MOS 6502.

Регістрова архітектура

У машинах даного типу процесор включає масив регістрів (регістровий файл), відомих як регістри загального призначення (РЗП). Ці регістри, в якомусь сенсі, можна розглядати як явно керований кеш для зберігання недавно використаних даних.

Розмір регістрів зазвичай фіксований і збігається з розміром машинного слова. До будь-якого регістра можна звернутися, вказавши його номер. Кількість РЗП в архітектурі типу CISC зазвичай невелика (від 8 до 32), і для представлення номера конкретного регістра необхідно не більше п'яти розрядів, завдяки чому в адресній частині команд обробки допустимо одночасно вказати номери двох, а часто і трьох регістрів (двох регістрів операндів і регістра результату). RISC-архітектура припускає використання істотно більшого числа РЗП (до декількох сотень), проте типова для таких ОМ довжина команди (звичайно 32 розряди) дозволяє визначити в команді до трьох регістрів.

Регістрова архітектура допускає розташування операндів в одному з двох запам'ятовуючих середовищ: основній пам'яті або регістрах. З урахуванням можливого розміщення операндів у рамках регістрових АСК виділяють три підвиди команд обробки: *регістр-регістр; регістр-пам'ять; пам'ять-пам'ять.*

У варіанті «*регістр-регістр*» операнди можуть знаходитися тільки в регістрах. В них же засилається і результат.

Підтип «*регістр-пам'ять*» припускає, що один з операндів розміщується в регістрі, а другий в основній пам'яті. Результат зазвичай заміщає один з операндів.

У командах типу «*пам'ять-пам'ять*» обидва операнди зберігаються в основній пам'яті. Результат заноситься в пам'ять.

Варіант «*регістр-регістр*» є основним в обчислювальних машинах типу RISC. Команди типу «*регістр-пам'ять*» характерні для CISC-машин. Нарешті, варіант «*пам'ять-пам'ять*» вважається неефективним, хоча і залишається в найбільш складних моделях машин класу CISC.

Можливу структуру та інформаційні тракти обчислювальної машини з регістровою архітектурою системи команд ілюструє рис. 2.29.

Операції завантаження регістрів з пам'яті і збереження вмісту регістрів у пам'яті ідентичні таким же операціям з акумулятором.

Відмінність полягає в етапі вибору потрібного регістра, що забезпечується відповідними селекторами.

Виконання операції в АЛП включає:

- вибір регістра першого операнда;
- визначення розташування другого операнда (пам'ять або регістр);
- подачу на вхід АЛП операндів і виконання операції;
- вибір регістра результату і занесення в нього результату операції з АЛП.

Звернемо увагу на те, що між АЛП і регістровим файлом повинні бути принаймні три шини.

До переваг регістрових АСК слід віднести: компактність отримуваного коду, високу швидкість обчислень за рахунок заміни звернень до основної пам'яті на звернення до швидких регістрів. З другого боку, дана архітектура вимагає довших інструкцій у порівнянні з акумуляторною архітектурою.

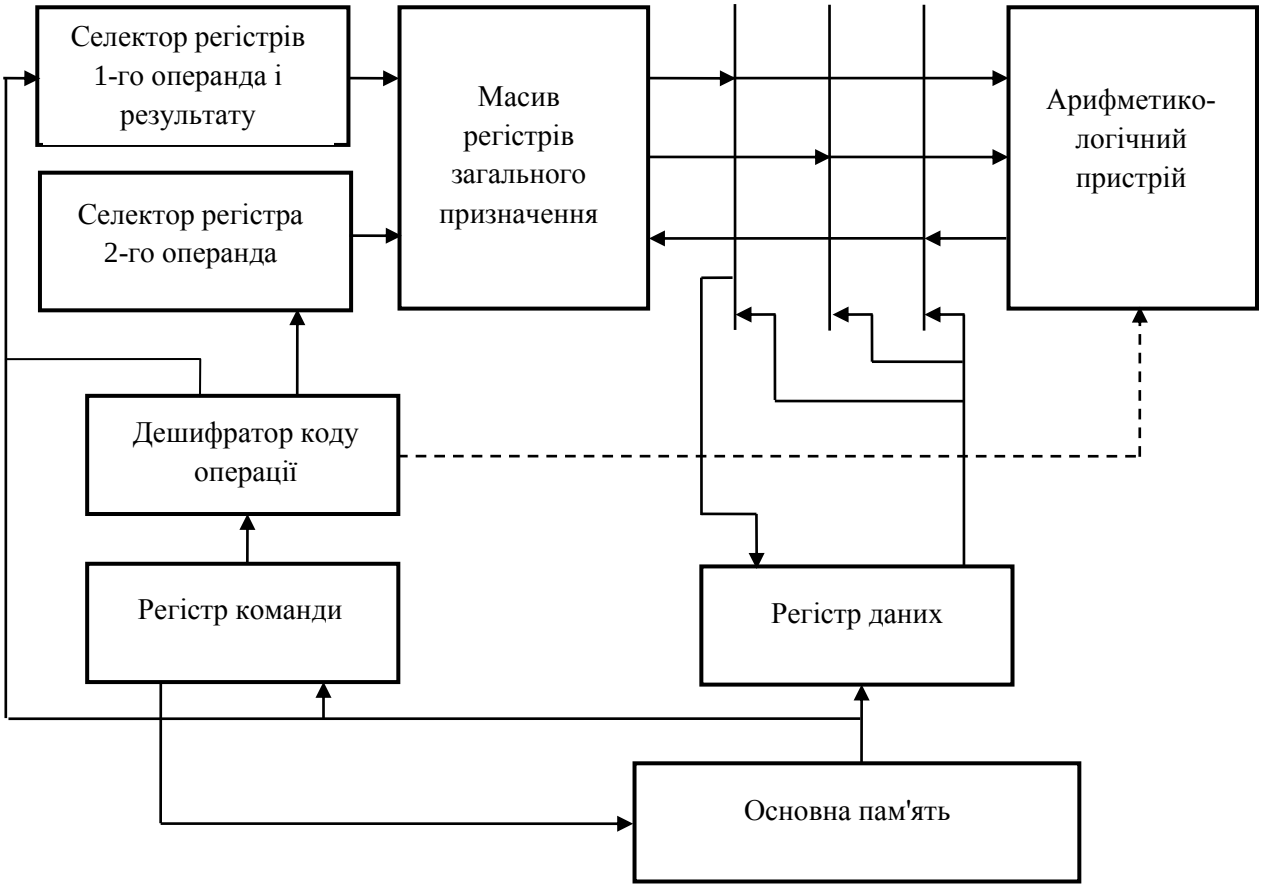


Рис. 2.29. Архітектура обчислювальної машини на базі регістрів загального призначення

Прикладами машин на базі РЗП можуть служити CDC 6600, PDP-11, IBM 360/370, всі сучасні персональні комп'ютери. Можна стверджувати, що в наші дні цей вид архітектури системи команд є переважаючим.

Архітектура з виділеним доступом до пам'яті

В архітектурі з виділеним доступом до пам'яті звернення до основної пам'яті можливо тільки за допомогою двох спеціальних команд: *load* і *store*. В англійській транскрипції дану архітектуру називають Load/Store architecture. Команда *load* (завантаження) забезпечує зчитування значення з основної пам'яті і занесення його в регістр процесора (у команді зазвичай вказується адреса комірки пам'яті і номер регістра). Пересилка інформації в протилежному напрямі проводиться командою *store* (збереження). Операнди у всіх командах обробки інформації можуть знаходитися тільки в регістрах процесора (найчастіше в регістрах загального призначення). Результат операції також заноситься в регістр. В архітектурі відсутні команди обробки, що допускають пряме звернення до основної пам'яті.

Склад та інформаційні тракти ОМ з виділеним доступом до пам'яті показані на рис. 2.30.

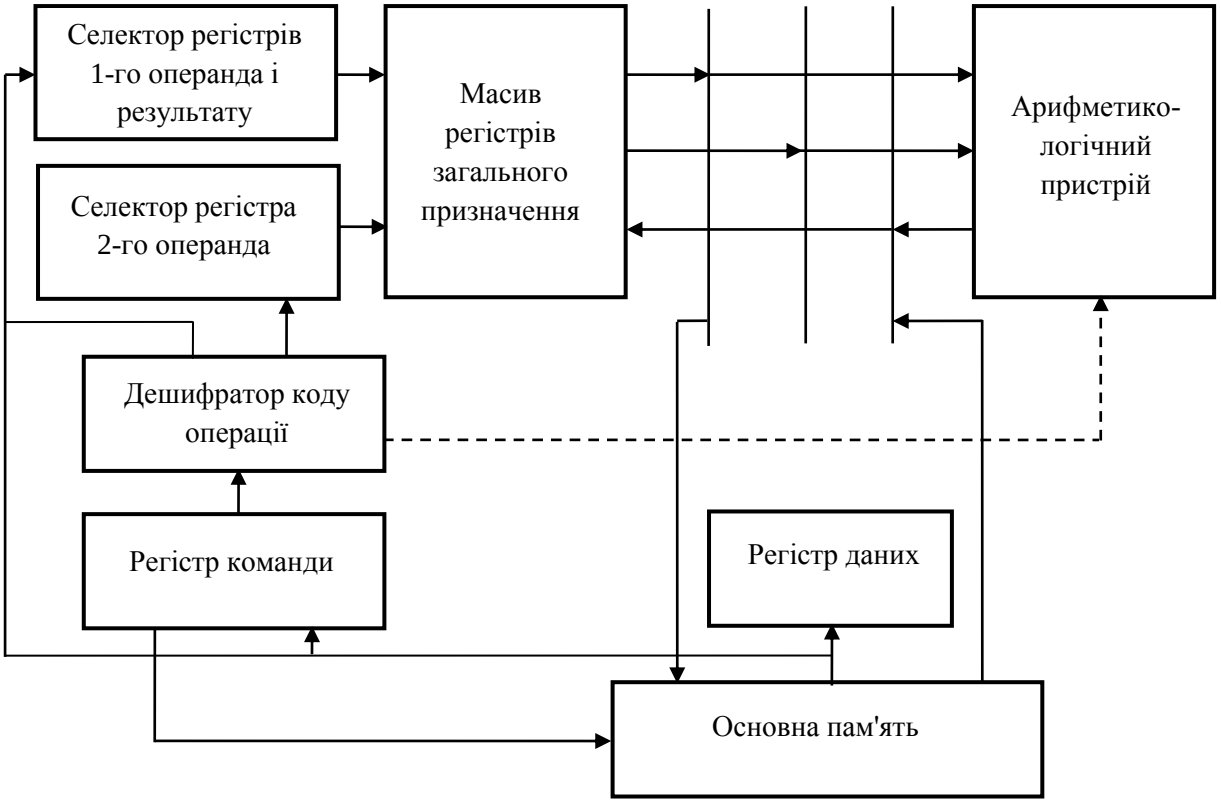


Рис. 2.30. Архітектура обчислювальної машини з виділеним доступом до пам'яті

Дві з трьох шин, розташованих між масивом РЗП і АЛП, забезпечують передачу в арифметико-логічний пристрій операндів, що зберігаються в двох регістрах загального призначення. Третя служить для занесення результату у виділений для цього регістр. Ці ж шини дозволяють завантажити в регістри вміст елементів основної пам'яті і зберегти в ОП інформацію, що знаходиться в РЗП.

АСК з виділеним доступом до пам'яті характерна для всіх обчислювальних машин з RISC-архітектурою. Команди в таких ОМ, як правило, мають довжину 32 біта і триадресний формат. Як приклади обчислювальних машин з виділеним доступом до пам'яті можна відзначити HP PA-RISC, IBM RS/6000, Sun SPARC, MIPS R4000, DEC Alpha і т. д. До переваг АСК слід віднести простоту декодування і виконання команди.

КОНТРОЛЬНІ ПИТАННЯ

1. Які типи команд використовуються в сучасних ОМ?
2. У чому полягає особливість і в якому форматі повинні бути подані операнди?
3. Які види команд умовного переходу зазвичай домінують в реальних програмах?
4. Які чинники визначають вибір формату команд?

5. Перерахуйте можливі шляхи скорочення довжини коду команди.

6. Яка особливість фон-неймановської архітектури дозволяє відмовитися від вказівки в команді адреси чергової команди?

7. Які чинники необхідно враховувати під час вибору оптимальної адресності команд?

8. Яку позицію коми у форматі з фіксованою комою можна вважати загальноприйнятою?

9. Чим у форматі з фіксованою комою заповнюються надмірні старші розряди?

10. Яку мінімальну кількість полів повинен містити формат з плаваючою комою?

11. Як у форматі з плаваючою комою вирішується проблема роботи з порядками, що мають різні знаки?

12. Від чого залежать точність і діапазон подання чисел у форматі з плаваючою комою?

13. Які чинники впливають на вибір розрядності цілих чисел?

14. З якими обмеженнями пов'язано використання безпосередньої адресації?

15. В яких випадках може бути зручна багаторівнева непряма адресація?

16. Які переваги дає адресація щодо лічильника команд (відносна адресація)?

17. У чому виявляються схожість і відмінності між базовою і індексною адресацією?

18. У чому полягає суть автоіндексування і в яких ситуаціях воно застосовується?

19. У чому полягає проблема семантичного розриву?

20. Поясніть відмінності в підходах що до подолання семантичного розриву, вживаних у ОМ з CISC- і RISC-архітектурами

3. ОРГАНІЗАЦІЯ ПАМ'ЯТІ КОМП'ЮТЕРА

У будь-якій ОМ, незалежно від її архітектури, програми і дані зберігаються в пам'яті. Функції пам'яті забезпечуються запам'ятовуючими пристроями (ЗП), призначеними для фіксації, зберігання і видачі інформації в процесі роботи ОМ. Процес фіксації інформації в ЗП називається записом, процес видачі інформації – читанням або зчитуванням, а спільно їх визначають як процеси звернення до ЗП.

3.1. ХАРАКТЕРИСТИКИ СИСТЕМ ПАМ'ЯТІ

Перелік основних характеристик, які необхідно враховувати, розглядаючи конкретний вид ЗП, включає:

- *місце розташування;*
- *ємність;*
- *одиницю пересилки;*
- *метод доступу;*
- *швидкодію;*
- *фізичний тип;*
- *фізичні особливості;*
- *вартість.*

За **місцем розташування** ЗП розділяють на процесорні, внутрішні і зовнішні. Найбільш швидкісні види пам'яті (регістри, кеш-пам'ять першого рівня) зазвичай розміщують на загальному кристалі з центральним процесором, а регістри загального призначення взагалі вважаються частиною ЦП. Другу групу (внутрішню пам'ять) утворюють ЗП, розташовані на системній платі. До внутрішньої пам'яті відносять основну пам'ять, а також кеш-пам'ять другого і подальших рівнів (кеш-пам'ять другого рівня може також розміщуватися на кристалі процесора). Повільні ЗП великої ємності (магнітні і оптичні диски, магнітні стрічки) називають зовнішньою пам'яттю, оскільки до ядра ОМ вони підключаються аналогічно пристроям вводу/виводу.

Ємність ЗП характеризують числом бітів або байтів, яке може зберігатися в запам'ятовуючому пристрої. На практиці застосовуються крупніші одиниці, а для їх позначення до слів «біт» або «байт» додають префікси: кіло, мега, гіга, тера, пета, экза (kilo, mega, giga, tera, peta, exa). Стандартно ці префікси означають множення основної одиниці вимірювань на 10^3, 10^6, 10^9, 10^{12}, 10^{15} і 10^{18} відповідно. У обчислювальній техніці, орієнтованій на двійкову систему числення, вони відповідають значенням, достатньо близьким до стандартних, але таких, що є цілим ступенем числа 2, тобто 2^{10}, 2^{20}, 2^{30}, 2^{40}, 2^{50}, 2^{60}.

Важливою характеристикою ЗП є **одиниця пересилки**. Для основної пам'яті (ОП) одиниця пересилки визначається шириною шини даних, тобто кількістю бітів, які передаються по лініях шини паралельно. Зазвичай одиниця пересилки рівна довжині слова, але не обов'язково. Стосовно зовнішньої пам'яті дані часто передаються одиницями, що перевищують розмір слова, і такі одиниці називаються *блоками*.

Під час оцінки швидкодії необхідно враховувати вживаний в даному типі ЗП **метод доступу** до даних. Розрізняють чотири основні методи доступу:

- *Послідовний доступ*. ЗП з послідовним доступом орієнтований на зберігання інформації у вигляді послідовності блоків даних, що називаються записами. Для доступу до потрібного елемента (слова або байта) необхідно прочитати всі передуючі йому дані. Час доступу залежить від положення необхідного запису в послідовності записів на носієві інформації і позиції елемента всередині даного запису. Прикладом може служити ЗП на магнітній стрічці.
- *Прямий доступ*. Кожен запис має унікальну адресу, що відображає її фізичне розміщення на носієві інформації. Звернення здійснюється як адресний доступ на початок запису, з подальшим послідовним доступом до певної одиниці інформації всередині запису. У результаті час доступу до певної позиції є величиною змінною. Такий режим характерний для магнітних дисків.
- *Довільний доступ*. Кожен елемент пам'яті має унікальну фізичну адресу. Звернення до будь-якої комірки займає один і той же час і може проводитись у довільній черговості. Прикладом можуть служити запам'ятовуючі пристрої основної пам'яті.
- *Асоціативний доступ*. Цей вид доступу дозволяє виконувати пошук комірок, що містять таку інформацію, в якій значення окремих бітів збігається зі станом однойменних бітів у заданому зразку. Порівняння здійснюється паралельно для всіх елементів пам'яті, незалежно від її ємності. За асоціативним принципом побудовані деякі блоки кеш-пам'яті.

Швидкодія ЗП є одним з найважливіших його показників. Для кількісної оцінки швидкодії зазвичай використовують три параметри:

- *Час доступу* ($T_Д$). Для пам'яті з довільним доступом він відповідає інтервалу часу від моменту надходження адреси до моменту, коли дані заносяться в пам'ять або стають доступними. У ЗП з рухомим носієм інформації – це час, що витрачається на встановлення головки запису/ зчитування (або носія) в потрібну позицію.
- *Тривалість циклу пам'яті* або *період звернення* ($T_ц$). Поняття застосовується до пам'яті з довільним доступом, для якої воно означає мінімальний час між двома послідовними зверненнями до пам'яті. Період звернення включає час доступу плюс деякий додатковий час. Додатковий час

може бути потрібним для затухання сигналів на лініях, а в деяких типах ЗП, де зчитування інформації приводить до її руйнування, – для відновлення зчитаної інформації.

- *Швидкість передачі.* Це швидкість, з якою дані можуть передаватися в пам'ять або з неї. Для пам'яті з довільним доступом вона дорівнює $1/T_{ц}$. Для інших видів пам'яті швидкість передачі визначається співвідношенням

$$T_N = T_A + \frac{N}{R} \ ,$$

де T_N – середній час зчитування або запису N бітів;

T_A – середній час доступу;

R – швидкість пересилки в бітах за секунду.

Кажучи про **фізичний тип** запам'ятовуючого пристрою, необхідно згадати три найбільш поширених технології ЗП – це напівпровідникова пам'ять, пам'ять з магнітним носієм інформації, використовувана в магнітних дисках і стрічках, і пам'ять з оптичним носієм – оптичні диски.

Залежно від застосованої технології слід враховувати і ряд **фізичних особливостей** ЗП, наприклад енергозалежність. В енергозалежній пам'яті інформація може бути спотворена або втрачена під час відключення джерела живлення. В енергонезалежних ЗП записана інформація зберігається і у разі відключення напруги живлення. Магнітна і оптична пам'ять – енергонезалежні. Напівпровідникова пам'ять може бути як енергозалежною, так і ні, залежно від її типу. Крім енергозалежності потрібно враховувати, приводить зчитана інформація до її руйнування чи ні.

Вартість ЗП прийнято оцінювати відношенням загальної вартості ЗП до його ємності в бітах, тобто вартістю зберігання одного біта інформації.

3.2. ІЄРАРХІЯ ЗАПАМ'ЯТОВУЮЧИХ ПРИСТРОЇВ

Під час створення системи пам'яті постійно доводиться вирішувати задачу забезпечення необхідної ємності і високої швидкодії за прийнятну ціну. Найбільш поширеним підходом тут є побудова системи пам'яті ОМ за ієрархічним принципом. Ієрархічна пам'ять складається із ЗП різних типів (рис. 3.1), які, залежно від характеристик, відносять до певного рівня ієрархії. Вищий рівень менше по ємності, швидше і має велику вартість в перерахунку на біт, ніж нижчий рівень.

Рівні ієрархії взаємозв'язані: всі дані на одному рівні можуть бути також знайдені на нижчому рівні, і всі дані на цьому нижчому рівні можуть бути знайдені на наступному рівні, що знаходиться нижче, і так далі.

Чотири верхні рівні ієрархії утворюють *внутрішню пам'ять* ОМ, а всі нижні рівні – це *зовнішня* або *вторинна* пам'ять. Рухаючись вниз по ієрархічній структурі:

1) зменшується співвідношення «вартість/біт»;
2) зростає ємність;
3) росте час доступу;
4) зменшується частота звернення до пам'яті з боку центрального процесора.

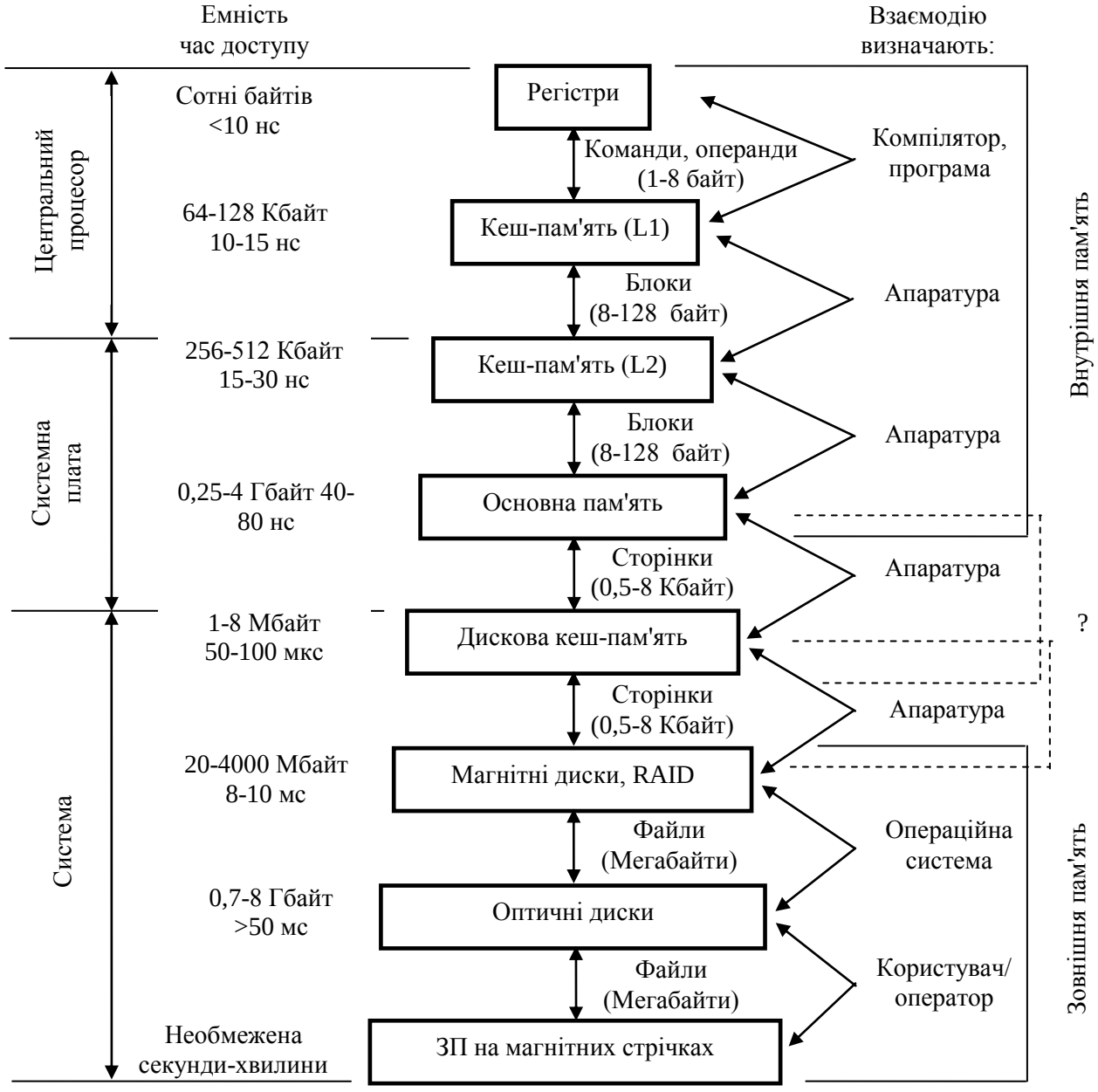


Рис. 3.1. Ієрархія запам'ятовуючих пристроїв

На кожному рівні ієрархії інформація розбивається на блоки, які є найменшою інформаційною одиницею, що пересилається між двома сусідніми рівнями ієрархії. Розмір блоків може бути фіксованим або змінним. У разі фіксованого розміру блока ємність пам'яті зазвичай кратна його розміру. Розмір

блоків на кожному рівні ієрархії найчастіше різний і збільшується від верхніх рівнів до нижніх.

Під час доступу до команд і даних, наприклад для їх зчитування, спочатку проводиться пошук у пам'яті верхнього рівня. Факт виявлення потрібної інформації називають *попаданням* (hit), інакше говорять про *промах* (miss). У разі промаху проводиться пошук в ЗП наступного нижчого рівня, де також можливі попадання або промах. Після виявлення необхідної інформації виконується послідовна пересилка блока, що містить шукану інформацію, з нижніх рівнів на верхні. Слід зазначити, що незалежно від числа рівнів ієрархії пересилка інформації може здійснюватися тільки між двома сусідніми рівнями.

Найшвидший, але і мінімальний по ємності тип пам'яті – це внутрішні регістри ЦП, які іноді об'єднують поняттям *надоперативний запам'ятовуючий пристрій*, – НОЗП. Як правило, кількість регістрів невелика, хоча в архітектурі зі скороченим набором команд їх число може доходити до декількох сотень. Основна пам'ять (ОП), значно більшої ємності, розташовується декількома рівнями нижче. Між регістрами ЦП і основною пам'яттю часто розміщують кеш-пам'ять, яка за ємністю відчутно програє ОП, але істотно перевершує останню за швидкодією, поступаючись в той же час НОЗП. У більшості сучасних ОМ є декілька рівнів кеш-пам'яті, які позначають буквою L і номером рівня кеш-пам'яті. На рис. 3.1 показані два таких рівні. В останніх розробках все частіше з'являється також третій рівень кеш-пам'яті (L3), причому розробники ОМ говорять про доцільність введення і четвертого рівня – L4. Кожен подальший рівень кеш-пам'яті має більшу ємність, але одночасно і меншу швидкодію в порівнянні з попередньою. Всі види внутрішньої пам'яті реалізуються на основі напівпровідникових технологій і в основному є енергозалежними. Довготривале зберігання великих об'ємів інформації (програм і даних) забезпечується зовнішніми ЗП, серед яких найбільш поширені запам'ятовуючі пристрої на базі магнітних і оптичних дисків, а також ЗП на магнітних стрічках.

Нарешті, ще один рівень ієрархії може бути доданий між основною пам'яттю і дисками. Цей рівень носить назву дискової кеш-пам'яті і реалізується у вигляді самостійного ЗП, такого, що включається до складу магнітного диска. Дискова кеш-пам'ять істотно покращує продуктивність при обміні інформацією між дисками і основною пам'яттю.

Ієрархія може бути доповнена і іншими видами пам'яті. Так, деякі моделі ОМ фірми IBM включають так звану розширену пам'ять (expanded storage), виконану на основі напівпровідникової технології, але маючу меншу швидкодію і вартість у порівнянні з ОП. Строго кажучи, цей вид пам'яті не входить в ієрархію, а є відгалуженням від неї, оскільки дані можуть передаватися тільки між розширеною і основною пам'яттю, але не допускається обмін між розширеною і зовнішньою пам'яттю.

3.3. ПРИНЦИПИ ПОБУДОВИ ОСНОВНИХ ТИПІВ ПАМ'ЯТІ

Способи організації пам'яті залежать від методів розташування та пошуку інформації в запам'ятовуючому масиві. За цією ознакою розрізняють адресну та безадресну пам'яті.

3.3.1. Адресні запам'ятовуючі пристрої

В пам'яті з адресною організацією розташування та пошук інформації в ЗП основані на використанні адреси зберігання слова (числа, команди і т.п.). Адресою служить номер комірки ЗП, в якій це слово розташовується.

Під час запису (або зчитування) слова в ЗП команда, яка ініціює цю операцію, повинна вказувати адресу (номер комірки), за якою проводиться запис (зчитування).

Типова структура адресної пам'яті показана на рис. 3.2.

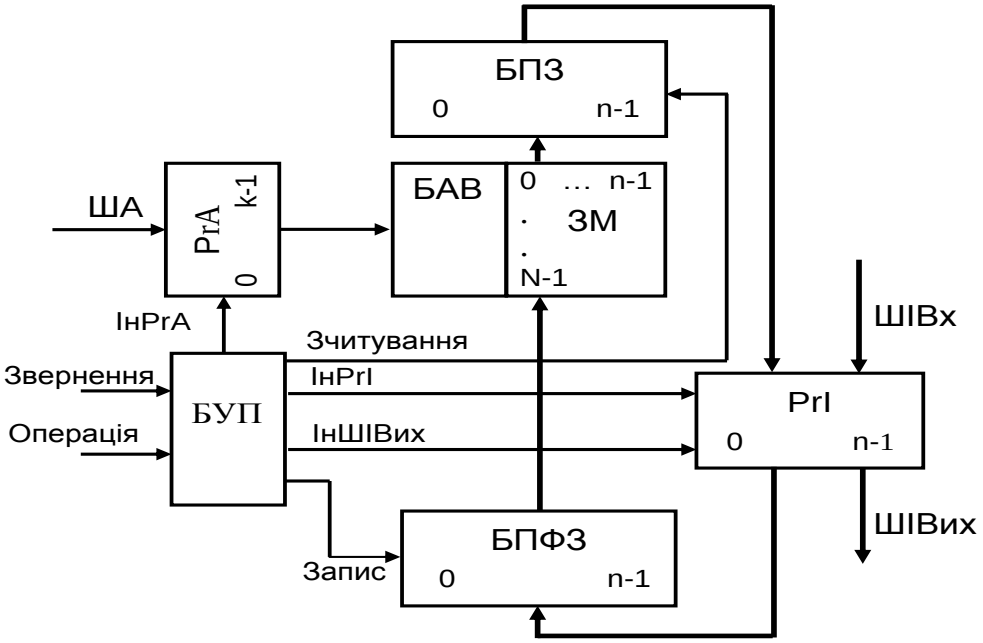


Рис.3.2. Структура адресної пам'яті

Структура містить запам'ятовуючий масив (ЗМ) з N n-розрядних комірок та його апаратурне обладнання, яке включає в себе регістр адреси (РгА), який має k розрядів $(k \geq log_2 N)$, інформаційний регістр (РгІ), блок адресної вибірки (БАВ), блок підсилювачів зчитування (БПЗ), блок розрядних підсилювачів-формувачів сигналів запису (БПФЗ) та блок управління пам'яттю (БУП).

По коду адреси в РгА БАВ формує у відповідній комірці пам'яті сигнали, які дозволяють провести зчитування або запис слова. Цикл звернення до пам'яті ініціюється надходженням в БУП ззовні сигналу *звернення*.

Загальна частина циклу звертання включає в себе прийом в РгА з шини ША адреси звертання та прийом в БУП і розшифровку управляючого сигналу *операція*, який вказує вид запитуваної операції (зчитування або запис).

Далі під час зчитування БАВ дешифрує адресу, виконує вибірку комірки пам'яті, посилає сигнал зчитування у вибрану комірку ЗП. Код записаного в комірку слова зчитується, підсилюється підсилювачами зчитування та передається в РгІ. Операція зчитування завершується видачею слова з РгІ на вихідну інформаційну шину ШІВих.

Під час запису, окрім виконання вказаної вище загальної частини циклу звертання, проводиться прийом слова, яке записується з вхідної інформаційної шини ШІВх в РгІ. Потім у комірку, яку вибрав БАВ, записується слово з РгІ. Зазвичай роль вхідного і вихідного виконує один і той же регістр.

Підсилювачі зчитування/запису служать для електричного узгодження сигналів на лініях даних і внутрішніх сигналів ІМС. Зазвичай число підсилювачів дорівнює числу запам'ятовуючих елементів у рядку матриці, і всі вони під час звернення до пам'яті підключаються до вибраної горизонтальної лінії. Кожна група підсилювачів, яка створює комірку, підключена до відповідних стовпців матриці, тобто вибір потрібної комірки в рядку забезпечується активізацією вертикальних ліній.

Блок управління БУП генерує необхідні послідовності управляючих сигналів, які ініціюють роботу окремих вузлів пам'яті.

3.3.2. Безадресні запам'ятовуючі пристрої

У безадресній пам'яті для пошуку інформації в запам'ятовуючому масиві використовують не адреси комірок, а інші принципи. До цього типу пам'яті відноситься асоціативна та стекова пам'яті.

Асоціативна пам'ять. У пам'яті цього типу пошук потрібної інформації проводиться не за адресою, а за її змістом (за асоціативною ознакою). При цьому пошук за асоціативною ознакою (або послідовно за окремими розрядами цієї ознаки) відбувається паралельно у часі для всіх комірок запам'ятовуючого масиву. В багатьох випадках асоціативний пошук дозволяє істотно спростувати та прискорити обробку даних. Це досягається за рахунок того, що в пам'яті цього типу операція зчитування інформації суміщена з виконанням ряду логічних операцій.

Типова структура асоціативної пам'яті показана на рис. 3.3.

Запам'ятовуючий масив має $N \ (n+1)$–розрядних комірок. Для вказування зайнятості комірки використовують службовий n-й розряд (0 – комірка вільна, 1 – в комірці записане слово).

По вхідній інформаційній шині ШІВх в регістр асоціативної ознаки (РгАО) в розряди 0 ÷ n-1 надходить n-розрядний асоціативний запит, а в регістр маски (РгМ) – код маски пошуку, при цьому n-й розряд РгМ установлюється в 0.

Асоціативний пошук проводиться лише для сукупності розрядів РгАО, яким відповідає 1 в РгМ (не замасковані розряди РгАО). Для слів, у яких цифри

в розрядах збіглися з незамаскованими розрядами РгАО, комбінаційна схема (КС) установлює 1 у відповідні розряди регістра збігання (РгЗб) і 0 в решту розрядів.

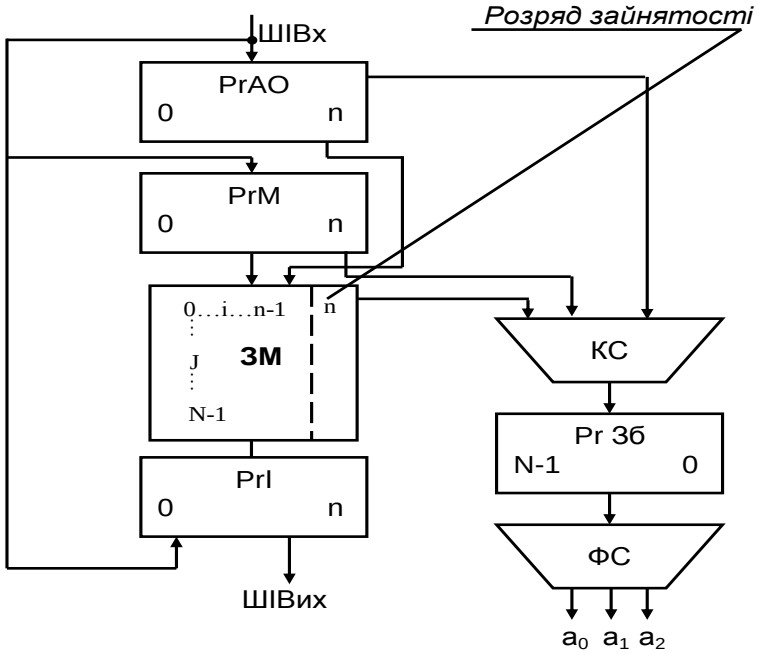


Рис. 3.3. Структура асоціативної пам'яті

Комбінаційна схема формування результатів асоціативного звертання (ФС) формує із слова, яке утворилося в РгЗб, сигнали a_0, a_1, a_2, що відповідають випадкам відсутності слів у ЗМ, які задовольняють асоціативні ознаки та наявність одного (a_1) і більше (a_2) таких слів.

Формування вмісту РгЗб та сигналів a_0, a_1, a_2, за вмістом РгАО, РгМ та ЗМ, називається операцією контролю асоціації. Ця операція є складовою частиною операцій зчитування та запису, хоч вона має самостійне значення.

Під час зчитування спочатку проводиться контроль асоціації за асоціативною ознакою в РгАО. Потім при a_0=1 зчитування скасовується через відсутність потрібної інформації, при a_1=1 зчитується слово, на яке вказує одиниця в РгЗб, при a_2=1 в РгІ зчитується слово із комірки, яка має найменший номер серед комірок, позначених 1 в РгЗб. Із РгІ зчитане слово видається на вихідну інформаційну шину (ШІВих).

Під час запису інформації спочатку відшукується вільна комірка. Для цього виконується операція контролю асоціації при РгАО = 111...10 та РгМ = 000...01, при цьому вільні комірки позначаються 1 в РгЗб. Для запису вибирається вільна комірка з найменшим номером або та, яка довше не використовувалася. В неї записують слово, яке надійшло із ШІВх в РгІ.

За допомогою операції контролю асоціації можна, не зчитуючи слів з пам'яті, визначити за вмістом РгЗб, скільки в пам'яті слів, які задовольняють асоціативні ознаки, наприклад реалізувати запити, типу скільки студентів мають відмінні оцінки з даної дисципліни. Під час використання відповідних

комбінаційних схем в асоціативній пам'яті можуть використовуватись досить складні логічні операції, такі, як пошук більшого (меншого) числа та інше.

Відзначимо, що для асоціативної пам'яті необхідні запам'ятовуючі елементи, які дозволяють виконувати зчитування без знищення записаної в них інформації.

Через відносно високу вартість асоціативна пам'ять рідко використовується як самостійний вид пам'яті.

Стекова пам'ять. Стекову пам'ять можна розглядати як сукупність комірок, що утворюють одновимірний масив, у якому сусідні комірки зв'язані одна з одною розрядними ланцюгами передачі слів. Запис нового слова проводиться в верхню комірку (вершину стека – комірку 0), при цьому всі записані раніше слова (включаючи і слова, записані в комірці 0) зсуваються вниз, в сусідні комірки з більшими на 1 номерами.

Зчитування можливе тільки з верхньої (нульової) комірки пам'яті, при цьому, якщо проводиться зчитування з видаленням, всі інші слова в пам'яті зсуваються вверх, в сусідні комірки з більшими номерами. В цій пам'яті порядок зчитування слів відповідає правилу: останнім надійшов – першим обслужився.

У ряді пристроїв розглянутого типу передбачається також операція простого зчитування слова з нульової комірки (без його видалення і зсуву слова в пам'яті). Іноді стекова пам'ять має лічильник стека (ЛчСт), який показує кількість занесених у пам'ять слів. Сигнал ЛчСт = 0 відповідає порожньому стеку.

Структурна схема стекової пам'яті (СтП) приведена на рис. 3.4.

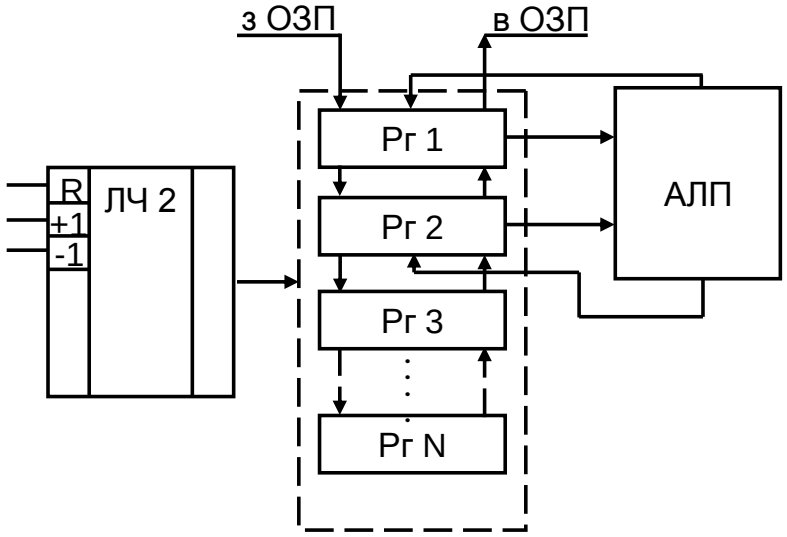


Рис. 3.4. Структурна схема стекової пам'яті

Пошук і переміщення інформації в СтП здійснюється з допомогою реверсивного лічильника (Лч). Він у будь-який момент часу вказує кількість зайнятих комірок СтП. Під час запису інформації показання Лч збільшуються

на одиницю, а вміст СтП переміщується на одну комірку вниз, а при зчитуванні – показання Лч зменшується на одиницю і вміст СтП переміщується на одну комірку вверх.

Оперативний запам'ятовуючий пристрій (ОЗП) має зв'язок тільки з верхньою коміркою пам'яті (Рг1), а процесор – з двома верхніми (Рг1 і Рг2). Операція, яка задається програмою, виконується над вмістом двох верхніх комірок – Рг1 і Рг2, а результат операції завжди залишається в Рг2. Якщо Рг1 вільний, то перед виконанням наступної операції вміст Рг2 передається в Рг1, із Рг3 в Рг2 і т.д. (вміст СтП переміщується на одну комірку вверх), а показання Лч зменшуються на одиницю.

Стекова пам'ять (її ще називають *магазинна* пам'ять) широко застосовується як в обчислювальних системах високої продуктивності, так і в міні- і мікро-ЕОМ, і пропонує використання безадресних команд обробки масивів даних, які утворюють стек.

3.4. ОРГАНІЗАЦІЯ ОСНОВНОЇ ПАМ'ЯТІ КОМП'ЮТЕРА

Основна пам'ять (ОП) є єдиним видом пам'яті, до якої ЦП може звертатися безпосередньо (виняток становлять лише регістри центрального процесора). Інформація, що зберігається на зовнішніх ЗП, стає доступною процесору тільки після того, як буде переписана в основну пам'ять.

Основну пам'ять утворюють запам'ятовуючі пристрої з довільним доступом. Такі ЗП утворені як масив комірок, а «довільний доступ» означає, що звернення до будь-якої комірки займає один і той же час і може проводитися в довільній послідовності. Кожна комірка містить фіксоване число запам'ятовуючих елементів і має унікальну адресу, що дозволяє розрізняти комірки під час звернення до них для виконання операцій запису і зчитування.

Наслідком величезних успіхів у області напівпровідникових технологій стала зміна елементної бази основної пам'яті. На зміну ЗП на базі феромагнітних кілець прийшли напівпровідникові мікросхеми, використання яких у наші дні стало широко розповсюдженим.

Основна пам'ять може включати два типи пристроїв: *оперативні запам'ятовуючі пристрої* (ОЗП) і *постійні запам'ятовуючі пристрої* (ПЗП).

Переважну частку основної пам'яті утворює ОЗП, який називають оперативним, тому що він допускає як запис, так і зчитування інформації, причому обидві операції виконуються однотипно, практично з однією і тією ж швидкістю, і проводяться за допомогою електричних сигналів. У англомовній літературі ОЗП відповідає абревіатура RAM – *Random Access Memory*, тобто «пам'ять з довільним доступом», що не зовсім коректно, оскільки пам'яттю з довільним доступом є також ПЗП і регістри процесора. Для більшості типів напівпровідникових ОЗП характерна енергозалежність – навіть під час

короткочасного переривання живлення інформація, що зберігається, втрачається. Мікросхема ОЗП повинна бути постійно підключена до джерела живлення і тому може використовуватися тільки як тимчасова пам'ять.

Другу групу напівпровідникових ЗП основної пам'яті утворюють незалежні мікросхеми ПЗП (ROM – *Read-Only Memory*). ПЗП забезпечує зчитування інформації, але не допускає її зміни (у ряді випадків інформація в ПЗП може бути змінена, але цей процес сильно відрізняється від зчитування і вимагає значно більшого часу).

3.4.1. Блочна організація основної пам'яті

Ємність основної пам'яті сучасних комп'ютерів достатньо велика, щоб її можна було реалізувати на одній інтегральній мікросхемі (ІМС). Необхідність об'єднання декількох ІМС виникає також, коли розрядність комірок у мікросхемі запам'ятовуючого пристрою менше розрядності слів комп'ютера.

Збільшення розрядності ЗП реалізується за рахунок об'єднання адресних входів ІМС ЗП, які об'єднуються. Інформаційні входи і виходи мікросхем є входами і виходами модуля ЗП збільшеної розрядності (рис. 3.5). Таку об'єднану сукупність мікросхем називають *модулем пам'яті.* Модулем можна називати і одну мікросхему, якщо вона вже має необхідну розрядність. Один або декілька модулів утворюють *банк пам'яті.*

Для отримання потрібної ємності ЗП потрібно певним чином об'єднати декілька банків пам'яті меншої ємності. Взагалі основна пам'ять комп'ютера практично завжди має блочну структуру, тобто містить декілька банків.

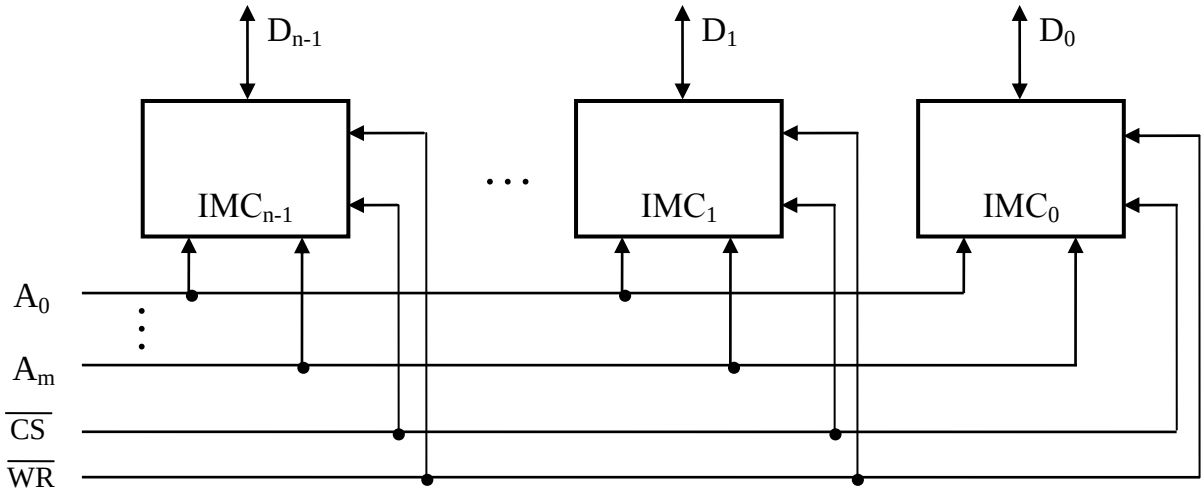


Рис. 3.5. Збільшення розрядності пам'яті

Під час використання блочної пам'яті, яка складається з B банків, адреса комірки A перетворюється в пару (b, w), де b - номер банку, w - адреса комірки всередині банку. Відомі три схеми розподілу розрядів адреси A між b і w : *блочна; циклічна; блочно-циклічна.*

Типова структура *блочної пам'яті* показана на рис. 3.6.

Адресний простір пам'яті розбитий на групи послідовних адрес, і кожна така група забезпечується окремим банком пам'яті. Для звернення до пам'яті використовується 9-розрядна адреса, сім молодших розрядів якої (**A_6 – A_0**) надходять паралельно на всі банки пам'яті і вибирають у кожному з них одну комірку. Два старших розряди адреси (**A_8, A_7**) містять номер банку. Вибір банку забезпечується або за допомогою дешифратора номера банку пам'яті або шляхом мультиплексування інформації (на рис. 3.6 показані обидва варіанти).

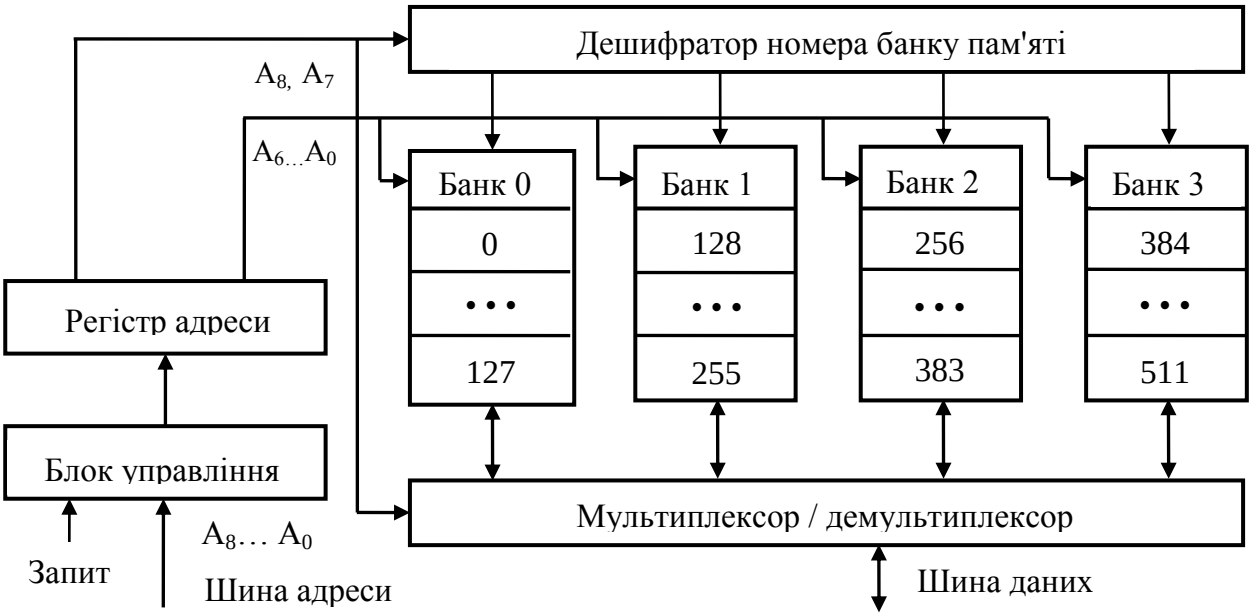


Рис. 3.6. Структура основної пам'яті блочного типу

У функціональному відношенні така пам'ять може розглядатись як єдиний ЗП, ємність якого дорівнює сумі ємностей складових, а швидкодія – швидкодії окремого банку.

Блочний принцип побудови оперативної пам'яті має ще одну перевагу – дозволяє зменшити час доступу до інформації. Це стає можливим завдяки потенціальному паралелізму, який є властивим для блочної організації.

Більшої швидкості доступу можна досягти за рахунок одночасного доступу до багатьох банків пам'яті. Одна з використовуваних для цього методик називається *розшаруванням пам'яті*. В її основу покладене так зване *чергування адрес* (address interleaving – *інтерлів*), яке полягає в зміні системи розподілу адрес між банками пам'яті. Причому чергування адрес базується на властивості локальності по зверненню, відповідно до якого послідовний доступ у пам'ять звичайно виконується до комірок, які мають суміжні адреси. Іншими словами, якщо в даний момент виконується звернення до комірки з адресою 5, то наступне звернення найвірогідніше буде до комірки з адресою 6, а потім 7 і т.д.

Чергування адрес забезпечується за рахунок *циклічного* розбиття адреси (для вибору банку використовуються два розряди адреси, а інші розряди – для вибору комірки в банку). У пам'яті з інтерлівом кожен наступний рядок

вибирається в новому банку, що дозволяє запам'ятовуючим елементам рядка з попереднього банку відновлювати свій стан (рис. 3.7).

Оскільки в кожному такті на шині адреси може бути присутнім адреса тільки однієї комірки, паралельне звернення до декількох банків неможливе, однак воно може бути організоване зі зсувом на один такт.

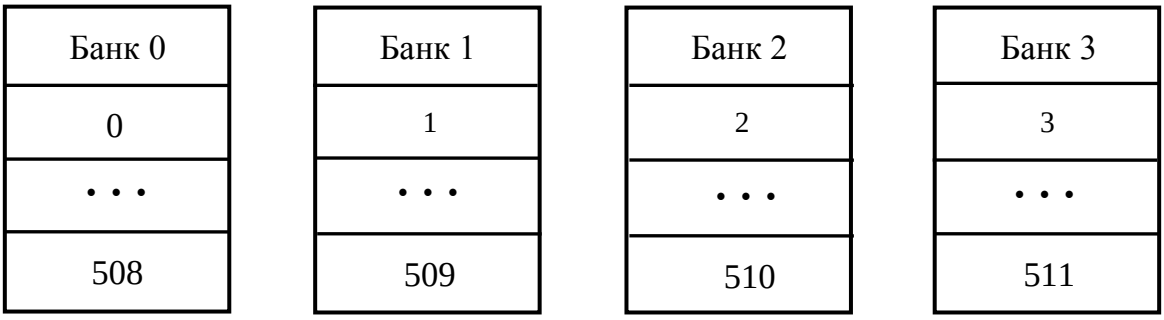


Рис. 3.7. Нумерація комірок у банках пам'яті з циклічною адресацією

У разі великої кількості банків середній час доступу до основної пам'яті скорочується в N разів (N - кількість банків), але за умови, що комірки, до яких проводиться звернення, відносяться до різних банків. Якщо ж запити до одного й того ж банку приходять один за одним, то кожний наступний запит повинен очікувати завершення обслуговування попереднього. Така ситуація називається *конфліктом по доступу.* У разі частого виникнення конфліктів по доступу метод стає неефективним [25].

У *блочно-циклічній* схемі розшарування пам'яті кожний банк складається з декількох модулів, що адресуються за круговою схемою (рис. 3.8).

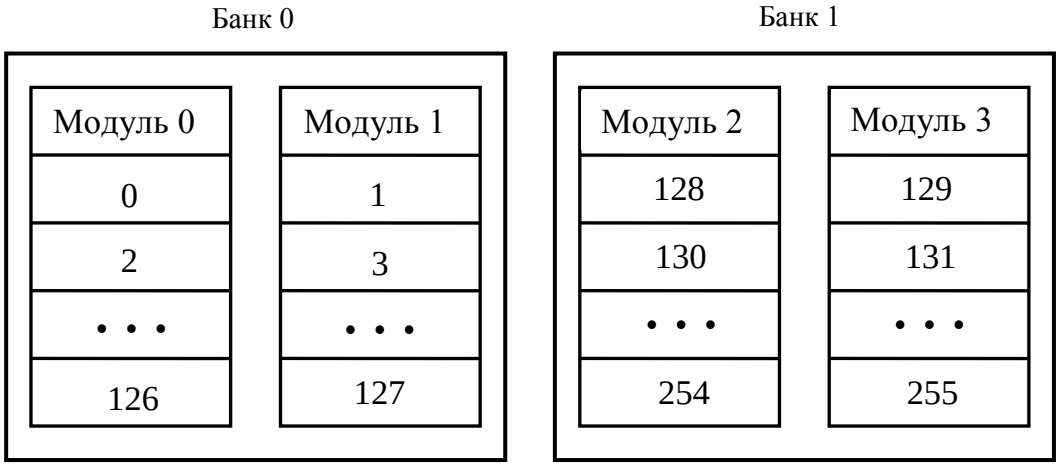


Рис. 3.8. Блочно-циклічна схема розшарування пам'яті

Адреси між банками розподілені по блочній схемі. Таким чином, адреса комірки розбивається на три частини. Старші біти визначають номер банку, наступна група розрядів адреси вказує на комірку в модулі, а молодші біти адреси вибирають модуль у банку.

У багатопроцесорних системах з загальною пам'яттю, де запити на доступ до пам'яті достатньо незалежні, у систему включають декілька контролерів пам'яті, що дозволяє окремим банкам функціонувати абсолютно автономно.

3.4.2. Організація мікросхем пам'яті

Як оперативну пам'ять комп'ютера використовують динамічну пам'ять, яка здатна зберігати інформацію тільки протягом достатньо короткого проміжку часу, після якого інформацію необхідно відновлювати, інакше вона буде загублена. Абревіатура динамічної пам’яті – DRAM (Dynamic Random Access Memory).

Динамічна пам'ять здатна запам'ятовувати біти інформації завдяки паразитній ємності (близько 10^{-15} Ф). Ця пам'ять сконструйована на базі структури компліментарної технології метал-оксид-напівпровідник (CMOS – Complimentary Metal Oxide Semiconductor). За CMOS-технологією, завдяки її безсумнівним технічним перевагам, будуються сучасні чіпи швидкодіючих електронних елементів з високою щільністю упакування.

Інтегральні мікросхеми (ІМС) пам'яті організовані у вигляді матриці комірок, кожна з яких, залежно від розрядності ІМС, складається з одного або більше запам'ятовуючих елементів (ЗЕ) і має власну адресу. Кожний ЗЕ здатний зберігати один біт інформації.

Популярна сьогодні ІМС пам'яті має об'єм 64 Мбіт – тобто 67108864 запам'ятовуючих елементів (ЗЕ). ЗЕ розміщується на перехрестях сітки з провідників. У цьому випадку кожний ЗЕ є під'єднаним до двох провідників – один з яких використовують для керування читанням (записом), інший для передачі (подачі) даних. Структура запам'ятовуючого елемента зображена на рис. 3.9.

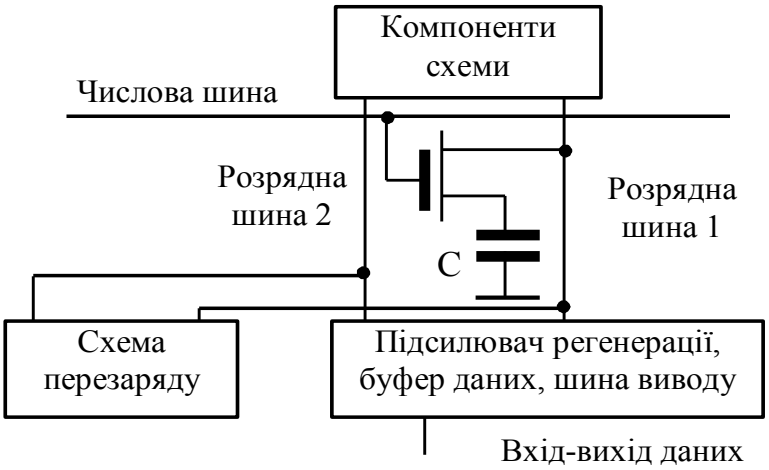


Рис. 3.9. Структура запам’ятовуючого елемента

Транзистор у динамічному ЗЕ працює як ключ, який керує передачею заряду. Під час запису в конденсатор біта інформації ключ відкривається, заряджаючи конденсатор до певної величини.

Для доступу до мікросхеми пам'яті з контролера ОЗП надходять сигнали керування, які переводять числову шину в активний стан. При цьому на числовій шині підвищується потенціал, транзистор відкривається і замикає

ланцюг: ***корпус => розрядна шина 1***. Якщо ємність заряджена, вона розряджається на розрядну шину, підвищуючи її потенціал.

Між розрядними шинами 1 і 2 виникає напруга. Струм, що при цьому циркулює, створює на вихідній шині заряд (одиниця). Якщо ємність не була зарядженою, то на виході формується струм протилежного напрямку й із шини даних знімається нуль. Процес запису зворотний читанню.

У DRAM кожен ЗЕ можна відшукати за його адресними координатами, що оформлені у рядки і стовпці (рис. 3.10).

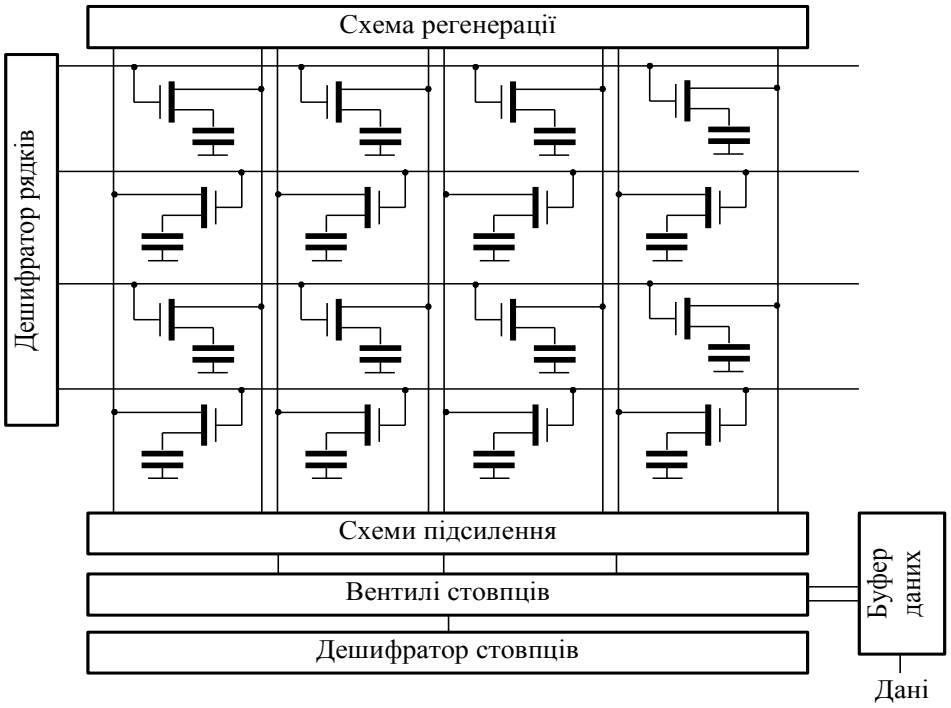


Рис. 3.10. Структура динамічного ОЗП

Запам'ятовуючі елементи, які об'єднані керуючим провідником, прийнято називати рядком, або *row* (розташовані в квадратній таблиці горизонтально). Запам'ятовуючі елементи, які об'єднані провідником, що передає значення, називають стовпцем, або *column* (розташовані по вертикалі). Вибір відповідної адреси рядка і стовпця дозволяє визначити місце ЗЕ.

Таким чином, під час вибору рядка читання здійснюється відразу на всіх ЗЕ, тобто на кожному із провідників стовпців, виникає напруга, обумовлена логічним значенням відповідного ЗЕ обраного рядка. Описану сукупність ЗЕ і логічні елементи, що їх обрамляють і які зв'язані з вибором рядків і стовпців, називають ядром ІМС.

Усі ЗЕ виводяться на загальну числову шину. Вміст декількох ЗЕ, об'єднаних на виході, утворює інформаційну групу – байт або слово і виводиться на шину даних пам'яті. Розрядність мікросхеми визначає кількість ЗЕ, які мають одну і ту ж адресу (така сукупність запам'ятовуючих елементів називається коміркою). Розрядність зовнішньої шини даних пам'яті дозволяє підвищити її пропускну здатність.

3.4.3. Принцип дії динамічної пам'яті

Адреса пам'яті містить відомості для вибору: *байта, банку, рядка і стовпця.* Вона надходять в один із портів контролера ОЗП, трансформуються в дві адреси – рядка і стовпця, які по шині МА потрапляють у DRAM (рис. 3.11) з деяким проміжком часу (ΔТ1 на рис. 3.12).

Контролер пам'яті оснащений портом для обміну даними з процесором і ще одним портом – для обміну з пристроями вводу/виводу на системній шині [22].

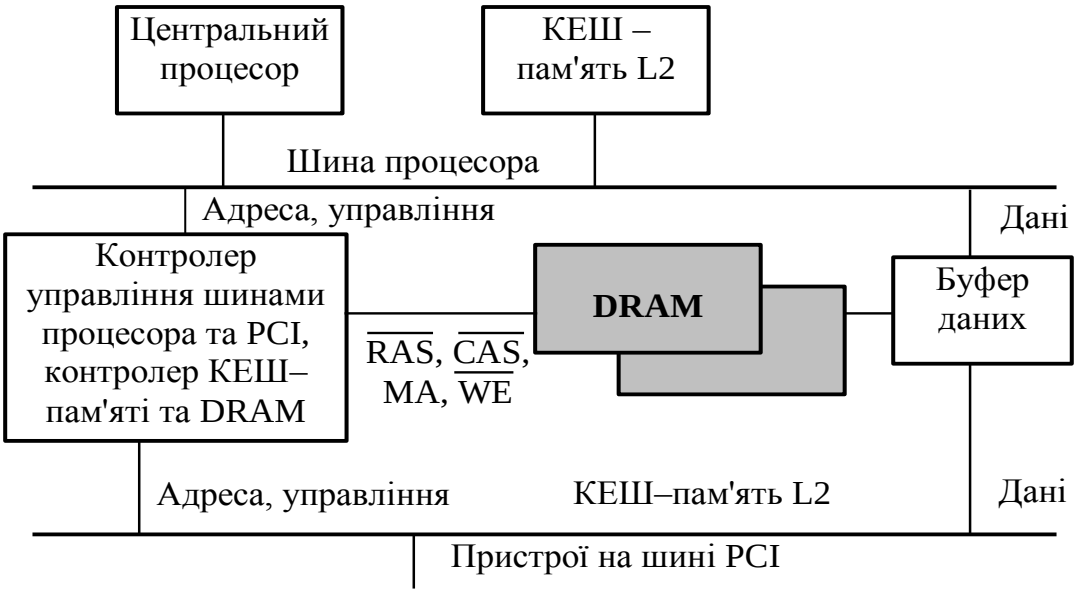


Рис. 3.11. Один з варіантів взаємодії з підсистемою пам'яті

Оскільки запитів для обміну багато, на вході підсистеми є арбітр. Він під'єднує до пам'яті пристрої відповідно до пріоритетів.

Шина між процесором і контролером *ОЗП* – ***FSB*** *(Front Side Bus)* тактується системними синхроімпульсами.

Кожний з елементів адресної групи стробується імпульсами сигналів керування ***RAS*** *(Row Address Strobe)* – адреса рядка і ***CAS*** *(Column Address Strobe)* – адреса стовпця (рис. 3.12).

Щоб стробування було надійним, ці сигнали подаються з затримкою, достатньою для завершення перехідних процесів на шині адреси і в адресних ланцюгах мікросхеми. Сигнал вибору мікросхеми CS (Crystal Select) дозволяє роботу ІМС і використовується для вибору певної мікросхеми в системах, які складаються з декількох ІМС. Вхід WE (Write Enable – дозвіл запису) визначає вид операції, яка виконується (зчитування або запис).

На якийсь час, поки ІМС пам'яті не використовує шину даних, інформаційні виходи мікросхеми переводяться в третій (високоімпедансний) стан. Управління перемиканням у третій стан забезпечується сигналом OE (Output Enable – дозвіл видачі вихідних сигналів). Цей сигнал активізується під час виконання операції читання.

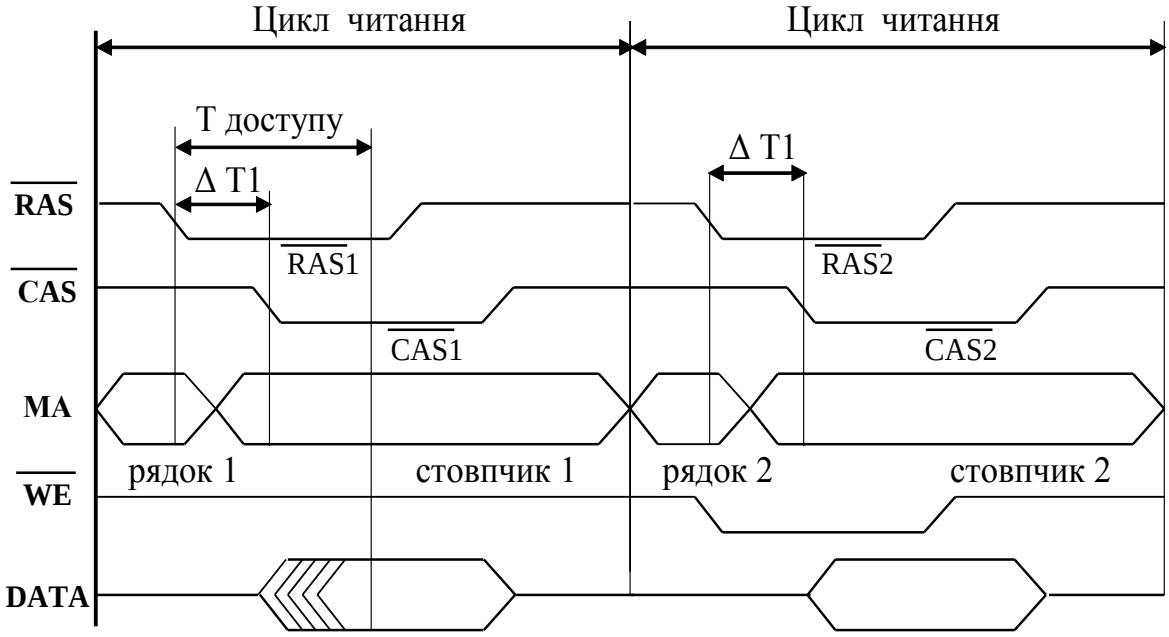


Рис. 3.12. Часова діаграма циклів доступу до ОЗП

Для більшості перерахованих вище управляючих сигналів активним зазвичай вважається їх низький рівень, що показано рискою над позначенням сигналу (рис. 3.12).

Типову процедуру доступу до пам'яті розглянемо на прикладі читання з ІМС з мультиплексуванням адрес рядків і стовпців. Спочатку на вході *WE* встановлюється рівень, відповідний операції читання, а на адресні контакти ІМС подається адреса рядка, що супроводжується сигналом *RAS*. По задньому фронту цього сигналу адреса запам'ятовується в регістрі адреси рядка мікросхеми, після чого дешифрується. Після стабілізації процесів, викликаних сигналом *RAS*, вибраний рядок підключається до підсилювачів читання/запис (ПЧЗ). Далі на вхід ІМС подається адреса стовпця, яка по задньому фронту сигналу *CAS* заноситься в регістр адреси стовпця. Одночасно готується вихідний регістр даних, куди після стабілізації сигналу *CAS* завантажується інформація з вибраних ПЧЗ.

Управління операціями з основною пам'яттю здійснюється контролером пам'яті. Зазвичай цей контролер входить до складу центрального процесора або реалізується у вигляді зовнішнього по відношенню до пам'яті пристрою. У останніх типах ІМС пам'яті частина функцій контролера покладається на мікросхему пам'яті. Хоча робота ІМС пам'яті може бути організована як по синхронній, так і по асинхронній схемі, контролер пам'яті – пристрій синхронний, тобто він спрацьовує виключно по тактових імпульсах. З цієї причини операції з пам'яттю прийнято описувати з прив'язкою до тактів.

У разі відсутності даних у кеш, доступ до ОЗП можна уявити так: за час ***першого*** і ***другого*** тактів синхронізації із шини ***FSB*** у контролер ОЗП

направляються керуючі й адресні сигнали. Сигнали аналізуються і керують логікою ОЗП.

Два-три (залежно від якості *DRAM***)** синхроімпульси витрачаються на запуск схеми дешифрації і вибір відповідного рядка.

Під час доступу до шин рядків активізується числова шина і всі *3Е* у даному рядку читаються. На розрядні шини надходять відповідні потенціали від конденсаторів.

На активізацію шин стовпців, під'єднання розрядних шин до буфера даних і витягнення із *3Е* пам'яті даних також потрібно ***два-три*** такти синхронізації. Ще ***один*** такт іде на доставку даних у буфер даних *DRAM*. По ***такту*** витрачається на доставку даних у контролер *ОЗП* і далі – у процесор.

Таким чином, ***за один цикл*** звертання до пам'яті система генерує взагалі **9 – 11 тактів синхронізації.**

Під час читання даних варто врахувати ще ***два*** такти, що витрачаються на відновлення заряду *3Е.*

Динамічна пам'ять енергозалежна і вимагає періодичного поповнення енергії в паразитних ємностях, що реалізується ***стандартною процедурою регенерації.*** Ця апаратна процедура ініціюється інтервальним таймером кожні 15,6 мкс (рис. 3.13) і виконується через канал регенерації динамічної пам'яті (РДП).

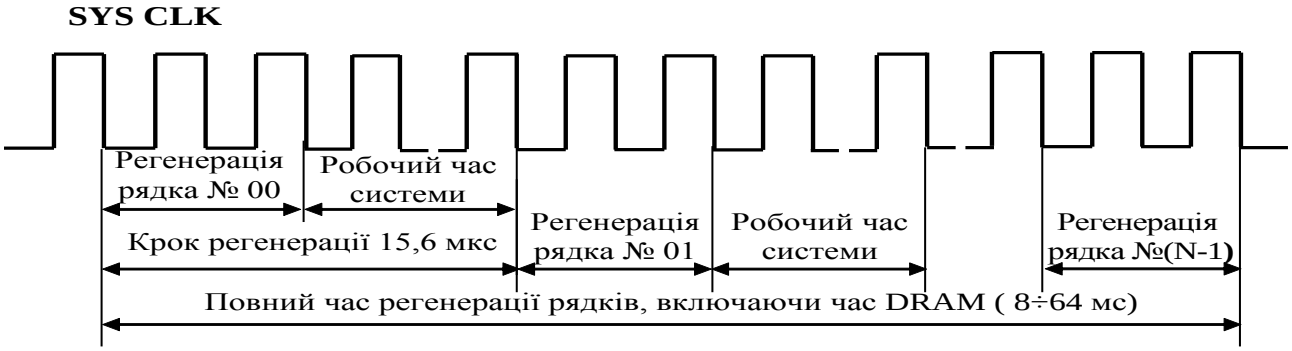


Рис. 3.13. Часова діаграма регенерації динамічної пам'яті

Для регенерації використовуються тільки строби *RAS*, а строби *CAS* у процесі не беруть участі. Протягом цього часу, який називається кроком регенерації, у *DRAM* перезаписується цілий рядок *3Е*. Так, протягом 8…64 мс обновляються всі рядки пам'яті.

Для перезапису *3Е* ОЗП досить перебирати рядок за рядком і виконувати «фіктивну» (без виведення даних на магістраль даних пам'яті) команду читання.

У цьому випадку кожен *3Е* рядка перезапишеться через схему перезарядження, а дані не потраплять у буфери виводу даних.

Шина даних знаходиться у високоімпедансному стані. На модифікацію *3Е* під час читання витрачається два такти синхронізації.

Очевидно, що процедура регенерації пам'яті (у класичному варіанті) «гальмує» роботу системи, оскільки в цей час обмін даними з ОЗП неможливий. Регенерація, заснована на звичайному перебиранні рядків (незалежно від послідовності), у сучасних типах DRAM не застосовується. Існує кілька економічних варіантів цієї процедури – розширений, пакетний, розподілений і ін.

Тимчасових характеристик динамічної пам'яті дуже багато, але найважливіших – три:

- час перезаряджання пам'яті – являє собою затримку, зв'язану з попереднім заряджанням розрядних шин опорною напругою;
- час доступу до пам'яті – активізація числової шини, у результаті чого на вихідну шину даних пам'яті викладається інформація;
- час циклу – складається з затримок часу перезаряджання і доступу.

Час затримки виведення даних DRAM вимірюється величинами від десятків до сотень наносекунд. Систему пригальмовують не тільки затримки всередині пам'яті. Будь-яке звертання до ОЗП супроводжується передачею у контролер пам'яті великої групи сигналів, що ускладнюють схемотехніку і підвищують латентність підготовчого періоду циклу обміну даними.

3.4.4. Методи підвищення швидкодії запам'ятовуючих пристроїв

Можливості «прискорення ядра» мікросхеми ЗП дуже обмежені і пов'язані в основному з мініатюризацією запам'ятовуючих елементів. Найбільші успіхи досягнуті в інтерфейсній частині ІМС, торкаються вони, головним чином, операції читання, тобто способів доставки вмісту комірки на шину даних. Найбільшого поширення набули наступні шість фундаментальних підходів: *послідовний; конвеєрний; регістровий; сторінковий; пакетний; подвоєної швидкості.*

Послідовний режим. Під час використання *послідовного режиму* (Flow through Mode) адреса і управляючі сигнали подаються на мікросхему до надходження синхроімпульсу.

У момент приходу синхроімпульсу вся вхідна інформація запам'ятовується у внутрішніх регістрах – по його передньому фронту, і починається цикл читання. Через деякий час, але в межах того ж циклу дані з'являються на зовнішній шині, причому цей момент визначається тільки моментом приходу синхронізуючого імпульсу і швидкістю внутрішніх ланцюгів мікросхеми.

Конвеєрний режим (pipelined mode) – це такий метод доступу до даних, при якому можна продовжувати операцію читання за попередньою адресою в процесі запиту до наступного.

Під час читання з пам'яті час, потрібний для витягання даних з комірки, можна умовно розбити на два інтервали. Перший з них – безпосередньо доступ

до масиву запам'ятовуючих елементів і витягання даних з комірки. Другий – передача даних на вихід (при цьому відбувається детектування стану комірки, підсилення сигналу та інші операції, необхідні для зчитування інформації). На відміну від послідовного режиму, де наступний цикл читання починається тільки після закінчення попереднього, в конвеєрному режимі процес розбивається на два етапи. Поки дані з попереднього циклу читання передаються на зовнішню шину, відбувається запит на наступну операцію читання. Таким чином, два цикли читання перекриваються в часі. Через ускладнення схеми передачі даних на зовнішню шину час зчитування збільшується на один такт і дані поступають на вихід тільки в наступному такті, але таке запізнювання спостерігається лише при першому читанні в послідовності операцій зчитування з пам'яті. Всі наступні дані поступають на вихід один за одним, хоча і з запізненням на один такт щодо запиту на читання. Оскільки цикли читання перекриваються, мікросхеми з конвеєрним режимом можуть використовуватися при частотах шини, що вдвічі перевищують допустиму для ІМС з послідовним режимом читання.

Регістровий режим (Register to Latch) використовується відносно рідко і його характерною рисою є наявність регістра на виході мікросхеми. Адреса і управляючі сигнали видаються на шину до надходження синхронізуючого імпульсу. З приходом позитивного фронту синхроімпульсу адреса записується у внутрішній регістр мікросхеми, і починається цикл читання. Зчитані дані заносяться в проміжний вихідний регістр і зберігаються там до появи негативного фронту (спаду) синхроімпульсу, а з його надходженням передаються на шину. Метод однозначно визначає момент появи даних на виході ІМС, причому змінюючи ширину імпульсу синхронізації, можна міняти час появи даних на шині. Дана властивість часто виявляється дуже корисною під час проектування спеціалізованих ОМ. За швидкодією мікросхеми з регістровим режимом ідентичні ІМС з послідовним режимом.

Сторінковий режим. В основі ідеї лежить той факт, що під час доступу до комірок з суміжними адресами (згідно з принципом локальності така ситуація найбільш вірогідна), причому до таких, де всі ЗЕ розташовані в одному рядку матриці, доступ до другої і подальших комірок можна проводити істотно швидше. Дійсно, якщо адреса рядка під час чергового звернення залишилась тією самою, то всі тимчасові витрати, пов'язані з повторним занесенням адреси рядка в регістр ІМС, дешифрацією, зарядом паразитної ємності горизонтальної лінії і тому подібне, можна виключити. Для доступу до чергової комірки досить подавати на ІМС лише адресу нового стовпця, супроводжуючи його сигналом CAS. Відзначимо, що звернення до першої комірки в послідовності проводиться стандартним чином – почерговим завданням адреси рядка і адреси стовпця, тобто тут час доступу зменшити практично неможливо. Розглянутий

режим називається *режимом сторінкового доступу* або просто *сторінковим режимом* (Page Mode). Під сторінкою розуміється рядок матриці ЗЕ. Мікросхеми, де реалізується сторінковий режим і його модифікації, прийнято характеризувати формулою *x-y-y-y*. Перше число *x* визначає кількість тактів системної шини, яке необхідне для доступу до першої комірки послідовності, а *y* – до кожної з наступних комірок. Так, вираз 7-3-3-3 означає, що для обробки першого слова необхідно 7 тактових періодів системної шини (протягом шести з яких шина простоює в очікуванні), а для обробки подальших слів – по три періоди, з яких два системна шина також простоює.

Режим швидкого сторінкового доступу (FPM - Fast Page Mode) є модифікацією стандартного сторінкового режиму. Основна відмінність полягає в способі занесення нової інформації в регістр адреси стовпця. Повна адреса (рядка і стовпця) передається тільки у разі першого звернення до рядка. Активізація буферного регістра адреси стовпця проводиться не по сигналу CAS, а по задньому фронту сигналу RAS. Сигнал RAS залишається активним впродовж всього сторінкового циклу і дозволяє заносити в регістр адреси стовпця нову інформацію не по спадаючому фронту CAS, а як тільки адреса на вході ІМС стабілізується, тобто практично по передньому фронту сигналу CAS. В цілому ж втрати часу скорочуються на два такти, які раніше були потрібні для передачі адреси кожного рядка і сигналу RAS. Реальний виграш, проте, спостерігається лише під час передачі блоків даних, що зберігаються в одному і тому ж рядку мікросхеми. Якщо ж програма часто звертається до різних областей пам'яті, переходячи з одного рядка ІМС на інший, переваги методу втрачаються. Режим знайшов широке застосування в мікросхемах ОЗП, особливо динамічного типу.

Пакетний режим (Burst Mode) – режим, при якому на запит за конкретною адресою пам'ять повертає пакет даних, що зберігаються не тільки за цією адресою, але і за декількома подальшими адресами.

Розрядність комірки пам'яті сучасних ОМ зазвичай рівна одному байту, тоді як ширина шини даних, як правило, складає чотири байти. Отже, одне звернення до пам'яті вимагає послідовного доступу до чотирьох суміжних комірок – пакета. Довжина пакета крім чотирьох може дорівнювати 1, 2 або 8-ми коміркам, які розташовані послідовно. З урахуванням цієї обставини в ІМС пам'яті часто використовується модифікація сторінкового режиму, що носить назву *групового* або *пакетного* режиму. Під час його реалізації адреса стовпця заноситься в ІМС тільки для першої комірки пакета, а перехід до чергового стовпця проводиться вже всередині мікросхеми. Це дозволяє для кожного пакета виключити три з чотирьох операцій занесення в ІМС адреси стовпця і тим самим ще більш скоротити середній час доступу.

Режим подвоєної швидкості. Важливим етапом у подальшому розвитку технології мікросхем пам'яті став режим DDR (Double Data Rate) – подвоєна швидкість передачі даних. Суть методу полягає в передачі даних по обох фронтах імпульсу синхронізації, тобто двічі за період. Таким чином, пропускна здатність збільшується в два рази.

Крім згаданих використовуються й інші прийоми підвищення швидкодії ІМС пам'яті, такі як включення до складу мікросхеми допоміжної кеш-пам'яті і незалежні тракти даних, що дозволяють одночасно проводити обмін з шиною даних і звернення до матриці ЗЕ і так далі.

Як приклад розглянемо організацію системи НОЗП-ОЗП. Вимірювання ефективності такої системи проводиться за відношенням попадання $h=a/b$, де a, b – кількість звернень, відповідно до НОЗП та ОЗП. Експериментально доведено, що відношення попадання h дуже залежить від ємності НОЗП, алгоритму зміни інформації, що використовується, і програм, які розв'язуються на ЕОМ [21].

Підвищення швидкодії може бути охарактеризоване коефіцієнтом підвищення швидкодії (КПШ)

$$K = \frac{T_0}{T_i},$$

де T_0 – цикл звернення до ЗП; T_i – цикл звернення до цього ж ЗП під час використанні i-го методу підвищення швидкодії.

Для підрахунку КПШ від введення НОЗП з циклом t_1 визначимо загальний час, необхідний для $(a + b)$ звернень до пам'яті без НОЗП та з ним:

$$T_0 \cdot (a+b) \qquad \text{і} \qquad t_1 \cdot a + T_0 \cdot b \ .$$

Звідси

$$K = \frac{T_0}{T_1} = \frac{T_0 \cdot (a+b)}{t_1 \cdot a + T_0 \cdot b} = \frac{1+h}{1+h \cdot t_1 / T_0},$$

що при $h>>1$ дає:

$$K \approx \frac{T_0}{t_1}.$$

Для дослідження залежності T_1 системи НОЗП-ОЗП від ємності НОЗП T_1 подають у вигляді $T_1 = t_1 + \varepsilon \cdot T_0$. Залежність ε від ємності m НОЗП, яка отримана методом цифрового моделювання, показана на рис. 3.14.

Як видно з рисунка, ємність НОЗП повинна приблизно дорівнювати 32-м коміркам, оскільки подальше збільшення m практично не впливає на швидкодію пам'яті типу НОЗП-ОЗП.

Наприклад, при T0 = 1000 нс та t1 = 50 нс введення НОЗП на 32 комірки підвищує швидкодію ЗП в 10 разів (T1 = 100 нс).

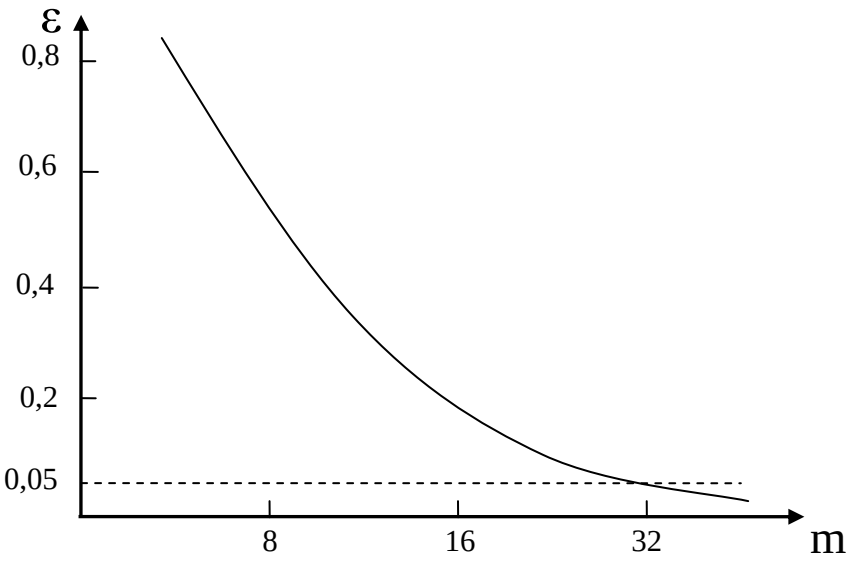


Рис. 3.14. Залежність ε від ємності НОЗП

Суттєво підвищити швидкодію можна також суміщенням операцій, яке реалізується шляхом розшарування комірок ЗП, суміщенням дешифрації адреси слова з вибіркою попереднього, застосуванням довгих комірок, здатних зберігати два або більше слів, створенням можливості звернення до ЗП багатьма незалежними адресами.

3.5. ОСНОВНІ ТИПИ ДИНАМІЧНОЇ ПАМ'ЯТІ

Динамічна пам'ять складається з ядра (масиву ЗЕ) та інтерфейсної логіки (буферних регістрів, підсилювачів читання даних, схеми регенерації та ін.). Хоча кількість видів DRAM вже перевищила два десятки, ядро в них організоване практично однаково. Головні відмінності зв'язані з інтерфейсною логікою, причому ці відмінності обумовлені також і областю використання мікросхем, наприклад, ІМС динамічної пам'яті входять і в склад відеоадаптерів.

3.5.1. Класифікація динамічної пам'яті

Основним критерієм, за яким можна класифікувати запам'ятовуючі пристрої основної пам'яті, є *спосіб синхронізації*. За цим принципом всі ОЗП поділяються на *синхронні* і *асинхронні*.

В мікросхемах, де реалізований *синхронний принцип*, процеси читання і запису виконуються одночасно з тактовими сигналами контролера пам'яті.

Робота ІМС *асинхронної пам'яті* не прив'язана жорстко до тактових імпульсів системної шини. Тому дані на цій шині з'являються в довільні моменти часу (асинхронно). Слід відзначити, що в асинхронних ЗП цикл читання починається тільки з приходом запиту від контролера пам'яті. Але оскільки контролер пам'яті (і системної шини) – пристрій синхронний, то відлік часу ведеться в тактах. І якщо дані з'являться на виходах ІМС навіть відразу

після тактового імпульсу, вони будуть оброблені тільки з приходом наступного імпульсу. Це обмежує можливості асинхронних ІМС.

Більшість з використовуваних у даний час типів мікросхем оперативної пам'яті не в змозі зберігати дані без зовнішнього джерела енергії, тобто є енергозалежними (volatile memory). Широке розповсюдження таких пристроїв пов'язане з рядом їх переваг у порівнянні з енергонезалежними типами ОЗП (non-volatile memory): більшою ємністю, низьким енергоспоживанням, вищою швидкодією і невисокою собівартістю зберігання одиниці інформації.

Енергозалежні ОЗП можна розділити на дві основні підгрупи: динамічну пам'ять (DRAM – Dynamic Random Access Memory) і статичну пам'ять (SRAM – Static Random Access Memory).

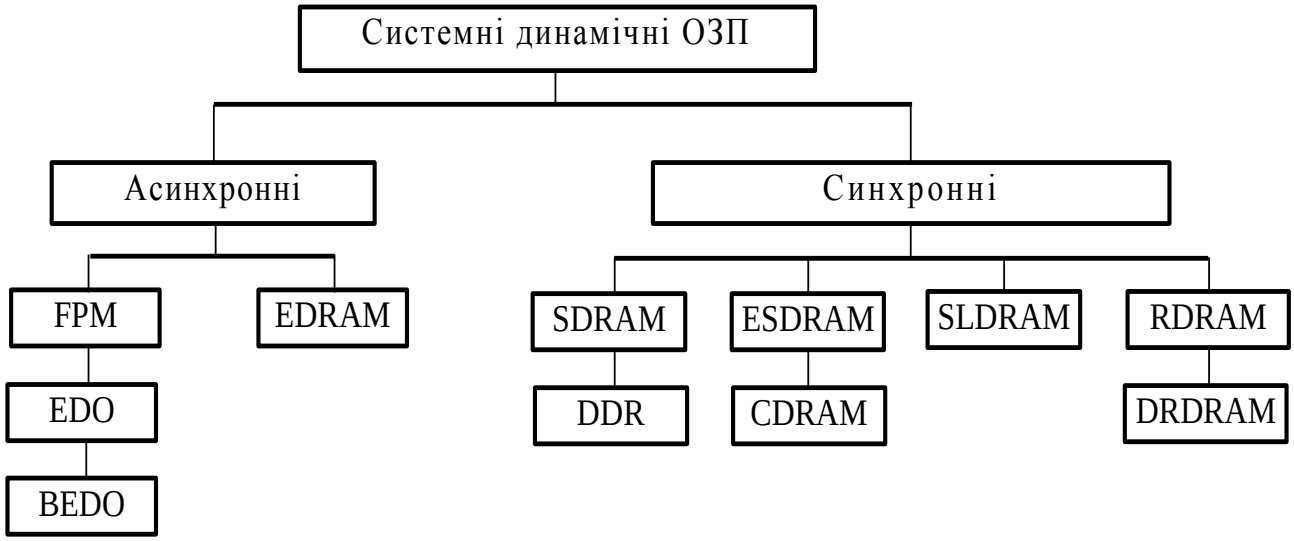


Рис. 3.15. Класифікація динамічних ОЗП

У статичних ОЗП запам'ятовуючий елемент може зберігати записану інформацію необмежено довго (за наявності напруги живлення). Запам'ятовуючий елемент динамічного ОЗП здатний зберігати інформацію тільки протягом достатнього короткого проміжку часу, після якого інформацію потрібно відновлювати наново, інакше вона буде втрачена. Динамічні ЗП, як і статичні, энергозалежні.

Роль запам'ятовуючого елемента в статичному ОЗП виконує тригер. Запам'ятовуючий елемент динамічної пам'яті значно простіший. Він складається з одного конденсатора і ключового транзистора (див. рис. 3.9).

Область застосування статичної і динамічної пам'яті визначається швидкістю і вартістю. Головною перевагою SRAM є вища швидкодія (приблизно на порядок вище, ніж у DRAM). Швидка синхронна SRAM може працювати з часом доступу до інформації, що дорівнює часу одного тактового імпульсу процесора. Проте через малу ємність мікросхем і високу вартість застосування статичної пам'яті, як правило, обмежене відносно невеликою за ємністю кеш-пам'яттю першого (L1), другого (L2) або третього (L3) рівнів. У

той же час найшвидші мікросхеми динамічної пам'яті на читання першого байта пакета все ще вимагають від п'яти до десяти тактів процесора, що уповільнює роботу всієї ОМ. Однак завдяки високій щільності упаковки ЗЕ і низькій вартості саме DRAM використовується під час побудові основної пам'яті ОМ.

Розглянемо різні типи мікросхем динамічних ОЗП системних DRAM. На початковому етапі це були мікросхеми асинхронної пам’яті. З розвитком комп'ютерної техніки зростали вимоги до ОЗП. З'явились синхронні ОЗП.

3.5.2. Асинхронні динамічні ОЗП

У перших мікросхемах динамічної пам'яті застосовувався найбільш простий спосіб обміну даними, який часто називають традиційним (***Conventional***) з робочою частотою від 4,77 до 40 Мгц. Він дозволяв читати і записувати інформацію в ЗЕ тільки за п'ять тактів: адреса рядка; сигнал RAS; адреса стовпця; сигнал CAS; операція читання-запис, яка відповідає значенню лінії WE (Write Enable).

Для Conventional загальне число тактів, які витрачаються на пересилання 4-х рядків даних (стандартна кількість при обміні даними), дорівнює 20 (5 тактів для доступу за першою адресою; 5 - за другою; 5 за третьою; 5 - за четвертою, тобто схема читання: (5-5-5-5). Тому через свою повільність вони з часом були замінені на більш прогресивні.

Пам'ять типу FPM *(Fast Page Mode)* або “*стандартна сторінкова пам'ять*” – це найбільш ранній тип пам'яті, що застосовувався у всіх 286 – 386 комп'ютерах. У ньому реалізований режим посторінкової адресації (fast page mode). Цей режим оснований на тому, що після вибору рядка в ядрі передача даних на вихід і з виходу виконується просто під'єднанням до вхідних-вихідних формувачів даних потрібного «стовпця» (стовпців, якщо розуміти під стовпцем один розряд у матриці ядра). Отже, під час повторних звернень до одного і того ж рядка ядра не потрібно подавати адресу рядка, дешифрувати його, читати рядок. У FPM підвищення швидкості обміну даними досягається завдяки передачі повної адреси (рядка і стовпця) тільки під час першого звернення до пам'яті. Під час інших звернень у межах того ж рядка вказується лише скорочена адреса (тільки стовпці). В результаті втрати часу скорочуються на два такти. Схема читання FPM тепер інша: 5-3-3-3 (разом 14 тактів) навіть на частоті 66МГц. Порівняно з *Conventional* (20 тактів) це збільшує продуктивність на цілих 70 %.

Однак якщо програма часто звертається до різних областей пам'яті, переходячи на інший рядок ядра, то формується повна адреса, що зводить переваги методу до мінімуму. На практиці часто відбувається обмін досить великими суцільними масивами даних (наприклад, багато команд процесора кодуються кількома байтами). Тому метод був покладений в основу всіх

наступних технологій, однак потрібно все-таки не забувати, що всі їхні переваги також виявляються тільки в межах однієї сторінки (рядка ядра).

Пам'ять типу EDO. Архітектура ***EDO*** *(extended data output)* або "*пам'ять з розширеним часом присутності даних на виході*" характеризується збільшеним порівняно з FPM часом збереження даних на виході мікросхеми.

Справа в тому, що в звичайних IMC FPM вихідні дані залишаються дійсними у разі активного сигналу CAS. Через це під час другого і наступного доступів до сторінки потрібно три такти: такт переключення CAS в активний стан, такт читання даних і такт перемикання CAS у неактивний стан. У IMC EDO дані запам'ятовуються у внутрішньому регістрі по активному (спадаючому) фронту сигналу CAS і зберігаються ще якийсь час після появи наступного активного фронту. Це дозволяє нормально використовувати дані, коли CAS переведений у неактивний стан. При цьому схема читання в EDO уже 5-2-2-2 (11), що на 20% швидше FPM (14). Час доступу складає 30-40 нс. Нормальна робота цієї пам'яті можлива навіть у випадку тактової частоти контролера пам'яті (і системної шини) 75 МГц.

Пам'ять EDO дотепер застосовується у всіх комп'ютерах з частотою процесора до 166 МГц (і системними платами на чипсетах до Intel 430 FX), а також у багатьох відеоприскорювачах тривимірної графіки. EDO також використовується в тих випадках, коли потужний контролер пам'яті сам оптимізує організацію банків пам'яті та їх чергування у випадку багатобанкової структури ОЗП, характерної для деяких серверів.

Пам'ять типу BEDO. В архітектурі ***BEDO*** (*burst EDO – EDO з пакетним пересиланням даних*) поряд з технологіями FPM і EDO використовується пересилання даних пакетами (burst). Новизна такого методу в тому, що під час першого звернення дані автоматично читаються відразу ж для кількох послідовних слів (адже ядро влаштоване так, що завжди читається цілий рядок, тобто всі стовпці стають відомі). При цьому для пересилання burst-пакета задаються адреса рядка і адреса тільки найпершого стовпця, а внутрішній лічильник автоматично стежить за тим, щоб був переданий весь пакет. Це виключає необхідність пересилати адреси для наступних ЗЕ.

Таким чином, завдяки burst-технології збільшується ефективність послідовного читання великих масивів даних.

Новий спосіб пересилання скорочує час читання кожного слова ще на такт, що дозволяє BEDO працювати за схемою 5-1-1-1 (разом 8 тактів). Однак для цього необхідна підтримка з боку набору системної логіки. В число таких наборів входять Intel 430 HX, VIA 580 VP, 590 VP. Максимальна паспортна робоча частота BEDO – 66 МГц, хоча IMC добре функціонують на частоті аж до 83 МГц.

Пам'ять типу EDRAM. Мікросхема EDRAM має більш швидке ядро і внутрішню кеш-пам'ять. В ролі кеш-пам'яті використовується статична пам'ять (SRAM) ємністю 2048 біт. Ядро EDRAM має 2048 стовпців, кожний з котрих з'єднаний з внутрішньою кеш-пам'яттю. Під час звернення до якої-небудь комірки одночасно зчитується весь рядок (2048 біт). Зчитаний рядок заноситься в SRAM, причому перенесення інформації в кеш-пам'ять практично не впливає на швидкодію, оскільки відбувається за один такт. Під час подальшого звернення до комірок того ж рядка дані беруться з більш швидкої кеш-пам'яті. Наступне звернення до ядра відбувається у разі доступу до комірки, що не розташована в рядку, який зберігається в кеш-пам'яті мікросхеми. Середній час доступу для мікросхеми наближається до значень, які є характерними для статичної пам'яті (близько 10 нс).

Завершуючи розгляд асинхронних типів ІМС, відзначимо, що їх швидкодію прийнято характеризувати часом циклу звертання, тобто мінімальним періодом, з яким можна виконати циклічне звертання за довільними адресами (всі п'ять операцій). Саме це мається на увазі, коли говорять про «60-нс модуль». Під час переходу до синхронної пам'яті (що використовує для роботи зовнішню тактову частоту) замість тривалості циклу доступу стали застосовувати мінімально припустимий період тактової частоти. Так з'явилися «10-нс модулі пам'яті», «8-нс» і навіть «7-нс».

3.5.3. Синхронні динамічні ОЗП

У мікросхемах, де реалізований синхронний принцип, процеси читання і запису виконуються одночасно з тактовими сигналами контролера пам'яті, а це дозволяє взяти все від пропускної здатності шини «процесор-пам'ять» і уникнути циклів очікування. Адресна і управляюча інформація фіксується в ІМС пам'яті. Після цього відповідна реакція мікросхеми відбудеться через чітко визначене число тактових імпульсів, і цей час процесор може використовувати для інших дій, які не зв'язані зі зверненням до пам'яті.

SDRAM – пам'ять. Першим поширеним типом синхронного ОЗП стала *SDRAM (Synch-ronous DRAM)* – пам'ять. Вона з'явилася тому, що пам'ять типу EDO не могла задовольнити зрослі запити процесорів з частотою вище 200 МГц у плані ефективної системи «підкачування» даних. Крім того, завдяки переходу на прогресивну технологію виробництва ціни модулів пам'яті EDO і SDRAM зрівнялися, і остання завоювала весь ринок. Саме SDRAM стала «відправною точкою» для інших технологій, у яких використовуються переваги синхронного обміну.

Кардинальні відмінності SDRAM від попередніх типів пам'яті такі:

- синхронний метод передачі даних на шину;
- конвеєрний механізм пересилання burst-пакета;

- використання кількох (двох або чотирьох) внутрішніх банків пам'яті;
- передача частини функцій контролера пам'яті логіці, що вкладена в саму IMC.

Оскільки SDRAM зараз стала домінуючим типом пам'яті, розглянемо всі ці відмінності докладніше.

Найзначнішою відмінністю SDRAM від більш ранніх типів пам'яті стала прив'язка (синхронізація) її роботи до тактових імпульсів системної шини. А оскільки частота процесора прямо зв'язана з частотою системної шини, то синхронізація дуже прискорює обмін даними між ОЗП і процесором. Синхронність дозволяє контролеру пам'яті точно «знати», коли дані готові, що знижує затримки в циклах чекання і пошуку даних. Під час синхронної роботи дані з'являються на виходах IMC пам'яті разом з тактовими імпульсами. При цьому немає тимчасової неузгодженості в роботі різних пристроїв, які беруть участь у передачі даних, що спрощує їх взаємодію.

Наступним за важливістю нововведенням можна вважати застосування конвеєрного механізму з пересилання пакета даних (*Burst Pipelining*). На відміну від простого механізму передачі пакетів, реалізованого в BEDO, конвеєр дозволяє передавати весь пакет потактно, забезпечуючи безперебійну роботу ОЗП навіть на багато більш високих частотах. Його переваги особливо виявляються у разі збільшення довжини пакета з 4 слів до розміру всього рядка банку.

Ще більшої продуктивності SDRAM досягається завдяки поділу масиву комірок на незалежні внутрішні банки пам'яті. Коли відбувається обмін даними з одним з банків, інші можуть готуватися до наступної операції. Коли ж перший банк завершить обмін, другий уже готовий передати наступний блок даних (це так званий почерговий доступ – interleaved). Таким чином, можна без затримок, зв'язаних з вибором рядка ядра, виконувати швидке послідовне читання даних з різних областей пам'яті.

Для того щоб синхронізувати пам'ять ззовні, в чип була вмонтована логіка керування. Таким чином, вузли пам'яті перебрали ряд функцій контролера ОЗП.

Синхронізм дозволяє мікросхемі SDRAM виконувати функції незалежно від затримок зовнішніх сигналів керування. Впровадження в чип механізму керування і спрощення схемотехніки скоротили накладні витрати часу доступу до ОЗП.

Отже, чип SDRAM став деякою подобою мікроконтролера з вбудованими елементами динамічної пам'яті. Мікросхема містить швидкісну синхронну шину даних. Для доступу до даних використовується триступенева конвеєрна архітектура з інтерлівом.

Спрощена структура SDRAM подана на рис. 3.16.

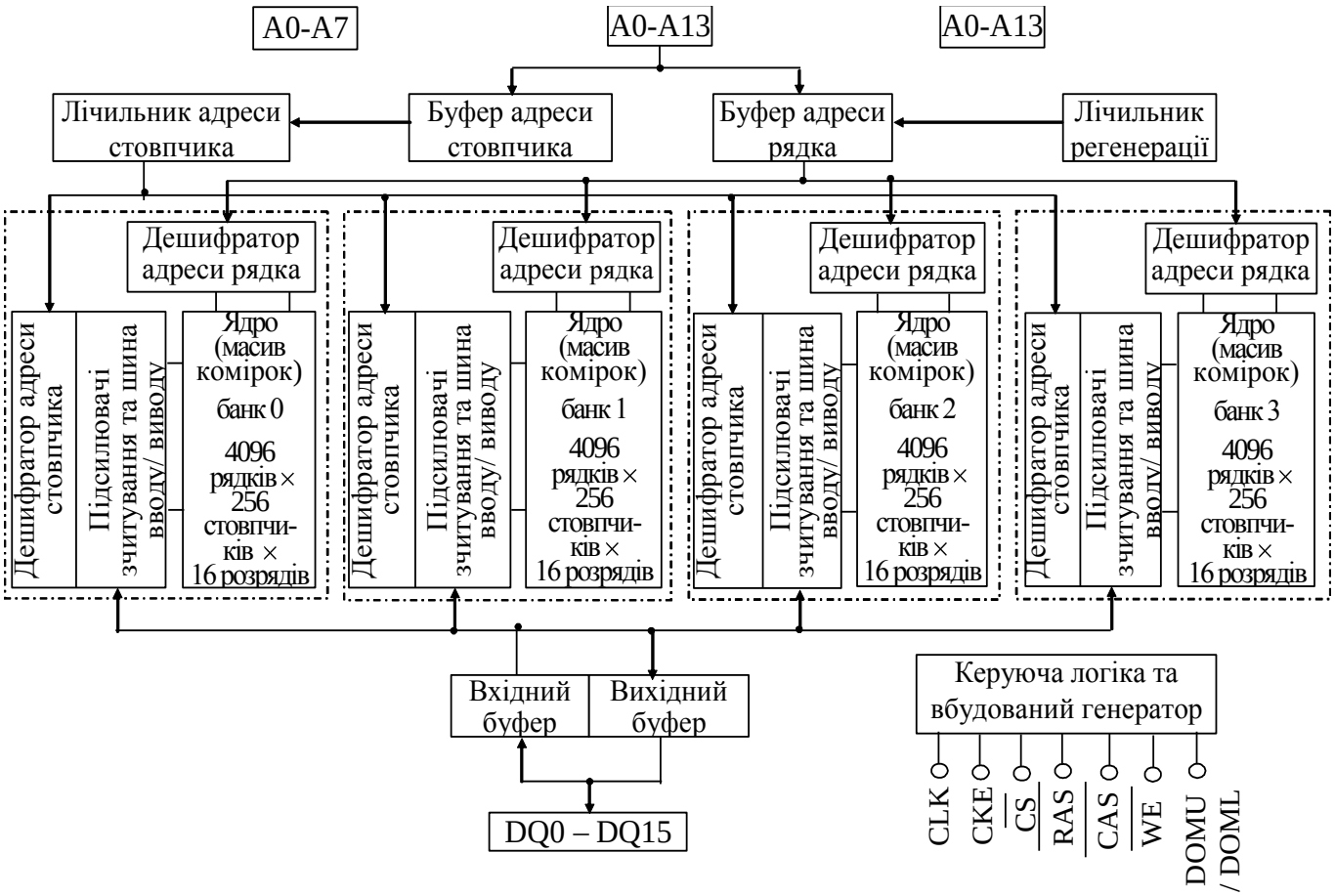


Рис.3.16. Структура SDRAM

SDRAM містить запам'ятовуючий масив, регістри, дешифратори і лічильник адреси стовпців. На вибраному рядку лічильник послідовно перебирає адреси стовпців, і за кожен такт із регістрів даних каналу вводу/виводу у вихідний буфер переміщається по вісім байтів даних. Таким чином, за чотири звертання до пам'яті читається повний рядок (32 байти). Перше звертання до восьми байтів (у разі промаху кеш) займає 9–11 тактів. На кожне наступне витрачається по такту.

Команди завантажуються в пам'ять SDRAM через програмувальний регістр режиму. Регістр дозволяє запрограмувати пам'ять для вибору режиму регенерації, автоматичної вибірки суміжно розташованих комірок пам’яті у межах банку, а також застосування інтерліву. Може бути запрограмована довжина пакета і змінений час затримки стробів CAS. Настроювання цієї затримки позначається на продуктивності системи.

Конвеєрна обробка пакетів даних дозволяє виключити зі списку тимчасових втрат дуже істотні позиції. Внутрішнє виконання команд знімає із системних шин частину навантаження і прискорює процес доступу до даних.

Архітектура синхронної пам'яті має модифікації.

DDR SDRAM - пам'ять. Важливим етапом у подальшому розвитку технології SDRAM стала ***DDR SDRAM*** (*Double Data Rate SDRAM* – SDRAM з подвійною швидкістю передачі даних). Цей тип пам'яті «ріднить» з

SDR SDRAM (*Single Data Rate SDRAM*) загальний тракт обробки адрес і сигналів керування, однакова конструкція банків пам'яті і загальних методів регенерації. Інше позначення DDR SDRAM пам'яті – SDRAM II (тобто SDRAM другого покоління). В табл. 3.1 приводяться відмінні риси SDRAM архітектури SDR і DDR. Основні відмінності цих технологій – в організації інтерфейсу даних і системі синхронізації. DDR працює на подвоєній частоті.

Таблиця 3.1

Особливості, що відрізняють SDRAM архітектури SDR та DDR

Особливості SDR SDRAM	Особливості DDR SDRAM
Інформація тактується тільки фронтом зовнішнього синхроімпульсу	Ввід команд тактується фронтом зовнішнього синхроімпульсу, а дані тактуються як фронтом, так і зрізом
Внутрішня та зовнішня шини даних однакової ширини, дані потрапляють на внутрішню шину через буфер вводу/виводу	Внутрішня шина даних ширше зовнішньої вдвічі, що дозволяє надсилати дані з комірок до буфера вводу/виводу попарно
	Для скорішого завершення операції читання використовується команда Burst Terminate, у звичайному режимі читання вихідний буфер закритий
За допомогою сигналу DQM (Data Strobe Mask) маскуються дані під час записування та керується вихідний буфер під час читання	За допомогою сигналу DQM (Data Strobe Mask) маскуються дані під час запису
	За допомогою сигналу DQS (Data Strobe) відбувається захоплення даних

За принципами роботи DDR SDRAM пам'ять схожа на SDRAM, але, на відміну від неї, може приймати і передавати дані в пакетному режимі на обох фронтах тактових імпульсів. Це подвоює швидкість передачі даних. Крім того, в DDR SDRAM використовується протокол DLL (Delay Locked Loop), який дозволяє пересунути в часі інтервал дійсного значення вихідних даних. Таким способом скорочуються простої системної шини під час читання даних на ній з декількох модулів пам'яті. Є мікросхеми DDR SDRAM з можливістю відключення схем DLL. Для цього в них є додатковий регістр режиму. Відключення DLL необхідне під час зниження тактової частоти з метою енергозбереження.

Існує декілька специфікацій DDR SDRAM залежно від тактової частоти системної шини: DDR 266, DDR 333, DDR 400, DDR 533. Так, пікова пропускна здатність мікросхеми пам'яті специфікації DDR 333 складає 2,7 Гбайт/с, а для DDR 400 – 3,2 Гбайт/с.

Розвитком DDR SDRAM стала розроблена організацією JEDEC (Joint Electron Device Engineering Council - Об'єднана рада по електронних пристроях) специфікація DDR2.

В пам'яті ***DDR2 SDRAM*** обмін даними здійснюється також на подвійній частоті синхронізації. Однак в DDR2 тактові частоти вищі (200-300 МГц), що потребує особистих вимог до передачі сигналів. Для виключення відводів від шин печатних провідників, які є необхідними під час використання корпусів з дворядним розташуванням виводів, мікросхеми DDR2 випускаються тільки в корпусах BGA з матрицею виводів. Крім того, мікросхеми мають внутрішні резистори-термінатори, призначені для покращання якості сигналів, які передаються на високих частотах.

Пам'ять DDR2 SDRAM розроблялась з урахуванням зворотної логічної сумісності: контролер DDR2 може працювати як з пам'яттю DDR, так і з DDR2. Набір команд DDR2 є розширенням набору DDR.

Мікросхеми DDR2 з частотою 200, 266, 333 і 400 МГц позначаються як DDR2-400, DDR2-533, DDR2-667 і DDR2-800.

Подальшим розвитком пам'яті DDR SDRAM стала ***DDR3 SDRAM***, яка прийшла на зміну DDR2 SDRAM. В DDR3 зменшено на 40% споживання енергії порівняно з модулями DDR2 SDRAM, що обумовлено зменшеною (1,5 В, в порівнянні з 1,8 В для DDR2 SDRAM та 2,5 В для DDR SDRAM) напругою живлення гнізд пам'яті.

Для локальної пам'яті графічних адаптерів використовуються спеціальні типи SDRAM з більш високими тактовими частотами і деякими особливостями часових діаграм:

GDDR SDRAM (Graphics Double Data Rate - подвійна швидкість передачі графічних даних) з тактовими частотами 166-300 МГц, забезпечує пропускну здатність 333-600 Мбіт/с на вивід, напруга живлення – 2,5-2,8 В. Логічно ця пам'ять відповідає звичайній пам'яті DDR SDRAM з довжиною пакета – 2, 4 або 8.

GDDR2 SDRAM з тактовими частотами 266-333 МГц забезпечує пропускну здатність 533-667 Мбіт/с на вивід. Напруга живлення – 1,8В. Цей тип пам'яті приблизно відповідає пам'яті DDR2 SDRAM: є додаткова програмована латентність, довжина пакета – 4 або 8.

GDDR3 SDRAM значно відрізняється від DDR2. Вона має практично таке ж технологічне ядро, як і DDR2, але показники споживання енергії та тепло-виділення були незначно знижені, забезпечуючи вищу швидкодію модулів пам'яті та спрощуючи систему охолодження. На відміну від використання

пам'яті DDR2 в графічних картах, пам'ять GDDR3 відмінна за внутрішніми особливостями від пам'яті DDR3 запровадженою JEDEC. Ця пам'ять використовує внутрішній термінатор, що дозволяє їй краще відповідати деяким вимогам роботи із графікою. Характеристики GDDR3 SDRAM: латентність запису програмована; програмована додаткова затримка. Довжина пакета – 4 або 8. Тактові частоти – 500-800 МГц, пропускна здатність – 1-2,6 Гбіт/с на контакт. Живлення – 1,8 або 2 В (для найбільш швидкодіючих).

DDR2 SDRAM та DDR3 SDRAM на сьогодні є найбільш розповсюдженими типами динамічної пам'яті персональних ОМ.

DR DRAM – пам'ять. Найбільш очевидні способи підвищення ефективності роботи процесора з пам'яттю – збільшення тактової частоти шини або ширини вибірки (кількості розрядів, що одночасно пересилаються). На жаль, спроби поєднання обох варіантів натрапляють на істотні технічні труднощі (з підвищенням частоти посилюються проблеми електромагнітної сумісності, важче стає забезпечити одночасність надходження споживачеві всіх бітів інформації, що паралельно пересилаються). В більшості синхронних DRAM (SDRAM, DDR) застосовується широка вибірка (64 біти) при обмеженій частоті шини.

Принципово відмінний підхід до побудови DRAM був запропонований компанією Rambus в 1997 році. В ньому наголос зроблено на підвищення тактової частоти до 400 Мгц при одночасному зменшенні ширини вибірки до 16 біт. Нова пам'ять відома як RDRAM (Rambus Direct RAM). Існує декілька різновидів цієї технології: Base, Concurrent і Direct. В них тактування ведеться по обидва фронти синхросигналів (як в DDR), завдяки чому результуюча частота складає відповідно 500-600, 600-700 і 800 Мгц. Два перші варіанти практично ідентичні, а ось зміни в технології Direct Rambus (DR DRAM) дуже значні.

DR DRAM *(Direct Rambus DRAM)* – різновид швидкої динамічної пам'яті з довільним доступом (рис. 3.17).

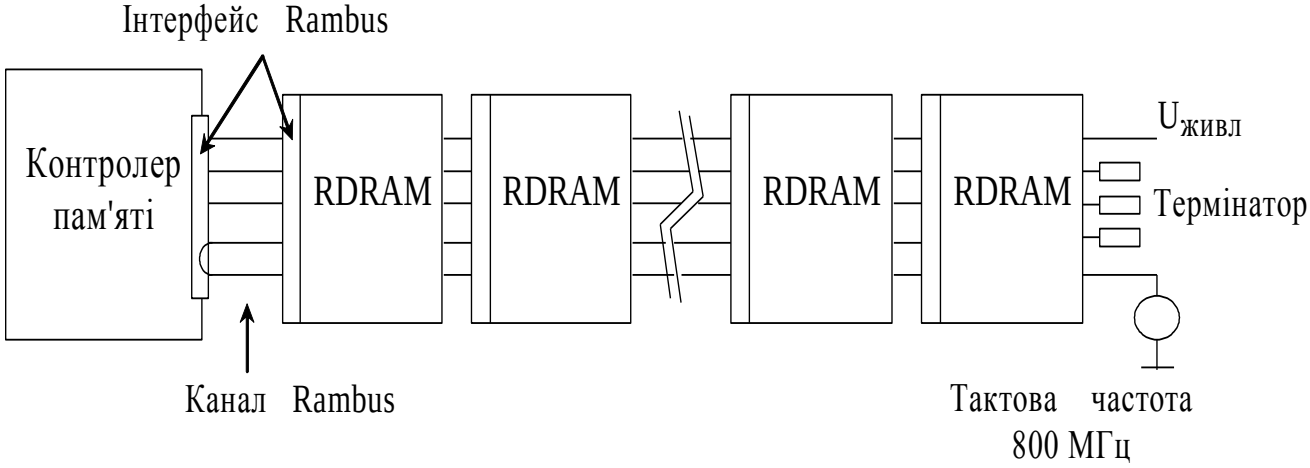


Рис. 3.17. Принцип організації пам'яті DR DRAM

Основа архітектури Rambus – банки пам'яті, «пронизані» швидкісним каналом. Канал являє собою електричну шину, що під'єднує елементи пам'яті до контролера і роз'ємів та застосовує асинхронний блочно-орієнтований протокол. «Канал Rambus» входить в модуль на одному його кінці, проходить через усі чіпи і виходить на іншому кінці модуля.

Шина даних синхронізується від зовнішнього джерела 400 МГц, як і DDR SDRAM**,** фронтом і зрізом, завдяки чому тактова частота синхронізації пам'яті – 800 МГц.

На логічному рівні інформація між контролером і пам'яттю передається пакетами. Розрізняють три види пакетів: пакети даних, пакети рядків і пакети стовпців. Пакети рядків і стовпців служать для передачі від контролера пам'яті команд управління відповідно лініями рядків і стовпців масиву запам'ятовуючих елементів. Ці команди замінюють звичайну систему управління мікросхемою за допомогою сигналів RAS, CAS, WE і CS.

Масив ЗЕ розбитий на банки. Їх число в кристалі ємкістю 64 Мбіт складає 8 незалежних або 16 здвоєних банків. У здвоєних банках пара банків використовує загальні підсилювачі читання/запису. Внутрішнє ядро мікросхеми має 128-розрядну шину даних, що дозволяє за кожною адресою стовпця передавати 16 байт. Під час запису можна використовувати маску, в якій кожен біт відповідає одному байту пакета. За допомогою маски можна вказати, скільки байтів пакета і які саме повинні бути записані в пам'ять.

Чітка робота на високій швидкості досягається завдяки цілому комплексу заходів:

- малій амплітуді сигналів (Rambus Signaling Logic з 0,8 В);
- добре продуманій топології шини;
- щільній установці мікросхем на модулі;
- установці термінаторів на кінцях доріжок, щоб гасити відображення.

Застосування декількох тактових імпульсів, які розповсюджуються у протифазі, зводить до мінімуму вплив перешкод. Спеціальна схема тактування мікросхем дає можливість погодити напрямок поширення тактових імпульсів. Це дозволяє надійно читати інформацію з ІМС незалежно від того, наскільки вони віддалені від контролера.

Контролер Direct Rambus посилає керуючі сигнали по шині і подає команди для регенерації ядра. Складається він із двох окремих функціональних блоків: Rambus Application Specific Integrated Circuit Cell (RAC) і контролера Rambus. Роль RAC – забезпечити фізичний і електричний інтерфейс контролера з зовнішньою шиною, а 8-розрядний контролер Rambus служить ядром 16-розрядного контролера Direct Rambus.

Шина Direct Rambus Channel з'єднує чипи пам'яті один з одним і модулі RIMM з контролером. Вона складається з трьох незалежних шин: 16-розрядної

шини даних і двох шин адреси (окремо для рядків і для стовпців) загальною «шириною» 8 біт. Передбачено і резервні розряди для адрес, що дозволяє нарощувати об'єм встановленої в комп'ютері пам'яті до максимуму – 1 Гб.

Сигнали керування внутрішньою логікою передаються пакетами по 8 слів. А мінімальний пакет даних складається з 16 байт, завдяки чому, як і у випадку з SDRAM, підвищується продуктивність при послідовному читанні великих масивів даних. Однак під час читання врозкид ефективність пам'яті падає.

Завдяки конвеєру, потік даних малими порціями розподіляється між банками таким чином, що втрати часу під час звернення до пам'яті мінімальні. Розподіл даних залежить від швидкості заповнення кожного банку. Велике число банків дозволяє ефективно використовувати синхронну внутрішню високошвидкісну магістраль даних. Два канали даних (кожен шириною по одному байту) дозволяють одержати пікову пропускну здатність вихідної шини даних до 3,2 Гб/с.

В публікаціях згадується робота Intel і Rambus над новою версією RDRAM, названою nDRAM, яка підтримуватиме передачу даних з частотами до 1600 Мгц [25].

Дуже висока початкова вартість пам'яті RDRAM призвела до того, що виробники потужних комп'ютерів віддали перевагу менш потужній, але більш дешевшій пам'яті DDR SDRAM.

SLDRAM - пам'ять. Потенційним конкурентом RDRAM на роль стандарту архітектури пам'яті для персональних ОМ виступає новий вид динамічного ОЗП, розроблений консорціумом виробників ОМ SyncLink Consortium і відомий під абревіатурою *SLDRAM*. На відміну від RDRAM, технологія якої є власністю компаній Rambus та Intel, даний стандарт – відкритий. На системному рівні технології дуже схожі. Дані і команди від контролера до пам'яті та назад в SLDRAM передаються пакетами по 4 або 8 посилок.

SLDRAM *(Synchronous Link DRAM)* – це також синхронна пам'ять. Як і DDR, вона синхронізує дані фронтом і зрізом тактового імпульсу. Керуюча інформація, як і дані, передається в пам'ять пакетами, що зменшує латентність. Пам'ять SLDRAM містить розширений файл програмованих регістрів і оперує великою кількістю команд.

Підвищення продуктивності досягається за рахунок поширення пакетного протоколу передачі даних на сигнали керування (від чого і пішла назва цього типу пам'яті Linked SDRAM). В SLDRAM адреси, команди, а також сигнали керування передаються в пакетному режимі по односпрямованій 10-розрядній шині Command Link. Дані передаються теж у пакетному режимі по двонаправленій 18-розрядній шині даних, причому передача відбувається на

обох фронтах тактових імпульсів, як і у випадку з DDR SDRAM. Величина всього пакета даних може дорівнювати цілій сторінці (рядку ядра). Оскільки пропускна здатність обох шин (команд і даних) однакова, можна перемикатися на іншу сторінку пам'яті без втрати продуктивності.

Порівняно з SDRAM набір команд у SLDRAM значно збільшений, що дуже полегшує роботу контролера. Команда являє собою чотири 10-бітних пакети (*це і є Link*) і містить всю інформацію для проведення наступної операції. Таким чином, зростає ефективність керування пам'яттю – усього за 4 такти передається вся інформація, що описує цілий масив даних. Це викликає стрибок у продуктивності SLDRAM.

Максимальна досяжна для нинішніх поколінь SLDRAM швидкість передачі перевищує 1 Гб/с на кожен розряд при частоті 400 МГц (що забезпечує ефективну частоту більше 800 МГц). Треба відзначити, що при такій частоті дуже важливо, щоб усі сигнали точно синхронізувалися з тактовими імпульсами системної шини і щоб усі мікросхеми пам'яті в межах одного модуля мали близькі тимчасові затримки. Для цього контролер, до якого можна підключити до 8 мікросхем пам'яті, програмує всі IMC модуля пам'яті так, щоб вони видавали дані на шину одночасно, незалежно від розкиду їх параметрів і ступеня віддалення мікросхем від контролера. В результаті найвіддаленіша мікросхема видає дані без затримки, а найближча – через проміжок часу, що потрібний для того, щоб сигнал розповсюдився від найвіддаленішої до найближчої. Ці значення визначаються в момент подачі живлення на IMC і постійно коректуються під час роботи.

Пам'ять типу ESDRAM. Цей тип пам'яті був розроблений компанією Enhanced Memory Systems, унаслідок чого одержав назву Enhanced DRAM. Спочатку з'явився *EDRAM* (з асинхронним інтерфейсом), а потім, з появою SDRAM, був розроблений *ESDRAM* (із синхронним).

Основні його відмінності від SDRAM – більш швидке ядро (час доступу – 27 нс замість стандартних 60); продуктивність, підвищена майже до рівня статичного ОЗП, за ціною динамічного; схована регенерація; гнучке використання кеш-пам'яті для забезпечення максимальної продуктивності під час різних типів звертань.

Найважливіша відмінність ESDRAM від інших IMC у тому, що в ній сполучаються два типи пам'яті – динамічний (з нього складається ядро) і статичний (для кеш-пам'яті).

Принцип роботи ESDRAM у тому, що з динамічної в кеш-пам'ять цілком переноситься весь рядок, у якому знаходиться ЗЕ, що читається. Після цього читання відбувається вже з кеш-пам'яті, а в ядрі в цей час можна вибирати потрібний рядок або проводити регенерацію. Перенесення майже не позначається на швидкодії, оскільки відбувається усього за один такт.

Завдяки вбудованій у ESDRAM кеш-пам'яті швидкість витягування даних порівняно зі звичайною динамічною пам'яттю збільшується уп'ятеро (12 нс проти 60). Що ж до операції запису, то вона, на відміну від читання, відбувається в обхід кеш-пам'яті, що збільшує продуктивність ESDRAM при поновленні читання з раніше вже завантаженого в кеш рядка.

Оскільки внутрішніх банків два, простої через їхню підготовку до операцій читання/запису зводяться до мінімуму. За сигналами і типорозмірами модулі ESDRAM цілком сумісні з DIMM SDRAM.

Недолік ESDRAM пам'яті – ускладнення контролера: він повинний враховувати можливість підготовки до читання в кеш-пам'ять нового рядка ядра. Крім того, при довільній послідовності адрес кеш-пам'ять використовується вкрай неефективно. Цього недоліку немає в іншому типі пам'яті – CDRAM.

Пам'ять типу CDRAM. Цей тип ОЗП розроблений у корпорації Mitsubishi і являє собою перероблений варіант ESDRAM, звільнений від деяких її недоліків. Зміни торкнулися кеш-пам'яті – її об'єму, принципу розміщення даних, засобів доступу.

В CDRAM об'єм одного блока даних, що поміщається в кеш, зменшений до 128 біт. Це дозволяє використовувати кеш-пам'ять набагато ефективніше, ніж в ESDRAM. Адже в цьому випадку в 16-кілобітному кеші можуть одночасно зберігатися копії з 128 різних ділянок пам'яті, що дозволяе ефективніше використовувати кеш-пам'ять. Заміна першої поміщеної в кеш ділянки пам'яті починається лише після заповнення останього (128-го) блока.

Необхідність керувати різнорідними типами пам'яті ще більш ускладнює контролер, однак ефективність кеш-пам'яті, яка розміщена «всередині» IMC, вища, ніж при традиційній архітектурі, тому що перенесення в кеш здійснюється блоками, у вісім разів більшими, ніж при видачі «назовні» з IMC DRAM.

Крім того, потрібно відзначити, що в IMC використовуються роздільні адресні лінії для статичного кеш і динамічного ядра. Оскільки перенесення з DRAM у SRAM пов'язане з видачею даних на шину, то часті, але короткі пересилання не знижують продуктивності всієї IMC під час зчитування з пам'яті великих об'ємів інформації і вирівнюють CDRAM з ESDRAM, а при вибірковому читанні перевага залишається за CDRAM.

3.5.4. Модулі пам'яті типу DRAM

Елементи пам'яті типу DRAM конструктивно виконуються або у вигляді окремих мікросхем у корпусі типу DIP, або у вигляді модулів пам'яті типу: SIP (Single In-line Package), SIMM (Single In-line Memory Module), DIMM (Dual In-line Memory Module), RIMM (Rambus In-line Memory Module). Мікросхеми в

корпусах типу DIP випускалися до використання модулів пам'яті. Ці мікросхеми мають два ряди контактів, розташованих вздовж довгих сторін чипа, і загнуті донизу.

Модулі SIP. Модулі типу SIP являють собою прямокутні плати із контактами у вигляді невеликих штирьків. Цей тип пам'яті на даний час практично не використовується, оскільки був витіснений модулями пам'яті типу SIMM.

Модулі SIMM. Модулі типу SIMM являють собою прямокутну плату із контактною полосою вздовж однієї із сторін, модулі фіксуються в роз'ємі поворотом за допомогою защібок. Найбільш поширені 30- і 72- контактні SIMM. Найчастіше вживаними були модулі на 4, 8, 16, 32 і навіть 64 Мбайт.

Модулі DIMM. Модулі типу DIMM найбільш поширені у вигляді 168-контактних модулів, встановлюваних в роз'єми вертикально і зафіксовуються защібками. В портативних пристроях широко використовуються SO DIMM – різновид DIMM малого розміру (англ. SO – small outline), які в першу чергу призначалися для портативних комп'ютерів. Найчастіше зустрічаються 72- і 144- контактні модулі типу SO DIMM. Пам'ять типу DDR SDRAM випускається у вигляді 184-контактних DIMM-модулів.

Модулі DDR2 DIMM схожі на звичайні DDR DIMM, проте мають велику кількість контактних виводів і по іншому розташовані установочні зазори, які запобігають неправильному використанню модулів DDR2. Наприклад, різне фізичне розташування зазорів не дозволить встановити модуль пам'яті DDR2 в роз'єм DDR (або SDR). Модулі DDR2 мають 240 контактних виводів, тобто значно більше, ніж модулі DDR або SDRAM DIMM.

Модулі RIMM. У модулях RIMM випускається пам'ять типу Direct RDRAM. Щоб забезпечити нормальне функціонування системи із сигналами, що вимірюються частками наносекунд (1,25 нс), була розроблена спеціальна топологія шини модуля пам’яті. Цей модуль назвали RIMM – Rambus In-line Memory Module (за аналогією з SIMM і DIMM). Сучасні RIMM можуть містити до 128 банків пам’яті. Особливість цього модуля – однакова довжина всіх сигнальних доріжок, завдяки цьому до всіх IMC модуля сигнали приходять одночасно. Справа в тому, що під час роботи на частоті 800 МГц величина RIMM рівняється з довжиною хвилі (близько 37 см), що ставить певні вимоги до розташування елементів на модулі.

Модулі RIMM представлені 168/184-контактними прямокутними платами, які обов'язково повинні встановлюватися виключно парами, а порожні роз'єми обов'язково повинні бути зайняті спеціальними заглушками. Це пов'язано із особливостями конструкції таких модулів. Також існують модулі 232-pin PC1066 RDRAM RIMM 4200, які не сумісні із 184-контактним роз'ємом.

3.6. ПОСТІЙНІ ЗАПАМ'ЯТОВУЮЧІ ПРИСТРОЇ

Слово «постійні» в назві цього виду запам'ятовуючих пристроїв відноситься до їх властивості зберігати інформацію за відсутності напруги живлення. Мікросхеми постійних запам’ятовуючих пристроїв (ПЗП) також побудовані за принципом матричної структури накопичувача, де у вузлах розташовані перемички у вигляді провідників, напівпровідникових діодів або транзисторів, що одним кінцем підключені до адресної лінії, а іншим – до розрядної лінії зчитування. У такій матриці наявність перемички може означати 1, а її відсутність – 0. У деяких типах ПЗП елемент, розташований на перемичці, виконує роль конденсатора. Тоді заряджений стан конденсатора означає 1, а розряджений – 0.

Основним режимом роботи ПЗП є зчитування інформації, яке мало відрізняється від аналогічної операції в ОЗП як по організації, так і по тривалості. Саме цю обставину підкреслює загальновизнана назва постійних ЗП – ROM (Read-Only Memory – пам'ять тільки для читання). В той же час запис у ПЗП в порівнянні з читанням зазвичай складніший і зв'язаний з великими витратами часу і енергії. Занесення інформації в ПЗП називають програмуванням або «прошивкою». Остання назва нагадує про те, що перші ПЗП виконувалися на базі магнітних осердь, а дані в них заносилися шляхом прошивки відповідних осердь провідниками зчитування. Сучасні ПЗП реалізуються у вигляді напівпровідникових мікросхем, які за можливостями і способом програмування розділяють на таки види:

- програмовані під час виготовлення;
- однократно програмовані після виготовлення;
- багатократно програмовані.

3.6.1. ПЗП, що програмуються під час виготовлення

Цю групу утворюють так звані масочні пристрої і саме до них прийнято застосовувати абревіатуру ПЗП. В літературі поширеніше позначення різних варіантів постійних ЗП скороченнями від англійських назв, тому надалі також використовуватимемо аналогічну систему. Для масочних ПЗП таким позначенням є ROM, яке збігається із загальною назвою всіх типів ПЗП. Іноді такі мікросхеми іменують MROM (Mask Programmable ROM – ПЗП, що програмуються за допомогою маски).

Занесення інформації в масочні ПЗП складає частину виробничого процесу і полягає в підключенні або непідключенні запам'ятовуючого елементу до розрядної лінії зчитування. Залежно від цього із ЗЕ завжди витягуватиметься 1 або 0. У ролі перемички виступає транзистор, розташований на перетині адресної і розрядної ліній. Які саме ЗЕ повинні бути підключені до вихідної

лінії, визначає маска, що «закриває» певні ділянки кристала. Під час створення масочних ПЗП застосовуються різні технології. В першому випадку маска просто не допускає металізації ділянки, що з'єднує транзистор з розрядною лінією зчитування. Друга технологія зв'язана з видом транзистора у вузлі. Маска визначає, який польовий транзистор повинен бути імплантований в даний вузол, що працює в збагаченому режимі або в режимі збіднення. В третьому варіанті маска задає товщину оксидного шару затвора транзистора. Залежно від цього на кристалі формується або стандартний транзистор, або транзистор з високим порогом спрацьовування.

У початковий період масочні мікросхеми були дорогі, проте зараз це один з найбільш дешевих видів ПЗП. Для ROM характерна висока щільність упаковки ЗЕ на кристалі і високі швидкості зчитування інформації. Основною сферою застосування є пристрої, що вимагають зберігання фіксованої інформації. Так, подібні ПЗП часто використовують для зберігання шрифтів у лазерних принтерах.

3.6.2. Однократно програмовані ПЗП

Створення масок для ROM виправдане у процесі виробництва великого числа копій. Якщо потрібна відносно невелика кількість мікросхем з даною інформацією, розумною альтернативою є однократно програмовані ПЗП.

Мікросхеми PROM. В ІМС типу PROM (Programmable ROM – *програмовані ПЗП*) інформація може бути записана тільки однократно. Першими такими ПЗП стали мікросхеми пам'яті на базі плавких запобіжників. У початковій мікросхемі у всіх вузлах адресні лінії з'єднані з розрядними. Занесення інформації в PROM проводиться електрично, шляхом перепалювання окремих перемичок, і може бути виконано постачальником або споживачем через якийсь час після виготовлення мікросхеми. Подібні ПЗП випускалися в рамках серій К556 і К1556. Пізніше з'явилися ІМС, де в перемичку входили два діоди, з'єднані назустріч. У процесі програмування видалити перемичку можна було за допомогою електричного пробою одного з цих діодів. У будь-якому варіанті для запису інформації потрібне спеціальне устаткування – програматори. Основними недоліками даного виду ПЗП були великий відсоток браку і необхідність спеціального термічного тренування після програмування, без якого надійність зберігання даних була невисокою.

Мікросхеми OTP EPROM. Ще один вид однократно програмованого ПЗП – це OTP EPROM (One Time Programmable EPROM – EPROM з однократним програмуванням). В його основі лежить кристал EEPROM (див. нижче), але поміщений у дешевий непрозорий пластиковий корпус без кварцевого вікна, через що він може бути запрограмований лише один раз.

3.6.3. Багатократно програмовані ПЗП

Процедура програмування таких ПЗП зазвичай припускає два етапи: спочатку проводиться стирання вмісту всіх або частини комірок, а потім проводиться запис нової інформації.

В цьому класі постійних запам'ятовуючих пристроїв виділяють декілька груп:

- EPROM (Erasable Programmable ROM - стирані програмовані ПЗП);
- EEPROM (Electrically Erasable Programmable ROM – електрично стирані програмовані ПЗП);
- флеш-пам'ять.

Мікросхеми EPROM. В EPROM запис інформації проводиться електричними сигналами, так само як в PROM, проте перед операцією запису вміст всіх комірок повинен бути приведений до однакового стану (стерто) шляхом дії на мікросхему ультрафіолетовим опромінюванням. Кристал поміщений в керамічний корпус, який має невелике кварцеве вікно, через яке і проводиться опромінювання. Щоб запобігти випадковому стиранню інформації, після опромінювання кварцеве вікно заклеюють непрозорою плівкою. Процес стирання може виконуватися багато разів. Кожне стирання займає близько 20 хв.

Дані зберігаються у вигляді зарядів плаваючих затворів МОН-транзисторів, що грають роль конденсаторів з дуже малим витоком заряду. Заряджений ЗЕ відповідає логічному нулю, а розряджений – логічній одиниці. Програмування мікросхеми відбувається з використанням технології інжекції гарячих електронів. Цикл програмування займає декілька сотень мілісекунд. Час зчитування близький до показників ROM і DRAM.

У порівнянні з PROM мікросхеми EPROM дорожче, але можливість багатократного перепрограмування часто є визначальною. Даний вид ІМС випускався в рамках серії К573 (зарубіжний аналог – серія 27ххх).

Мікросхеми EEPROM. Привабливішим варіантом багатократно програмованої пам'яті є електрично стирана програмована постійна пам'ять EEPROM. Стирання і запис інформації в цю пам'ять проводяться побайтово, причому стирання – не окремий процес, а лише етап, що відбувається автоматично під час запису. Операція запису займає істотно більше часу, чим зчитування – декілька сотень мікросекунд на байт. У мікросхемі використовується той же принцип зберігання інформації, що і в EPROM. Програмування EEPROM не вимагає спеціального програматора і реалізується засобами самої мікросхеми.

Випускаються два варіанти мікросхем: з послідовним і паралельним доступом, причому на частку перших припадає 90% всіх ІМС цього типу. В EEPROM з доступом по послідовному каналу (SEEPROM – Serial EEPROM)

адреси, дані і управляючі команди передаються по одному провіднику і синхронізуються імпульсами на тактовому вході. Перевагою SEEPROM є малі габарити і мінімальне число ліній вводу/виводу, а недоліком – великий час доступу.

В цілому EEPROM дорожче, ніж EPROM, а мікросхеми мають менш щільну упаковку комірок, тобто меншу ємність.

Флеш-пам'ять. Відносно новий вигляд напівпровідникової пам'яті – це *флеш-пам'ять* (назву flash можна перевести як «спалах блискавки», що підкреслює відносно високу швидкість перепрограмування). Вперше анонсована в середині 80-х років, флеш-пам'ять багато в чому схожа на EEPROM, але використовує особливу технологію побудови запам'ятовуючих елементів. Аналогічно EEPROM, у флеш-пам'яті стирання інформації проводиться електричними сигналами, але не побайтово, а по блоках або повністю. Тут слід зазначити, що існують мікросхеми флеш-пам'яті з розбиттям на дуже дрібні блоки (сторінки) і автоматичним посторінковим стиранням, що зближує їх за можливостями з EEPROM. Як і у випадку з EEPROM, мікросхеми флеш-пам'яті випускаються у варіантах з послідовним і паралельним доступом.

За організацією масиву ЗЕ розрізняють мікросхеми типу:

- Bulk Erase (тотальне очищення) – стирання допустиме тільки для всього масиву ЗЕ;
- Boot Lock – масив роздільний на декілька блоків різного розміру, вміст яких може очищатися незалежно;
- Flash File – масив роздільний на декілька рівноправних блоків однакового розміру, вміст яких може стиратися незалежно.

Повністю вміст флеш-пам'яті може бути очищений за одну або декілька секунд, що значно швидше, ніж у EEPROM. Програмування (запис) байта займає час близько 10 мкс, а час доступу під час читання складає 35-200 нс.

Як і в EEPROM, використовується тільки один транзистор на біт, завдяки чому досягається висока щільність розміщення інформації на кристалі (на 30% вище, чим у DRAM**).**

3.6.4. Енергонезалежні оперативні запам'ятовуючі пристрої

Під поняття *енергонезалежний* ОЗП (NVRAM - Non-Volatile RAM) підпадає декілька типів пам'яті. Від перепрограмованих постійних ЗП їх відрізняє відсутність етапу стирання, що передує запису нової інформації, тому замість терміну «програмування» для них вживають стандартний термін «запис».

Мікросхеми BBSRAM. До даної групи відносяться звичайні статичні ОЗП із вбудованим літієвим акумулятором і посиленим захистом від спотворення інформації в момент включення і відключення зовнішнього

живлення. Для їх позначення застосовують абревіатуру BBSRAM (Battery-Back SRAM).

Мікросхеми NVRAM. Інший підхід реалізований у мікросхемі, розробленій компанією Simtec. Особливість її в тому, що в одному корпусі об'єднано статичний ОЗП і перепрограмована постійна пам'ять типу EEPROM. Під час включення живлення дані копіюються з EEPROM в SRAM, а під час виключення – автоматично перезаписуються з SRAM в EEPROM. Завдяки такому прийому даний вид пам'яті можна вважати незалежним.

Мікросхеми FRAM. FRAM (Ferroelectric RAM – фероелектрична пам'ять) розроблена компанією Ramtron. За швидкодією даний ЗП дещо поступається динамічним ОЗП і поки розглядається лише як альтернатива флеш-пам'яті. Віднесення FRAM до оперативних ЗП обумовлено відсутністю перед записом явно вираженого циклу стирання інформації.

Запам'ятовуючий елемент FRAM схожий на ЗЕ динамічного ОЗП, тобто складається з конденсатора і транзистора. Відмінність полягає в діелектричних властивостях матеріалу між обкладками конденсатора. В FRAM цей матеріал володіє великою діелектричною постійною і може бути поляризований за допомогою електричного поля. Поляризація зберігається аж до її зміни протилежно направленим електричним полем. Дані зчитуються за рахунок дії на конденсатор електричного поля. Величина струму, що виникає при цьому, залежить від того, чи змінює прикладене поле напрям поляризації на протилежний чи ні, що може бути зафіксоване підсилювачами зчитування. В процесі зчитування вміст ЗЕ руйнується і повинен бути відновлений шляхом повторного запису, тобто як і DRAM, даний тип ЗП потребує регенерації. Кількість циклів перезапису для FRAM зазвичай складає 10 млрд.

Головна перевага даної технології в значно вищій швидкості запису в порівнянні з EEPROM. В той же час відносна простота ЗЕ дозволяє добитися високої щільності розміщення елементів на кристалі.

3.7. СПЕЦІАЛЬНІ ТИПИ ОПЕРАТИВНОЇ ПАМ'ЯТІ

У ряді практичних завдань вигіднішим виявляється використання спеціалізованої архітектури ОЗП, де стандартні функції (запис, зберігання, зчитування) поєднуються з деякими додатковими можливостями або враховують особливості застосування пам'яті. Такі види ОЗП називають спеціалізованими і до них відносять:

- пам'ять для відеоадаптерів;
- пам'ять з множинним доступом (багатопортові ОЗП);
- пам'ять типу черги (ОЗП типу FIFO).

Два останні типи відносяться до статичних ОЗП.

3.7.1. Оперативні запам'ятовуючі пристрої для відеоадаптерів

Використання пам'яті в відеоадаптерах має свою специфіку і для реалізації додаткових вимог звертаються до дещо інших типів мікросхем. Так, під час створення динамічних зображень часто досить просто змінити розташування інформації, що вже зберігається у відеопам'яті. Замість того щоб багато разів пересилати по шині одні і ті ж дані, лиш дещо змінивши їх розташування, вигідніше змусити мікросхему пам'яті перемістити дані, що вже в ній зберігаються, з однієї області ядра в іншу. На ІМС пам'яті можна також покласти операції по зміні кольору точок зображення.

Стисло розглянемо деякі з типів ОЗП, орієнтованих на застосування як відеопам'ять.

Мікросхеми SGRAM. Абревіатура SGRAM (Synchronous Graphic DRAM – синхронний графічний динамічний ОЗП) позначає спеціалізований вид синхронної пам'яті з підвищеною внутрішньою швидкістю передачі даних. SGRAM може самостійно виконувати деякі операції над відеоданими, зокрема блочний запис. Передбачено два режими такого запису. В першому – режимі блочного запису (Block Write) – можна змінювати колір відразу восьми елементів зображення (пікселів). Призначення другого режиму – блочного запису з маскуванням певних бітів (Masked Write або Write-per-Bit) – запобігти зміні кольору для окремих пікселів блока, що пересилається. Є також модифікація даної мікросхеми, відома як DDR SGRAM, відмінність якої очевидна з приставки DDR. Використання обох фронтів синхросигналів веде до відповідного підвищення швидкодії ІМС.

Мікросхеми VRAM. ОЗП типу VRAM (Video RAM) відрізняється високою продуктивністю і призначений для потужних графічних систем. Під час розробки ставилося завдання забезпечити постійний потік даних у разі оновлення зображення на екрані. Для типових значень роздільності і частоти оновлення зображення інтенсивність потоку даних наближається до 200 Мбіт/с. В таких умовах процесору важко отримати доступ до відеопам'яті для читання або запису. Щоб вирішити цю проблему, в мікросхемі зроблені істотні архітектурні зміни, які дозволяють відособити обмін між процесором і ядром VRAM для читання/запису інформації і операції по видачі інформації на схему формування відеосигналу (ЦАП – цифро-аналоговий перетворювач). Зв'язок пам'яті з процесором забезпечується паралельним портом, а з ЦАП – додатковим послідовним портом. Крім того, динамічне ядро DRAM доповнене пам'яттю з послідовним доступом (SAM – Serial Access Memory) ємністю 4 Кбайт. Обидва види пам'яті зв'язані між собою широкою внутрішньою шиною. Інформація, що виводиться на екран, порціями по 4 Кбайт з ядра пересилається в SAM і вже звідти, в послідовному коді (послідовний код формується за

допомогою підключених до SAM регістрів зсуву), поступає на ЦАП. У момент перезапису в SAM нової порції ядро VRAM повністю готове до обслуговування запитів процесора. Разом з цими режимами мікросхема реалізує режим Flash Write, що дозволяє очистити цілий рядок пам'яті. Є також можливість маскувати певні комірки, захищаючи їх від запису.

Мікросхеми WRAM. Даний вид мікросхем, розроблений компанією Samsung, багато в чому схожий на VRAM. Це також двопортова пам'ять, що допускає одночасний доступ з боку процесора і ЦАП, але по конструкції вона дещо простіша, ніж VRAM. Наявні в VRAM, але рідко використовувані функції виключені, а замість них введені додаткові функції, прискорюючі вивід на екран тексту і заповнення одним кольором великих площ екрана. В WRAM застосована швидкісніша схема буферизації даних і збільшена розрядність внутрішньої шини. Прискорено також ядро мікросхеми, за рахунок використання режиму швидкісного сторінкового режиму (UFP - Ultra Fast Page), що забезпечує час доступу близько 15 нс. В середньому WRAM на 50% продуктивніше, ніж VRAM, і на 20% дешевше. Застосовується мікросхема в могутніх відеоадаптерах.

Мікросхеми MDRAM. Мікросхема типу MDRAM (Multibank DRAM – багатоблоковий динамічний ОЗП) розроблена компанією MoSys і орієнтована на графічні карти. Пам'ять містить багато незалежних банків по 1К 32-розрядних слів кожен. Банки підключені до швидкої і широкої внутрішньої шини. Кожен банк може виконувати певні операції незалежно від інших банків. Відмова будь-якого з банків веде лише до скорочення сумарної ємності пам'яті і деякого зниження показників швидкодії. Завдяки блоковій побудові технологія дозволяє виготовляти мікросхеми практично будь-якої ємності, не обов'язково кратної степеню числа 2.

Мікросхеми 3D-RAM. Цей тип пам'яті розроблений спільно компаніями Mitsubishi і Sun Microsystems з орієнтацією на тривимірні графічні прискорювачі. Крім масиву запам'ятовуючих елементів, мікросхема 3D-RAM (тривимірна RAM) містить процесор (арифметико-логічний пристрій) і кеш-пам'ять. Процесор дозволяє виконувати деякі операції із зображенням прямо в пам'яті. Основні перетворення над пікселами реалізуються за один такт, оскільки стандартна послідовність дій «зчитав, змінив, записав» зводиться до однієї операції – «змінити», виконуваної в момент запису. Процесор мікросхеми дозволяє за секунду виконати близько 400 млн. операцій по обробці даних і зафарбувати до 4 млн. елементарних трикутників. Кеш-пам'ять забезпечує більш рівномірне навантаження на процесор під час інтенсивних обчислень. Ядро 3D-RAM складається з чотирьох банків загальною ємністю 10 Мбіт. Розмір рядків пам'яті вибраний таким, щоб у межах однієї і тієї ж області пам'яті знаходилися якомога більше тривимірних об'єктів. Це дає

можливість заощадити час на переходи з рядка на рядок. За ціною даний тип мікросхем порівнянний з VRAM.

3.7.2. Багатопортові ОЗП

Стандартний однопортовий ОЗП має по одній шині адреси, даних і управління і в кожен момент часу забезпечує доступ до комірки пам'яті тільки одному пристрою.

На відміну від стандартного в *n*-портовому ОЗП є *n* незалежних наборів шин адреси, даних і управління, які гарантують одночасний і незалежний доступ до ОЗП *n* пристроям. Дана властивість дозволяє істотно спростити створення багатопроцесорних і багатомашинних обчислювальних систем, де багатопортовий ОЗП виступає в ролі загальної або спільно використовуваної пам'яті. В рамках однієї ОМ подібний ОЗП може забезпечувати обмін інформацією між ЦП і ПВВ (наприклад, контролером магнітного диска) набагато ефективніше, ніж прямий доступ до пам'яті. В даний час серійно випускаються дво- і чотирипортові мікросхеми. Оскільки архітектурні рішення в обох випадках схожі, розглянемо організацію двопортових ОЗП.

У двопортовій пам'яті є два набори адресних, інформаційних і управляючих сигнальних шин, кожен з яких забезпечує доступ до загального масиву ЗЕ (рис. 3.18). Оскільки двопортовому ОЗП властива симетрична структура, надалі набори шин називатимемо «лівим» (Л) і «правим» (П). В цілому організація матриці ЗЕ залишається традиційною.

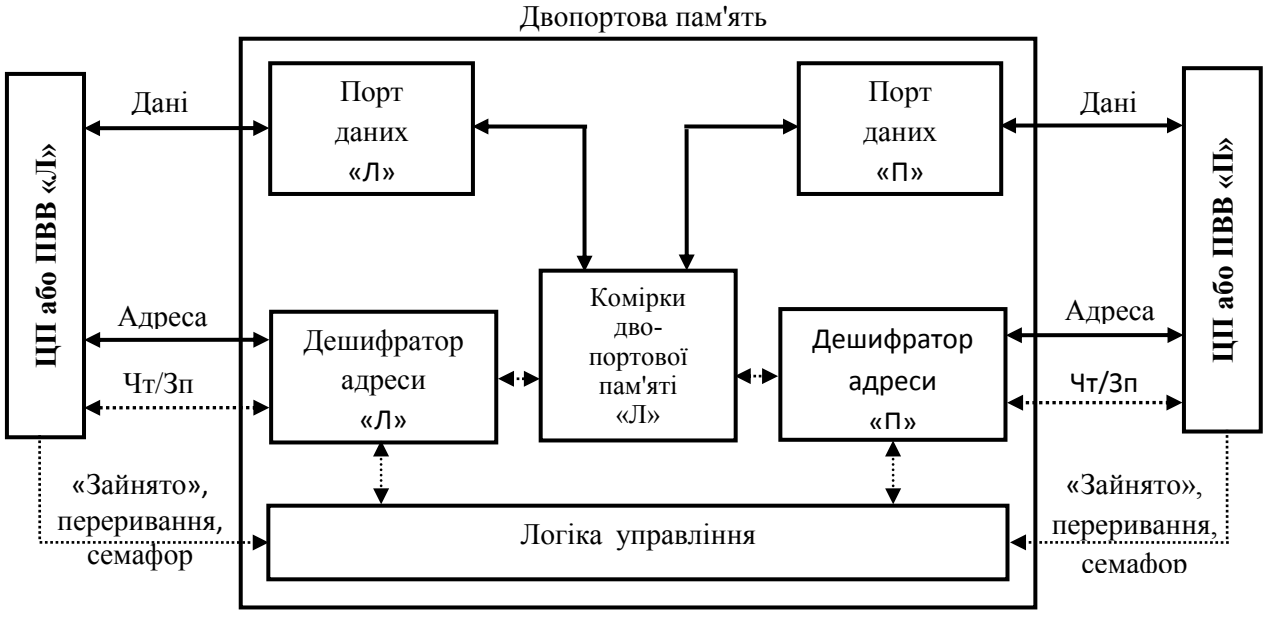


Рис. 3.18. Структура двопортового ОЗП

Доступ до комірок можливий як через ліву, так і через праву групу шин, причому якщо Л- і П-адреси різні, ніяких конфліктів не виникає. Проблеми потенційно можливі, коли Л- і П-пристрої одночасно звертаються за однією і

тією ж адресою і хоча б один з цих пристроїв намагається виконати операцію запису. В цьому випадку, якщо один з портів читає інформацію, а інший проводить запис в ту ж комірку, вірогідне зчитування недостовірної інформації. Під час спроби одночасного введення в комірку з двох напрямів в неї може бути занесена невизначена комбінація із записуваних слів. Незважаючи на те що вірогідність подібних ситуацій по оцінках не перевищує 0,1%, такий варіант необхідно враховувати, для чого в двопортовій пам'яті є схема арбітражу з використанням сигналів «Зайнято».

Логіка арбітражу в мікросхемі реалізована апаратними засобами, які забезпечують формування сигналу «Зайнято», що забороняє запис у комірку для тієї половини, на якій адреса з'явиться пізніше, а також ухвалення рішення на користь одного з вхідних портів при одночасному надходженні адрес.

Крім можливості доступу до комірок з двох напрямів, двопортова пам'ять забезпечується засобами для обміну повідомленнями між підключеними до неї пристроями: системою переривання і системою семафорів. Першу із них називають апаратною, а другу – програмною.

В системі переривань двопортової пам'яті дві останні комірки мікросхеми (з найбільшими адресами) використовуються як «поштові скриньки» для обміну повідомленнями між пристроями, підключеними до Л- і П-портів. Повідомленню від лівого пристрою виділена комірка з парною адресою (якщо ємність пам'яті рівна 1К, то це буде адреса $3FF_{16}$), а від правого – з непарним ($3FE_{16}$). Коли пристрій записує інформацію в свою «поштову скриньку», формується запит переривання до пристрою, підключеного до протилежного порту. Цей сигнал автоматично скидається, коли адресат прочитує інформацію зі своєї «поштової скриньки».

Система семафорів – це наявний у двопортовій пам'яті набір з восьми тригерів, стан яких може бути прочитаний і змінений з боку будь-якого з портів. Тригери грають роль програмних семафорів або прапорів, за допомогою яких Л- і П-пристрої можуть сповіщати один одного про якісь події. Суть цих подій не зафіксована і визначається програмами, що реалізуються. Зазвичай семафори потрібні для надання одному з процесорів монопольного права роботи з певним блоком даних до завершення всіх необхідних операцій з цим блоком. У цьому випадку процесор, що монополізує блок даних, встановлює один з семафорів у стан 1, а після закінчення – в 0. Другий процесор, перш ніж звернутися до даного блока, прочитує семафор і у разі одиничного стану останнього повторює прочитування і аналіз семафора до тих пір, поки перший процесор не встановить його в стан 0. Природно, що в програмному забезпеченні Л- і П-процесорів розподіл і правила використання семафорів повинні бути узгоджені.

Часто однієї мікросхеми багатопортової пам'яті не вистачає через недостатню ємність однієї ІМС або зважаючи на малу розрядність комірок. У

обох випадках необхідно з'єднати декілька мікросхем, відповідно паралельно або послідовно. Якщо декілька мікросхем об'єднуються в ланцюжок для досягнення потрібної розрядності слова, виникає проблема з арбітражем під час одночасного звернення до однієї і тієї ж комірки. Для виключення подібної ситуації мікросхеми багатопортової пам'яті випускаються в двох варіантах: ведучі (master) і ведені (slave). Ухвалення рішення проводиться тільки у ведучих мікросхемах, а ведені функціонують відповідно до інструкції, отриманої від ведучого. Таким чином, у ланцюжку використовується тільки одна мікросхема типу «ведучий», а усі інші ІМС повинні мати тип «ведений».

3.7.3. Пам'ять типу FIFO

У багатьох випадках ОЗП застосовується для буферизації потоку даних, коли дані зчитуються з пам'яті в тій же послідовності, в якій вони туди заносилися, але надходження і зчитування відбуваються з різною швидкістю. Часто для цієї мети застосовують звичайний ОЗП, проте тут одночасний запис і зчитування інформації неможливі. Ефективнішим видом ОЗП, де обидві дії можуть вестися одночасно, служить пам'ять типу FIFO. Мікросхема є двопортовим ОЗП, де один порт призначений для занесення інформації, а другий – для зчитування. Для FIFO-пам'яті характерні всі технологічні прийоми, властиві двопортовій пам'яті, зокрема способи арбітражу під час одномоментного звернення до однієї і тієї ж комірки. В той же час є і істотні відмінності.

Перша – у мікросхеми немає входів для вказівки адреси комірки, занесення і зчитування даних проводиться в порядку їх надходження через одну вхідну точку і одну вихідну.

Друга відмінність пов'язана з необхідністю стеження за станом черги. Для цього в мікросхемі є регістри покажчики адрес початку і кінця черги, а також спеціальні прапорці, які вказують на дві ситуації: відсутність даних (у цьому випадку блокується зчитування з мікросхеми) і повне заповнення пам'яті (блокується запис).

3.8. ОРГАНІЗАЦІЯ КЕШ-ПАМ'ЯТІ

Кеш-пам'ять персональних комп'ютерів є високошвидкісним буфером, побудованим на мікросхемах SRAM (Static RAM – статична оперативна пам'ять), який безпосередньо обмінюється даними з процесором. Така пам'ять наявна у всіх 32-розрядних сучасних процесорах.

Як уже наголошувалося, як елементна база основної пам'яті в більшості ОМ використовуються мікросхеми динамічних ОЗП, швидкодія яких на порядок нижче швидкодії центрального процесора. В результаті процесор вимушений простоювати декілька тактових періодів, поки інформація з ІМС

пам'яті встановиться на шині даних ОМ. Якщо ОП виконати на швидких мікросхемах статичної пам'яті, вартість ОМ істотно зросте. Економічно прийнятне вирішення цієї проблеми було запропоноване М. Уїлксом у 1965 році в процесі розробки ОМ Atlas і полягає воно у використанні дворівневої пам'яті, коли між ОП і процесором розміщується невелика, але швидкодіюча буферна пам'ять. У процесі роботи такої системи в буферну пам'ять копіюються ті ділянки ОП, до яких проводиться звернення з боку процесора. В загальноприйнятій термінології – проводиться *відображення* ділянок ОП на буферну пам'ять. Виграш досягається за рахунок раніше розглянутої властивості локальності – якщо відобразити ділянку ОП в більш швидкодіючу буферну пам'ять і переадресувати на неї всі звернення в межах скопійованої ділянки, можна добитися істотного підвищення продуктивності ОМ.

Уїлкс називав дану буферну пам'ять підпорядкованою (slave memory). Пізніше поширення набув термін *кеш-пам'ять* (від англійського слова *cache* – притулок, тайник), оскільки така пам'ять зазвичай прихована від програміста в тому сенсі, що він не може її адресувати і може навіть взагалі не знати про її існування. Вперше кеш-системи з'явилися в машинах моделі 85 сімейств IBM 360.

3.8.1. Загальні питання кешування пам'яті

У загальному вигляді використання кеш-пам'яті пояснимо таким чином. Коли ЦП намагається прочитати слово з основної пам'яті, спочатку здійснюється пошук копії цього слова в кеші. Якщо така копія існує, звернення до ОП не проводиться, а в ЦП передається слово, що є вибраним з кеш-пам'яті. Дану ситуацію прийнято називати успішним зверненням або *попаданням* (hit). За відсутності слова в кеші, тобто у разі неуспішного звернення – *промаху* (miss), – необхідне слово передається в ЦП з основної пам'яті, але одночасно з ОП в кеш-пам'ять пересилається блок даних, що містить це слово.

На рис. 3.19 приведена структура системи з основною і кеш-пам'яттю.

ОП складається з 2n слів, що адресуються, де кожне слово має унікальну n-розрядну адресу. Під час взаємодії з кешем ця пам'ять розглядається як M блоків фіксованої довжини по K слів у кожному (M = 2n/K). Кеш-пам'ять складається із C блоків аналогічного розміру (блоки в кеш-пам'яті прийнято називати рядками), причому їх число значно менше числа блоків у основній пам'яті (C << M). У разі зчитування слова з якого-небудь блока ОП цей блок копіюється в один з рядків кеша. Оскільки число блоків ОП більше числа рядків, окремий рядок не може бути постійно виділений одному і тому ж блоку ОП. З цієї причини кожному рядку кеш-пам'яті відповідає тег (ознака), що містить відомості про те, копія якого блока ОП у даний момент зберігається в даному рядку.

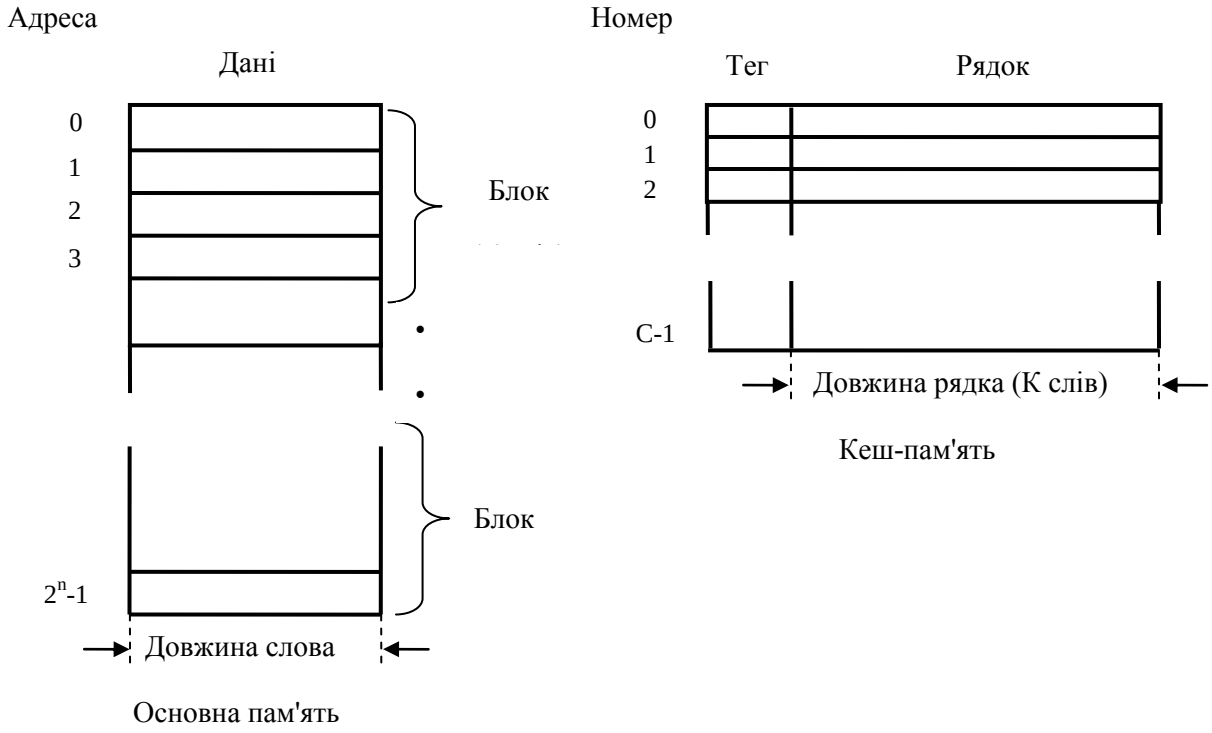


Рис. 3.19. Структура системи з основною і кеш-пам'яттю

Інформація про те, який саме блок займає даний рядок, і про його стан зберігається у пов'язаній з даним рядком комірці спеціальної пам'яті тегів (tag RAM). Для процесорів i486 і старших процесорів P5 довжина рядка збігається з об'ємом даних, що передаються за один пакетний цикл (для i486 це – 4 x 4 = 16 байт, для Pentium – 4 x 8 = 32 байт). Можливий також варіант секторованого (sectored) кеша, в якому один рядок містить кілька суміжних комірок – секторів. Їх розміри відповідають мінімальній порції обміну даних кеша з оперативною пам'яттю. При цьому в записі каталогу, що відповідає кожному рядку, повинні зберігатися біти дійсності для кожного сектора даного рядка. Секторування економить пам'ять, яка необхідна для зберігання каталогу у разі збільшення ємності кеша, оскільки більша кількість бітів каталогу відводиться під тег.

На ефективність застосування кеш-пам'яті в ієрархічній системі пам'яті впливає цілий ряд моментів. До найбільш істотних з них можна віднести:

- ємність кеш-пам'яті;
- розмір рядка;
- спосіб відображення основної пам'яті на кеш-пам'ять;
- алгоритм заміщення інформації в заповненій кеш-пам'яті;
- алгоритм узгодження вмісту основної і кеш-пам'яті;
- число рівнів кеш-пам'яті.

Вибір ємності кеш-пам'яті − це завжди певний компроміс. З одного боку, кеш-пам'ять повинна бути достатньо мала, щоб її вартісні показники були близькі до величини, характерної для ОП. З іншої – вона повинна бути

достатньо великою, щоб середній час доступу в системі, що складається з основної і кеш-пам'яті, визначався часом доступу до кеш-пам'яті. Чим більша ємність кеш-пам'яті, тим більше логічних схем повинні брати участь в її адресації. Як наслідок, ІМС кеш-пам'яті підвищеної ємності працюють повільніше в порівнянні з мікросхемами меншої ємності, навіть якщо вони виконані за однією і тією ж технологією.

Реальна ефективність використання кеш-пам'яті залежить від характеру вирішуваних завдань, і неможливо заздалегідь визначити, яка її ємність буде дійсно оптимальною. Незважаючи на очевидні протиріччя, є видимою і загальна тенденція: у міру збільшення ємності кеш-пам'яті вірогідність промахів спочатку істотно знижується, але досягши певного значення ефект згладжується і стає неістотним [25]. Встановлено, що для більшості завдань близькою до оптимальної є кеш-пам'ять ємністю від 1 до 512 Кбайт.

Ще одним важливим чинником, що впливає на ефективність використання кеш-пам'яті, служить розмір рядка. Коли в кеш-пам'ять поміщається рядок, разом з необхідним словом туди потрапляють і сусідні слова. У міру збільшення розміру рядка вірогідність промахів спочатку падає, оскільки в кеш, згідно принципу локальності, потрапляє все більше даних, які знадобляться найближчим часом. Проте вірогідність промахів починає рости, коли розмір рядка стає надмірно великим. Пояснюється це тим, що:

- великі розміри рядка зменшують загальну кількість рядків, які можна завантажити в кеш-пам'ять, а мале число рядків приводить до необхідності частої їх зміни;
- у міру збільшення розміру рядка кожне додаткове слово виявляється далі від запитаного, тому таке додаткове слово менш імовірно знадобиться в найближчому майбутньому.

Залежність між розміром рядка і вірогідністю промахів багато в чому визначається характеристиками конкретної програми, через що важко рекомендувати певне значення величини рядка. Практика показує, що найбільш близьким до оптимального можна визнати розмір рядка, рівний 4-8 одиницям, що адресуються (словам або байтам). На практиці розмір рядка зазвичай вибирають рівним ширині шини даних, що пов'язує кеш-пам'ять з основною пам'яттю.

У табл. 3.2 наведено динамічні параметри кешів різних рівнів у сучасних комп'ютерах фірм Intel та AMD.

Кеш-пам'ять в сучасних комп'ютерах будується за дворівневою схемою, але може бути і трирівневою.

Первинний (внутрішній) кеш (L1 Cache), або кеш першого рівня, вбудований у процесори, починаючи з i486. Ємність такого кеша порівняно з основною оперативною пам'яттю невелика і становить 8-64 Кбайт, швидкодія – 10-12 нс.

Таблиця 3.2

Динамічні параметри кешів різних рівнів

Тип процесора	Pentium III	Athlon	Pentium 4	Athlon 64 X2	Core2 Duo/Quad
Швидкодія кеша першого рівня, нс (МГц)	0,71(1400)	0,71(1400)	0,26(3800)	0,33(3000)	0,34(2930)
Об'єм кеша першого рівня, (МБ)	32	128	20	256	64/128
Швидкодія кеша другого рівня, нс (МГц)	0,71(1400)	0,71(1400)	0,26(3800)	0,33(3000)	0,34(2930)
Об'єм кеша другого рівня, (МБ)	512	512	2048	2048	4096/8192
Швидкодія шини пам'яті, нс (МГц)	7,5(133)	3,8(266)	1,25(800)	2,5(400)	0,94(1066)

Він може бути побудований з використанням двох архітектур: *принстонської* та *гарвардської*. *Принстонська архітектура* передбачає використання спільної пам'яті L1 для зберігання даних і команд. За *гарвардської архітектури* пам'ять L1 розподіляється на дві рівні частини, одна з яких використовується для зберігання команд, інша – для даних. Кеш L1 безпосередньо вбудований в мікросхему ядра процесора.

Вторинний (зовнішній) кеш (L2 Cache), або кеш другого рівня, встановлюється на системній платі, а в процесорах P6 – на картриджі в одній упаковці з ядром і підключається до спеціальної внутрішньої шини процесора. Виготовляється у вигляді окремої мікросхеми. Якщо він встановлюється на системній платі, то працює на її частоті й знаходиться поруч з мікросхемою процесора.

Кеш третього рівня (L3 Cache) застосовується не часто, тому що його реалізація багато коштує (ємність повинна бути більша, ніж у вторинного). Вбудований кеш L3 є у процесорів Xeon MP, його розмір досягає 8 Мбайт.

Спочатку кеші проектувались як асинхронні і не були синхронізовані із системною шиною. Згодом, у 1995 році, було розроблено кеш синхронного типу, який працює синхронно із системною шиною процесора. Це підвищує його швидкодію та ефективність. Крім того, в цей же час функціонування процесорів було доповнено конвеєрним монопольним режимом (Pipeline Burst mode), що дало можливість скоротити період очікування за рахунок зменшення кількості станів очікування після першої передачі даних. Ці режими дали змогу підвищити ефективність комп'ютерів приблизно на 20%.

Ефективність і можливості кеша залежать від його контролера. Більшість контролерів мають обмеження на ємність кешованої пам'яті. Для комплекту

мікросхем системної логіки 430TX, який застосовується в комп'ютерах на основі процесора Pentium, вона становить 64 Мбайт. У такому випадку кешуються дані тільки в межах перших 64 Мбайт основної оперативної пам'яті, а всі дані після перших 64 Мбайт ніколи не потрапляють в кеш, і при зверненні до них завжди необхідні всі стани очікування, що визначаються повільнішою логікою DRAM [23].

До значного уповільнення роботи комп'ютера загалом можуть призвести тип програмного забезпечення та адреси, за якими зберігаються дані оперативної пам'яті. Так, наприклад, операційні системи Windows 95/98 і NT завантажуються в оперативну пам'ять; і якщо встановлено оперативну пам'ять ємністю 96 Мбайт, то операційна система і прикладні програми завантажуватимуться безпосередньо у верхні 32 Мбайт, які не кешуються. В цьому випадку можна зменшити ємність оперативної пам'яті до 64 Мбайт, тобто немає необхідності встановлювати ємність оперативної пам'яті більшу, ніж дає змогу її кешувати комплект мікросхем системної логіки.

Контролер кеша повинен забезпечувати когерентність (coherency) – узгодженість даних кеш-пам'яті обох рівнів з даними основної пам'яті при тому, що звернення до цих даних може здійснюватись не тільки процесором, а й іншими активними пристроями (busmaster), під'єднаними до системних шин PCI, VLB, ISA та ін. У складі обчислювальної системи (мережі) може бути кілька процесорів зі своїм внутрішнім кешем, що слід враховувати.

Контролер кеша сучасних процесорів розміщений або в мікросхемі North Bridge комплекту мікросхем системної логіки на основі процесора Pentium, або на платі процесорів Pentium Pro, Pentium II, Pentium III і Pentium IV.

У процесі обчислень ЦП може не тільки зчитувати існуючу інформацію, але і записувати нову, що приводить до оновлення вмісту кеш-пам'яті.

Поведінка кеш-контролера під час здійснення операції запису в пам'ять, коли копія ділянки, що вимагається, знаходиться в певному рядку кеша, визначається його алгоритмом роботи чи стратегією запису (Write Policy).

Існують *дві основні стратегії запису даних* з кеша в основну пам'ять: *наскрізний запис* – WT (*Write Through*) і *зворотний запис* – WB (*Write Back*).

WT – запис передбачає виконання кожної операції запису, яка потрапляє в кешований блок, одночасно в рядок кеша і в основну пам'ять. При цьому процесор вимушений щоразу здійснювати тривалу операцію запису в основну пам'ять. За такого запису достатньо тільки інформації тега. Але така простота запису не дає високої ефективності. Існують варіанти алгоритму WT із застосуванням *відкладеного буферизованого запису*, за якого дані в основну пам'ять переписуються через FIFO – буфер (First-In First-Out – "першим прийшов – першим і вийшов") під час вільних тактів шини. Під час використання буферизації процесор повністю звільняється від роботи з основною пам'яттю.

WB – запис дає змогу зменшити кількість операцій запису на системній шині основної пам'яті. Коли блок основної пам'яті, в який повинен здійснюватися запис, відображений в кеші, то фізичний запис спочатку відбуватиметься в цей дійсний рядок кеша і буде позначений як нечистий (dirty) або модифікований, тобто такий, що вимагає вивантаження в основну пам'ять. Тільки після цього вивантаження рядок стане чистим (clean), і його можна буде використати для кешування інших блоків без втрати цілісності даних. В основну пам'ять дані переписуються тільки цілим рядком. WB-алгоритм складніший в реалізації, ніж WT-алгоритм, але істотно ефективнішій. Підтримка кешування із зворотним записом потребує додаткових інтерфейсних сигналів, якщо здійснюється звернення з боку інших контролерів системної шини.

3.8.2. Основні архітектури кеш-пам'яті

Залежно від способу визначення взаємної відповідальності рядка кеша і ділянки основної пам'яті розрізняють три архітектури кеш-пам'яті: *кеш прямого відображення* (direct-mapped cache), *повністю асоціативний кеш* (fully associative cache) та їх комбінація – *набірно-асоціативний кеш* (set-associative cache), який має дві модифікації – множинно-асоціативне відображення і відображення секторів.

Кеш прямого відображення. Адреса пам'яті, за якою відбувається звернення до кеша прямого відображення, однозначно визначає рядок кеша, де може знаходитись необхідний блок.

Сутність архітектури прямого відображення полягає в тому, що кожен рядок кеша може відображати з будь-якої сторінки кешованої пам'яті тільки відповідний йому рядок. При цьому на кожний рядок кеша може претендувати багато сторінок пам'яті з однаковою молодшою частиною адреси, що є зміщенням всередині сторінки. Один рядок у певний момент містить копію тільки однієї з цих сторінок. Адресу (номер) рядка в кеш-пам'яті називають *індексом* (index).

Інформацію про те, яку саме сторінку основної пам'яті займає даний рядок, тобто старшу частину адреси чи номер сторінки, містить тег. Пам'ять тегів має кількість комірок, яка дорівнює кількості рядків кеша, а її розрядність повинна бути достатньою, щоб вмістити старші біти адреси кешованої пам'яті, які не потрапили на шину адреси кеш-пам'яті. Крім адресної частини тега кожний рядок кеша відображає біти ознак дійсності та модифікування даних. Організація кеш-пам'яті з прямим відображенням показана на рис. 3.20.

На цьому рисунку показана основна пам'ять з 14-розрядною адресою блоків, тобто адреса блоків може змінюватись від 0 до 16383. Логіка кеш-пам'яті інтерпретує ці 14 біт як 7-розрядний тег і 7-розрядне поле рядка.

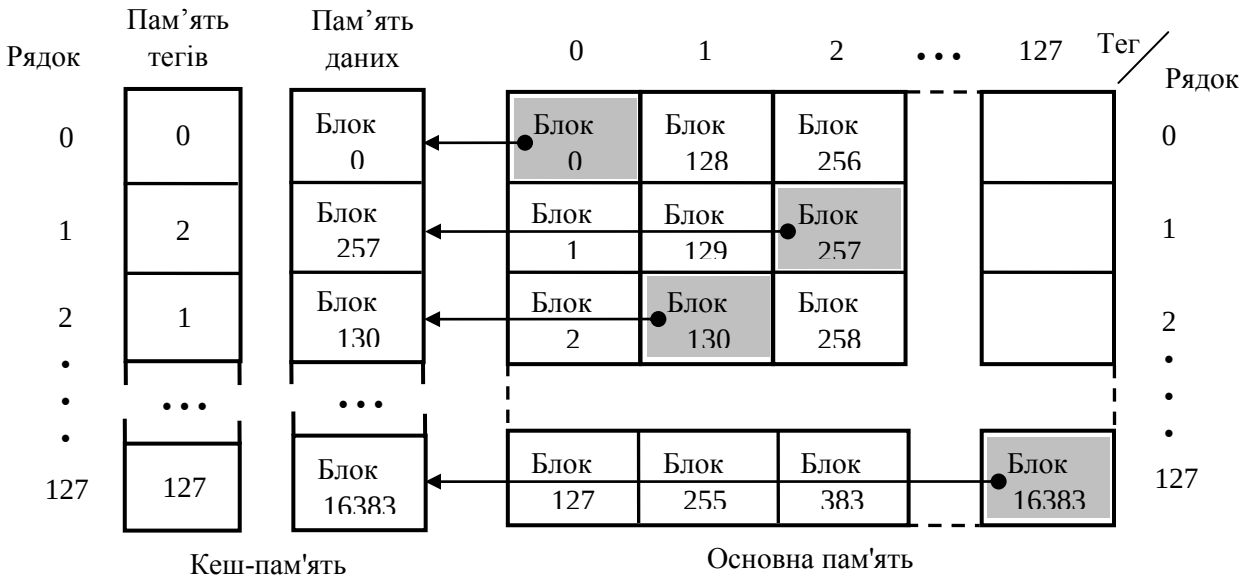


Рис. 3.20. Організація кеш-пам'яті з прямим відображенням

Поле рядка вказує на один із 128 рядків кеш-пам'яті, а саме на той, де може бути відображеним блок з заданою адресою. У свою чергу поле тега визначає, який саме зі списку блоків, що закріплені за даним рядком кеша, буде відображеним. Коли блок фактично заноситься в пам'ять даних кеша, в пам'ять тегів кеш-пам'яті необхідно записати тег цього блока. Тегом служать сім старших розрядів адреси блока.

На початку кожного звернення до кеш-пам'яті контролер спочатку зчитує комірку каталогу із даним індексом, порівнює біти тега зі старшими бітами адреси пам'яті та аналізує ознаку дійсності. Такий аналіз виконується в спеціальному *циклі стеження* (snoop cycle), який ще називають *циклом запиту* (inguire). Якщо в результаті аналізу з'ясується, що потрібний блок не знаходиться в кеші, то генерується або продовжується цикл звернення до основної пам'яті (випадок кеш-промаху). Коли ж є кеш-попадання, то запит обслуговується кеш-пам'яттю. У випадку кеш-промаху після зчитування основної пам'яті нові дані розміщуються в рядку кеша за умови, що він чистий, а в його тегу розташовуються старші біти адреси і встановлюється ознака дійсності даних. З основної пам'яті рядок переписується в кеш повністю, незалежно від обсягу даних, що вимагаються, оскільки ознака дійсності належить до всіх його байтів. Якщо контролер реалізує випереджаюче зчитування (read ahead), то в наступні вільні цикли шини поновиться і наступний рядок, якщо він був чистим. Читання “із запасом” дає змогу за необхідності здійснювати пакетний цикл зчитування з кеша через межу рядка.

Кеш розглянутого типу використовується у вторинному кеші більшості системних плат. Недоліком такої організації кеш-пам'яті є робота “вхолосту” (cache trashing), коли в процесі виконання програми процесора по черзі будуть потрібні блоки (сторінки) пам'яті, зміщені один стосовно іншого на величину,

кратну розміру сторінки. Чергове звернення заміщуватиме дані, зчитані у попередньому зверненні, які будуть необхідні в наступному. Таким чином, це буде суцільна низка кеш-промахів. Зменшує кількість кеш-попадань також переключення сторінок у багатозадачних обчислювальних системах. Оскільки різні задачі претендуватимуть на одні й ті самі рядки кеша, то збільшення його розмірів за архітектури прямого відображення не дає суттєвого підвищення ефективності. Підвищити її, не збільшуючи ємність кеша, можна тільки зміною структури кеш-пам'яті.

Повністю асоціативний кеш. У повністю асоціативному кеші, на відміну від попередньої архітектури, будь-який його рядок може відображати будь-який блок пам’яті (рис. 3.21). Це істотно підвищує ефективність використання його обмеженої ємності.

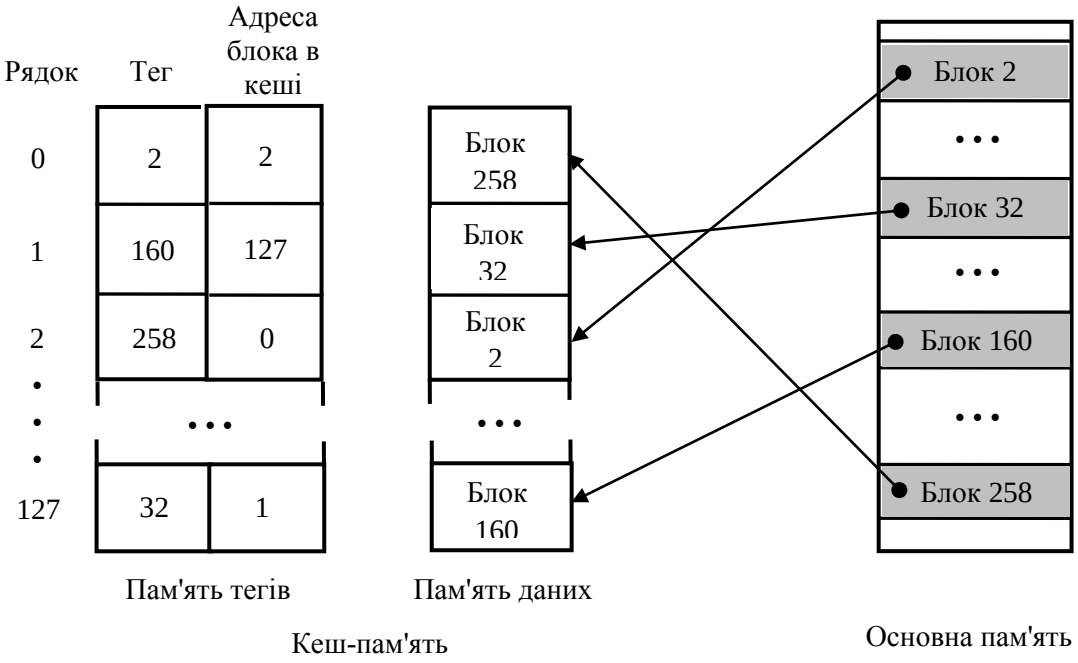


Рис. 3.21. Кеш-пам'ять з асоціативним відображенням

Такий кеш містить два банки: пам'ять даних і пам'ять тегів. Всі біти адреси кешованого блока, за винятком бітів, які визначають розташування даних у рядку (зміщення), зберігаються в пам'яті тегів. Повна адреса ОП для повністю асоціативної пам'яті ділиться на дві частини. Старші розряди є тегом і визначають номер блока даних в ОП, а молодші – зміщення в рядку кеша.

Довжина блока ОП, як і раніше, дорівнює довжині рядка кеш-пам'яті. Як пам'ять тегів у даній архітектурі використовується асоціативна пам'ять. При цьому кількість комірок асоціативної пам'яті тегів дорівнює кількості рядків кеша, тобто кожному рядку кеша віповідає однойменна комірка тега.

Під час запису блока даних з ОП в рядок кеша одночасно в тег записується старша частина адреси (номер рядка). При цьому для визначення наявності даних кеш-пам'яті, які вимагаються, необхідне порівняння тегів

усіх рядків зі старшою частиною адреси, а не одного або кількох, як у разі прямого відображення або набірно-асоціативній архітектурі. Таким чином, відпадає необхідність послідовного перебирання комірок пам'яті тегів. Виконується тільки паралельний аналіз усіх комірок.

Читання слова з кеш-пам'яті починається з асоціативного пошуку тега адресованого слова в асоціативній пам'яті тегів по коду, який є старшою частиною адреси ОП. Якщо під час асоціативного пошуку знаходиться тег адресованого слова, то воно зчитується з рядка кеша, номер якого визначається номером комірки тега, що збігся, а положення в рядку – зміщенням. Якщо результат асоціативного пошуку негативний, то адресованого слова немає в кеш-пам'яті і його необхідно зчитувати з ОП. При цьому разом з передачею адресованого слова в процесор в кеш-пам'ять переписується весь блок, в якому знаходиться це слово.

Недоліком цієї архітектури кеш-пам'яті є велика апаратурна складність пошуку інформації, зв'язана з необхідністю виконувати порівняння усіх тегів паралельно по всіх комірках асоціативної пам'яті. З підвищенням ємності кеш-пам'яті, відповідно і асоціативної пам'яті тегів, зростає її складність і вартість. Тому така архітектура використовується тільки для побудови кеш-пам'яті невеликої ємності (первинного кеша в деяких процесорах).

Набірно-асоціативний кеш. Набірно-асоціативна архітектура кеша дає можливість кожній сторінці основної кешованої пам'яті претендувати на один з кількох рядків кеша, об'єднаних в набір (set). В кеші такої архітектури є кілька паралельно і погоджено працюючих каналів прямого відображення, де контролер кеша приймає рішення про те, в який з рядків набору помістити черговий блок даних. Розглянемо організацію кеш-пам'яті з множинно-асоціативним відображенням.

Множинно-асоціативне відображення відноситься до групи методів частково-асоціативного відображення. Воно є одним з можливих компромісів, що поєднує переваги прямого і асоціативного способів відображення і, певною мірою, вільним від їх недоліків. Кеш-пам'ять (як тегів, так і даних) розбивається на v підмножин (надалі називатимемо такі підмножини модулями), кожна з яких містить k рядків (прийнято говорити, що модуль має k входів). Залежність між модулем і блоками ОП така ж, як і при прямому відображенні: на рядки, що входять в модуль i, можуть бути відображені тільки цілком певні блоки основної пам'яті, відповідно до співвідношення $i = j \bmod v$, де j - адреса блока ОП. У той же час розміщення блоків по рядках модуля – довільне, і для пошуку потрібного рядка в межах модуля використовується асоціативний принцип.

На рис. 3.22 показаний приклад чотиривходової кеш-пам'ятіз множинно-асоціативним відображенням. Пам'ять даних кеш-пам'яті розбита на 32 модулі по 4 рядки в кожному. Пам'ять тегів містить 32 комірки, в кожній з яких може зберігатися 4 значення тегів (поодинці на кожен рядок модуля). 14-розрядна

адреса блока ОП подається у вигляді двох полів: 9-розрядного поля тега і 5-розрядного поля номера модуля.

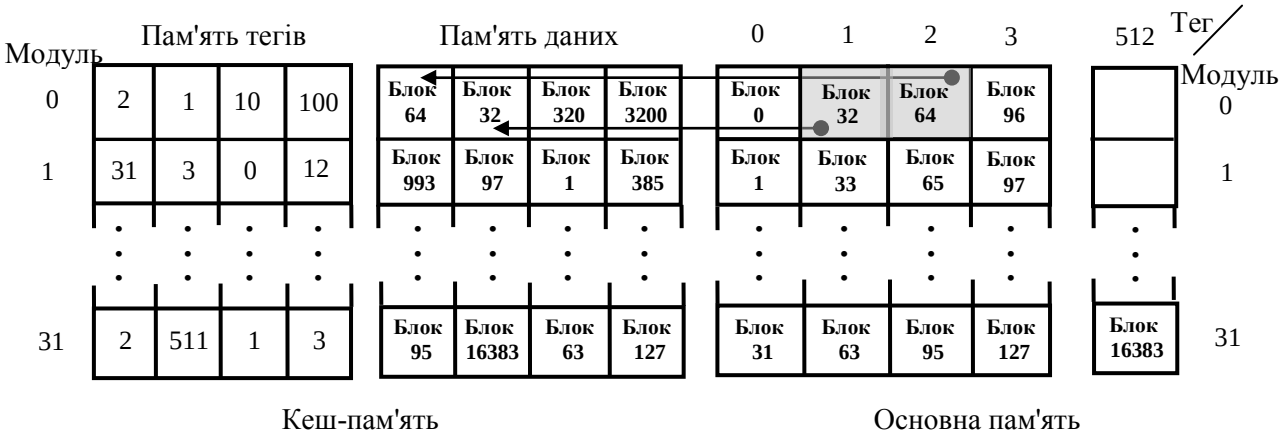


Рис. 3.22. Кеш-пам'ять з множинно-асоціативним відображенням

Номер модуля однозначно указує на один з модулів кеш-пам'яті. Він також дозволяє визначити номери тих блоків ОП, які можна відображати на цей модуль. Такими є блоки ОП, номери яких під час ділення на 2^5 дають у залишку число, що збігається з номером даного модуля кеш-пам'яті. Так, блоки 0, 32, 64, 96 і так далі в основній пам'яті відображаються на модуль з номером 0; блоки 1, 33, 65, 97 і так далі відображаються на модуль 1 і т. д. Будь-який з блоків у послідовності може бути завантажений у будь-який з чотирьох рядків відповідного модуля.

У випадку такої постановки роль тега виконують 9 старших розрядів адреси блока ОП, в яких міститься порядковий номер блока в послідовності блоків, що відображаються на один і той же модуль кеш-пам'яті. Наприклад, блок 65 у послідовності блоків, що відображаються на модуль 1, має порядковий номер 2 (відлік ведеться від 0).

Під час звернення до кеш-пам'яті 5-розрядний номер модуля указує на конкретну комірку пам'яті тегів (це відповідає прямому відображенню). Далі проводиться паралельне порівняння кожного з чотирьох тегів, що зберігаються в цій комірці, з полем тега адреси, що поступила, тобто пошук потрібного тега серед чотирьох можливих здійснюється асоціативно.

У граничних випадках, коли ***v=m, k=1,*** множинно-асоціативне відображення зводиться до прямого, а при ***v=1, к=m*** − до асоціативного.

Найбільш загальний вид організації множинно-асоціативного відображення – використання двох рядків на модуль **(*v=m/2, k=2*).** Чотиривходова множинно-асоціативна кеш-пам'ять **(*v=m/4, k=4*)** дає додаткове поліпшення за порівняно невелику додаткову ціну [25]. Подальше збільшення числа рядків у модулі істотного ефекту не привносить.

Слід зазначити, що саме цей спосіб відображення найширше розповсюджений у сучасних мікропроцесорах.

3.8.3. Структура засобів кешування пам'яті

Засоби кешування пам'яті містять два рівні кеш-інструкцій і даних (L1 Cache і L2 Cache), буфери асоціативної трансляції TLB блока сторінкової переадресації і буфери запису. Вони можуть бути подані в різних варіаціях, зокрема розміщені на кристалі або картриджі процесора, або на системній платі, починаючи з процесора i80486. Процесор i80386 містить тільки буфери TLB. Кеш-пам'ять, що встановлювалась на системній платі, не підтримувалась процесором.

Загальну структуру засобів кешування пам'яті для 32-розрядних процесорів фірми Intel, у тому числі для процесорів P6, наведено на рис. 3.23.

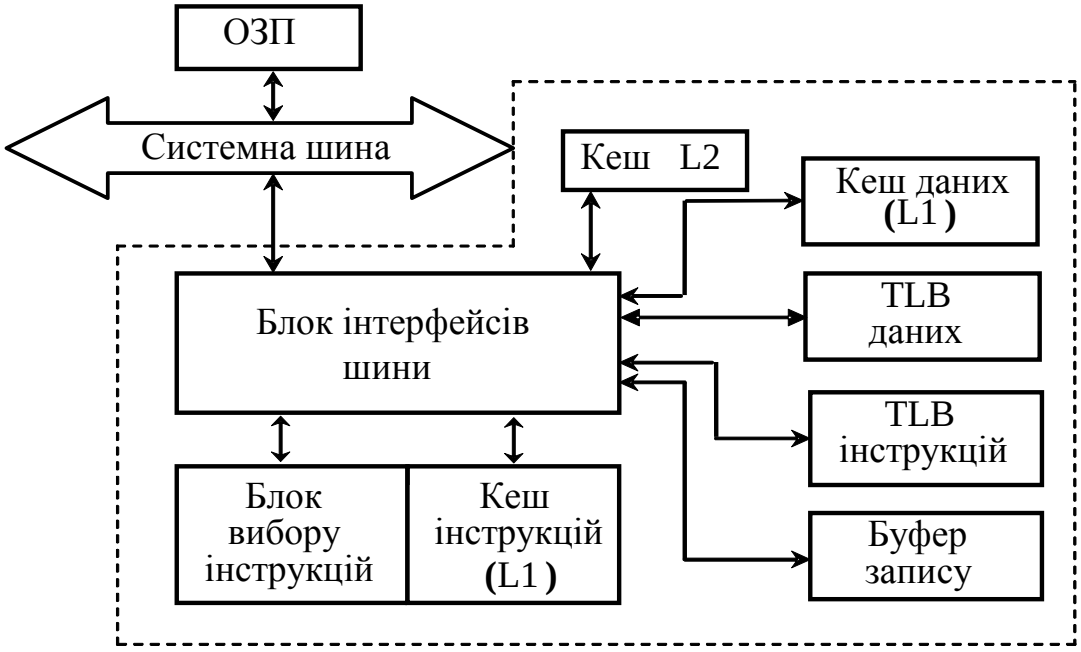


Рис. 3.23. Загальна структура засобів кешування пам'яті

Первинний кеш інструкцій тісно пов'язаний з блоком попередньої вибірки, а первинний кеш даних - з виконавчим блоком процесора. Вторинний кеш є спільним і в процесорах P6 підключений до окремої внутрішньої шини кеш-пам'яті. В процесорах i486 і Pentium вторинний кеш є зовнішнім і підключається до зовнішньої системної шини процесора. В процесорах Celeron 266 і 300 вторинний кеш відсутній.

Довжина рядка кеша в i486 – 16 байт, в процесорах P5 другого покоління і P6 – 32 байт. Рядки заповнюються цілком пакетними циклами зчитування – 4 передачі на рядок з основної пам'яті, вирівняними за 32-байтними межами.

Будь-який внутрішній запит процесора на звернення до пам'яті спрямовується у внутрішній кеш.

Якщо запитувана ділянка пам'яті наявна у рядку внутрішнього кеша, то він обслуговує цей запит. У випадку промаху запит задовольняється, як тільки необхідні дані зчитуються з ОЗП. Заповнення рядка до кінця відбувається паралельно з обробкою одержаних даних. Виділення і заміщення у процесорах i486 і P5 виконуються тільки для кеш-промахів під час зчитування. У випадку промахів запису заповнення рядків здійснюється тільки в процесорах P6. Кешування доступне в будь-якому режимі процесора.

Буфер асоціативної трансляції TLB зберігає входження в каталог і в таблиці сторінок, до яких звертались останнім часом. В i486 для даних та інструкцій використовується єдиний TLB, у процесорах P5 і P6 ці буфери роздільні.

Великі сторінки, зокрема 2 Мбайт у режимі PAE (Physical Address Extension – режим розширення фізичної адреси до 36 біт) і 4 Мбайт в PSE (Page Size Extension – прапорець розширення розміру сторінки) обслуговуються роздільними TLB.

Буфери запису пов'язані з виконавчим блоком процесора. Вони дають змогу на деякий час відкласти фактичний запис у зовнішній кеш і основну пам'ять, пропонуючи шину для інших обмінів, необхідних для виконання наступних інструкцій. Запис буферизується в усіх режимах роботи процесора, але буферизація запису в порти вводу/ виводу не відбувається.

Кеш-пам'ять побудована з урахуванням можливості звернень до неї зовнішніх об'єктів, зокрема інших процесорів та контролерів. Процесори мають механізми зовнішнього стеження за станом свого кеша. Для підтримки узгодження даних кеша та основної пам'яті процесор відпрацьовує цикли стеження (Snoop Cycle чи Inquire Cycle), які ініціюються зовнішньою для нього системою. В цих циклах, які відбуваються у разі звернення до пам'яті з боку зовнішнього абонента, процесор визначає наявність даних, що вимагаються, в своєму кеші. Якщо вони відображаються в кеші, то дії процесора залежать від стану відповідного рядка кеша і типу зовнішнього звернення. Звернення під час запису призведе до анулювання даного рядка. Звернення під час зчитування до ділянки, яка відповідає модифікованому (“брудному”) рядку, призведе до вивантаження його вмісту в основну пам'ять перед тим як зовнішній абонент виконає реальне зчитування. В процесорах P6 звернення до “брудного” рядка з боку інших процесорів може спричинити вивантаження його вмісту безпосередньо в процесор, що звертався. Це відповідно збереже час, а вивантаження цього рядка в основну пам'ять відбудеться пізніше, згідно з алгоритмом оберненого запису.

Починаючи з процесорів Pentium, їх кеш підтримує протокол MESI (Modified-Exclusive-Shared-Invalid – протокол підтримки когерентності пам'яті за наявністю кеша, названий за визначеним станом рядків: Модифікована -

Виняткова – Роздільна – Недійсна). Первинний кеш інструкцій реалізує лише частину протоколу – SI, оскільки не допускає запису.

У просторі основної пам'яті комп'ютера є ділянки, для яких кешування принципово недопустиме, зокрема розподільна пам'ять адаптерів. Для таких ділянок непридатний алгоритм оберненого зв'язку. Крім того, кешування інколи відключають у разі виконання однократно виконуваних ділянок програми з тим, щоб з кеша не витісняти корисніші фрагменти програми.

3.9. ПОНЯТТЯ ВІРТУАЛЬНОЇ ПАМ'ЯТІ

Для більшості типових застосувань ОМ характерна ситуація, коли розміщення всієї програми в ОП неможливо через її великий розмір. У цьому, проте, і немає принципової необхідності, оскільки в кожен момент часу «увага» машини концентрується на визначених порівняно невеликих ділянках програми. Таким чином, в ОП досить зберігати тільки використовувані в даний період частини програм, а решта частин може розташовуватися на зовнішніх ЗП (ЗЗП). Складність подібного підходу в тому, що процеси звернення до ОП і ЗЗП істотно розрізняються, і це ускладнює завдання програміста. Виходом з такої ситуації була поява в 1959 році ідеї *віртуалізації пам'яті* [25], під якою розуміється метод автоматичного управління ієрархічною пам'яттю, при якому програмістові здається, що він має справу з єдиною пам'яттю великої ємкості і високої швидкодії. Цю пам'ять називають *віртуальною* (що здається) *пам'яттю*. За своєю суттю віртуалізація пам'яті є способом апаратної і програмної реалізації концепції ієрархічної пам'яті.

В рамках ідеї віртуалізації пам'яті ОП розглядається як лінійний простір N адрес, названий *фізичним простором* пам'яті. Для завдань, де потрібно більш ніж N комірок, надається значно більший простір адрес (зазвичай рівний загальній ємності всіх видів пам'яті), названий *віртуальним простором*, у загальному випадку не обов'язково лінійний. Адреси віртуального простору називають *віртуальними*, а адреси фізичного простору – *фізичними* (рис. 3.24).

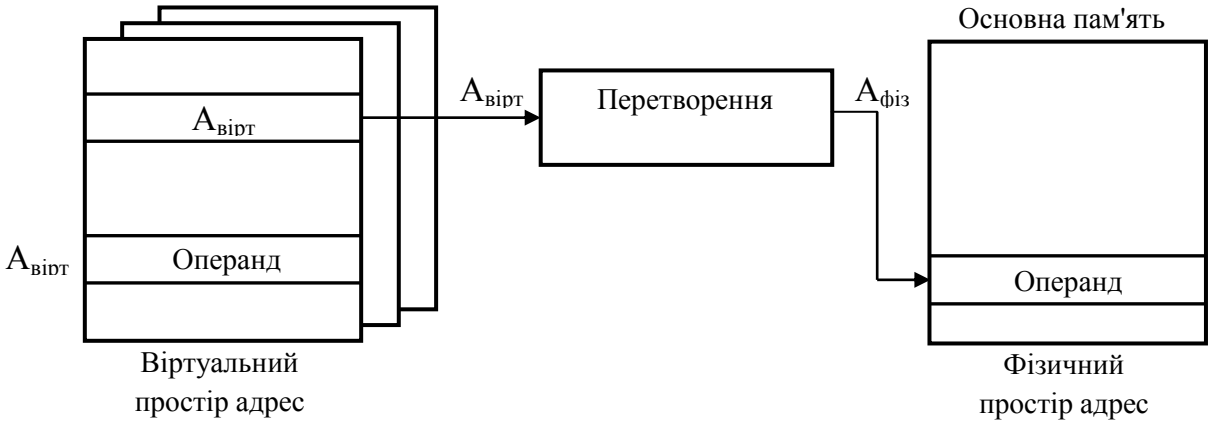


Рис. 3.24. Відображення віртуальної адреси на фізичну

Програма пишеться у віртуальних адресах, але оскільки для її виконання потрібно, щоб оброблювані команди і дані знаходилися в ОП, потрібно, щоб кожній віртуальній адресі відповідала фізична. Таким чином, у процесі обчислень необхідно, перш за все, переписати з ЗЗП в ОП ту частину інформації, на яку вказує віртуальна адреса (відобразити віртуальний простір на фізичний), після чого перетворити віртуальну адресу у фізичну(рис. 3.24).

Серед систем віртуальної пам'яті можна виділити два класи: системи з фіксованим розміром блоків (сторінкова організація) і системи із змінним розміром блоків (сегментна організація). Обидва варіанти зазвичай суміщають (сегментно-сторінкова організація).

3.9.1. Сторінкова організація пам'яті

Цілям перетворення віртуальних адрес у фізичні служить сторінкова організація пам'яті. Її ідея полягає в розбитті програми на частини рівної величини, що називаються *сторінками*. Розмір сторінки зазвичай вибирають в межах 4-8 Кбайт, але так, щоб він був кратний ємності одного сектора магнітного диска. Віртуальний і фізичний адресні простори розбиваються на блоки розміром у сторінку. Блок основної пам'яті, відповідний сторінці, часто називають *сторінковим кадром* або *фреймом* (page frame). Сторінкам віртуальної і фізичної пам'яті привласнюють номери. Процес доступу до даних за їх віртуальною адресою ілюструє рис. 3.25.

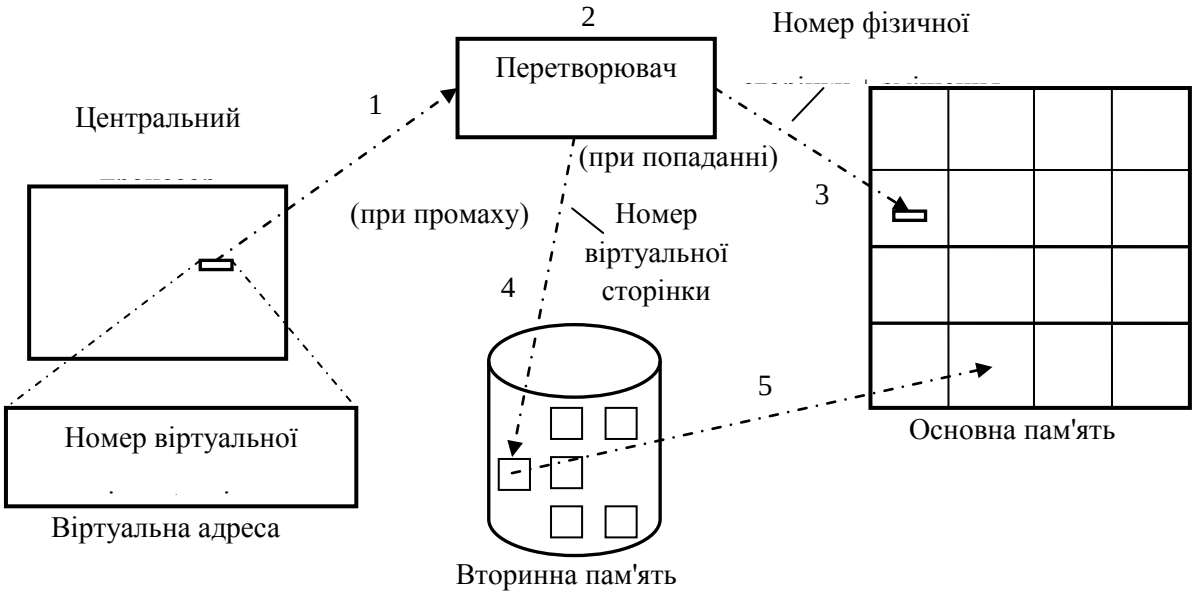


Рис. 3.25. Сторінкова організація віртуальної пам'яті

Центральний процесор звертається до комірки, вказавши її віртуальну адресу (1), яка складається з номера віртуальної сторінки і зсуву щодо її початку. Ця адреса поступає в систему перетворення адрес (2), з метою отримання з неї фізичної адреси комірки в основній пам'яті (3). Оскільки зсув у віртуальній і фізичній адресі однаковий, перетворенню піддається лише номер

сторінки. Якщо перетворювач виявляє, що потрібна фізична сторінка відсутня в основній пам'яті (відбувся промах або сторінковий збій), то потрібна сторінка зчитується із зовнішньої пам'яті і заноситься в ОП (4, 5).

Перетворювач адрес – це частина операційної системи, що транслює номер віртуальної сторінки в номер фізичної сторінки, розташованої в основній пам'яті, а також апаратура, що забезпечує цей процес і що дозволяє прискорити його. Перетворення здійснюється за допомогою так званої *сторінкової таблиці*. За відсутності потрібної сторінки в ОП перетворювач адрес виробляє ознаку сторінкового збою, по якому операційна система припиняє обчислення, поки потрібна сторінка не буде зчитана з вторинної пам'яті і поміщена в основну.

Віртуальний простір повністю описується двома таблицями (рис. 3.26): сторінковою таблицею і картою диска (вважатимемо, що вторинна пам'ять реалізована на магнітних дисках). Таблиця сторінок визначає, які віртуальні сторінки знаходяться в основній пам'яті і в яких фізичних фреймах, а карта диска містить інформацію про сектори диска, де зберігаються віртуальні сторінки на диску.

Число записів у сторінковій таблиці (СТ) в загальному випадку рівне кількості віртуальних сторінок. Кожен запис містить поле номера фізичної сторінки (НФС) і чотири ознаки: **V**, **R**, **M** і **A**.

Ознака присутності **V** встановлюється в одиницю, якщо віртуальна сторінка в даний момент знаходиться в основній пам'яті. В цьому випадку в полі номера фізичної сторінки знаходиться відповідний номер. Якщо **V** = 0, то у разі спроби звернутися до даної віртуальної сторінки перетворювач адреси генерує сигнал сторінкового збою (page fault), і операційна система робить дії із завантаження сторінки з диска в ОП, звертаючись для цього до карти диска. В карті вказано, на якій доріжці і в якому секторі диска розташована кожна з віртуальних сторінок. Завантаження сторінки з диска в ОП супроводжується записом у відповідний рядок сторінкової таблиці (указується номер фізичної сторінки, куди була завантажена віртуальна сторінка).

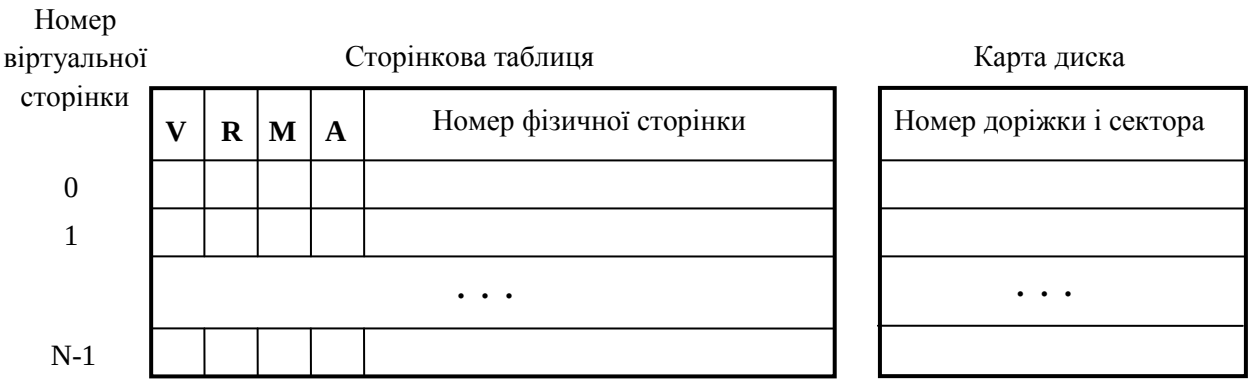


Рис. 3.26. Структура сторінкової таблиці і карти диска

У принципі карта диска може бути суміщена із сторінковою таблицею шляхом додавання до останньої ще одного поля. Іншим варіантом може бути збільшення розрядності поля номера фізичної сторінки і зберігання в нім номерів доріжок і секторів для віртуальних сторінок, відсутніх в основній пам'яті. В цьому випадку вид інформації, що зберігається, визначатиме ознаку **V**.

Ознака використання сторінки **R** встановлюється у разі звернення до даної сторінки. Ця інформація використовується в алгоритмі заміщення сторінок для вибору тієї з них, яку можна найбільш безболісно видалити з ОП, щоб звільнити місце для нової. Проблеми заміщення інформації в ОП вирішуються так само, як і для кеш-пам'яті.

Оскільки в ОП знаходяться лише копії сторінок, а їх оригінали зберігаються на диску, необхідно забезпечити ідентичність оригіналів і копій. У ході обчислень вміст окремих сторінок може змінюватися, що фіксується шляхом установки в одиницю *ознаки модифікації* **M**. Під час видалення сторінки з ОП перевіряється стан ознаки **M**. Якщо **M** = 1, то перед видаленням сторінки з основної пам'яті її необхідно переписати на диск, а при **M** = 0 цього можна не робити.

Ознака прав доступу **A** служить цілям захисту інформації і визначає, який вид доступу до сторінки дозволений: тільки для читання, тільки для запису, для читання і для запису.

Коли програма завантажується в ОП, вона може бути направлена в будь-які вільні в даний момент сторінкові кадри, незалежно від того, чи розташовані вони підряд чи ні. Сторінкова організація дозволяє скоротити об'єм пересилок інформації між зовнішньою пам'яттю і ОП, оскільки сторінку не потрібно завантажувати до тих пір, поки вона дійсно не знадобиться.

Спосіб реалізації СТ життєво важливий для ефективності техніки віртуальної адресації, оскільки кожне звернення до пам'яті припускає звернення до сторінкової таблиці. Найбільш швидкий спосіб – зберігання таблиці в спеціально виділених для цього регістрах, але від нього доводиться відмовлятися у разі великого об'єму СТ. Залишається практично один варіант – виділення сторінкової таблиці області основної пам'яті, незважаючи на те, що це приводить до двократного збільшення часу доступу до інформації. Щоб скоротити цей час, до складу ОМ включають додатковий ЗП, що називається *буфером швидкого перетворення адреси* (TLB – Translation Look-aside Buffer), або *буфером асоціативної трансляції*, або *буфером випереджаючої вибірки* і який є кеш-пам'яттю. Під час кожного перетворення номера віртуальної сторінки в номер фізичної сторінки результат заноситься в TLB: номер фізичної сторінки в пам'ять даних, а віртуальної – в пам'ять тегів. Таким чином, у TLB потрапляють результати декількох останніх операцій трансляції адрес. Під час кожного звернення до ОП перетворювач адрес спочатку проводить пошук у

пам'яті тегів TLB номера необхідної віртуальної сторінки. У разі попадання адреса відповідної фізичної сторінки береться з пам'яті даних TLB. Якщо в TLB зафіксований промах, то процедура перетворення адрес проводиться за допомогою сторінкової таблиці, після чого здійснюється запис нової пари «номер віртуальної сторінки – номер фізичної сторінки» в TLB. Структура TLB показана на рис. 3.27.

Номер віртуальної сторінки	**V**	**R**	**M**	**A**	Номер фізичної сторінки
. . .	. . .				

Пам'ять тегів Пам'ять даних

Рис. 3.27. Структура буфера швидкого перетворення адрес

Буфер перетворення адрес зазвичай реалізується у вигляді повністю асоціативної або множинно-асоціативної кеш-пам'яті з високим ступенем асоціативності і часом доступу, що збігається з аналогічним показником для кеш-пам'яті першого рівня (L1). Число входів у типових TLB невелике (64-256). Так, TLB мікропроцесора Pentium III має 64 входи при розмірі сторінки 4 Кбайт, що дозволяє дістати швидкий доступ до адресного простору в 256 Кбайт.

Серйозною проблемою в системі віртуальної пам'яті є великий об'єм сторінкових таблиць, який пропорційний числу віртуальних сторінок. Таблиця займає значну частину ОП, а на пошук іде багато часу, що вкрай небажано.

Один із способів скорочення довжини таблиць заснований на введенні багаторівневої організації таблиць. У цьому варіанті інформація оформляється у вигляді декількох сторінкових таблиць порівняно невеликого об'єму, які утворюють другий рівень. Перший рівень представлений таблицею з каталогом, де вказано місцеположення кожної із сторінкових таблиць (адреса початку таблиці в пам'яті) другого рівня. Спочатку в каталозі визначається розташування потрібної сторінкової таблиці і лише потім проводиться звернення до потрібної таблиці. Про ефект, що досягається, можна судити з наступного прикладу. Хай адресна шина ОМ має ширину 32 біта, а розмір сторінки дорівнює 4 Кбайт. Тоді кількість віртуальних сторінок, а отже, і число входів у єдиній сторінковій таблиці складе 2^{20}. У разі дворівневої організації можна обійтися однією сторінкою першого рівня на 1024 (2^{10}) входів і 1024 сторінковими таблицями на таке ж число входів.

Інший підхід називають *способом обернених* або *інвертованих сторінкових таблиць*. Таку таблицю в якомусь сенсі можна розглядати як збільшений

еквівалент TLB, який відрізняється тим, що вона зберігається в ОП і реалізується не апаратурою, а програмними засобами. Число входів у таблицю визначається ємністю ОП і дорівнює числу сторінок, яке може бути розміщене в основній пам'яті. Одночасно з цим є і традиційна СТ, але зберігається вона не в ОП, а на диску. Для пошуку потрібного запису в інвертованій таблиці використовується хешування, коли номер запису в таблиці обчислюється відповідно до якоїсь хеш-функції. Аргументом цієї функції служить номер шуканої віртуальної сторінки. Хешування дозволяє прискорити операцію пошуку. Якщо потрібна сторінка в ОП відсутня, проводиться звернення до основної таблиці на диску і після завантаження сторінки в ОП коректується і інвертована таблиця.

3.9.2. Сегментно-сторінкова організація пам'яті

У разі сторінкової організації передбачається, що віртуальна пам'ять – це безперервний масив з наскрізною нумерацією слів, що не завжди можна визнати оптимальним. Зазвичай програма складається з декількох частин – кодової, інформаційної та стекової. Оскільки заздалегідь невідомі довжини цих складових, то зручно, щоб під час програмування кожна з них мала власну нумерацію слів, відлічуваних з нуля. Для цього організовують систему *сегментованої пам'яті*, виділяючи у віртуальному просторі незалежні лінійні простори змінної довжини, що називаються *сегментами*. Кожен сегмент є окремою логічною одиницею інформації, що містить сукупність даних або програмний код і розташована в адресному просторі користувача. В кожному сегменті встановлюється своя власна нумерація слів, починаючи з нуля. Віртуальна пам'ять також розбивається на сегменти, з незалежною адресацією слів усередині сегмента. Кожній складовій програми виділяється сегмент пам'яті. Віртуальна адреса визначається номером сегмента і адресою всередині сегмента. Для перетворення віртуальної адреси у фізичну використовується спеціальна *сегментна таблиця*.

Недоліком такого підходу є те, що неоднаковий розмір сегментів приводить до неефективного використання ОП. Так, якщо ОП заповнена, то у разі заміщення одного з сегментів потрібно витіснити такий, розмір якого дорівнює або більше розміру нового. У разі багатократного повтору подібних дій в ОП залишається багато вільних ділянок, недостатніх за розміром для завантаження повного сегмента. Вирішенням проблеми служить *сегментно-сторінкова організація пам'яті*. В ній розмір сегмента вибирається не довільно, а задається кратним розміру сторінки.

Структуру віртуальної адреси і процес перетворення її у фізичну адресу ілюструє рис. 3.28.

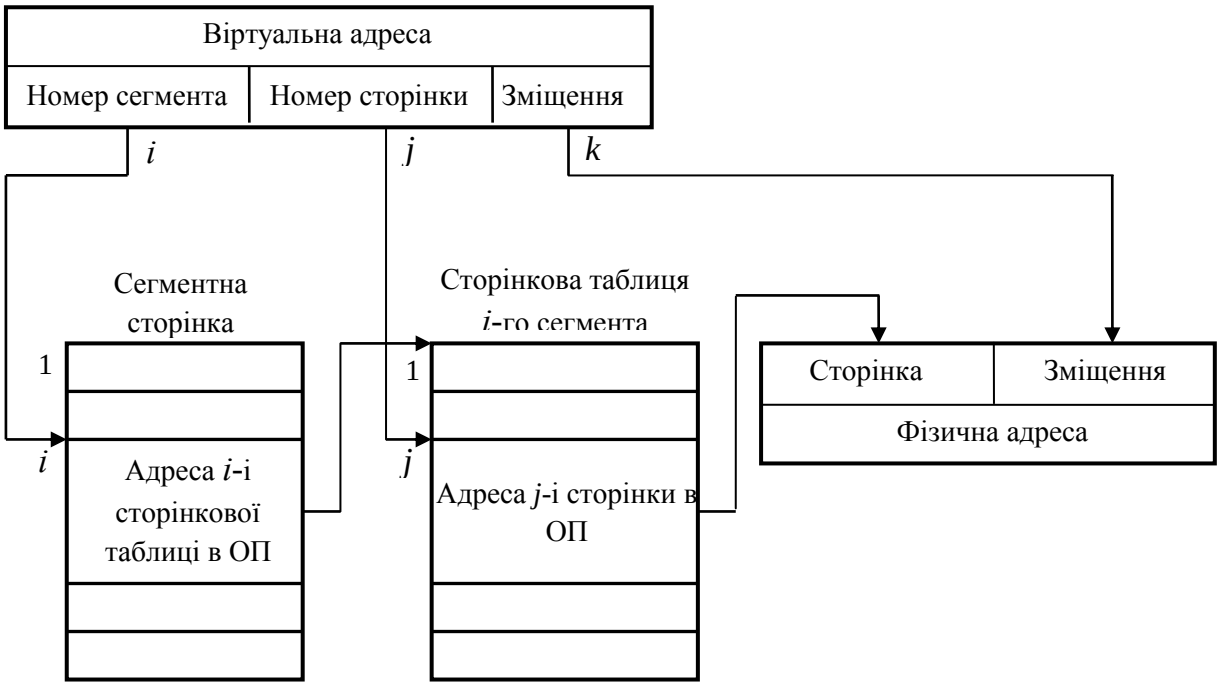


Рис. 3.28. Перетворення адреси при сегментно-сторінковій організації пам'яті

Сегмент може містити те або інше, але обов'язково ціле число сторінок, навіть якщо одна із сторінок заповнена частково. Виникає певна ієрархія в організації доступу до даних, що складається з трьох ступенів: сегмент > сторінка > слово. Цій структурі відповідає ієрархія таблиць, які служать для перекладу віртуальних адрес у фізичні. В сегментній таблиці програми перераховуються всі сегменти даної програми з вказівкою початкових адрес СТ, які відносяться до кожного сегмента. Кількість сторінкових таблиць дорівнює числу сегментів і будь-яка з них визначає розташування кожної із сторінок сегмента в пам'яті, які можуть розташовуватися не підряд – частина сторінок може знаходитися в ОП, останні – у зовнішній пам'яті

Для отримання фізичної адреси необхідний доступ до сегментної та однієї із сторінкових таблиць, тому перетворення адреси може займати багато часу.

3.10. ОРГАНІЗАЦІЯ ЗАХИСТУ ПАМ'ЯТІ

Сучасні обчислювальні машини, як правило, працюють у багатокористувацькому і багатозадачному режимах, коли в основній пам'яті одночасно знаходяться програми, що відносяться як до різних користувачів, так і до різних завдань одного користувача. Якщо навіть ОМ виконує тільки одну програму, в ОП, крім цієї програми і даних до неї, завжди присутні фрагменти операційної системи. Кожному завданню в основній пам'яті виділяється свій адресний простір. Такі простори, якщо тільки це спеціально не передбачено, зазвичай незалежні. В той же час у програмах можуть міститися помилки, які приводять

до вторгнення в адресний простір інших завдань. Наслідком цих помилок може стати спотворення інформації, що належить іншим програмам. Отже, в ОМ обов'язково повинні бути передбачені заходи, що запобігають несанкціонованій дії програм одного користувача на роботу програм інших користувачів і на операційну систему.

Щоб перешкодити руйнуванню одних програм іншими, досить захистити область пам'яті даної програми від спроб запису в неї з боку інших програм (захист від запису). В ряді випадків необхідно мати можливість захисту і від читання з боку інших програм, наприклад, у разі обмежень на доступ до системної інформації.

Захист від вторгнення програм у чужі адресні простори реалізується різними засобами і способами, але в будь-якому варіанті до системи захисту пред'являються дві вимоги: її реалізація не повинна помітно знижувати продуктивність ОМ і вимагати дуже великих апаратних витрат.

Задача зазвичай вирішується за рахунок поєднання програмних і апаратних засобів, хоч відповідальність за охорону адресних просторів від несанкціонованого доступу зазвичай покладається на операційну систему. Коротко розглянемо апаратні аспекти проблеми захисту пам'яті.

Кільця захисту. Захист адресного простору операційної системи від несанкціонованого вторгнення з боку призначених для користувача програм зазвичай організовують за рахунок апаратно реалізованого розділення системного і користувацького рівнів привілеїв. Передбачаються як мінімум два режими роботи процесора: системний (режим супервізора – «наглядача») і користувацький. Таку структуру прийнято називати *кільцями захисту* і зображати у вигляді концентричних кіл, де призначений для користувача режим поданий зовнішнім кільцем, а системний – внутрішнім колом. У системному режимі програмі доступні всі ресурси ОМ, а можливості призначеного для користувача режиму істотно обмежені. Перемикання з призначеного для користувача режиму в системний здійснюється спеціальною командою. В більшості сучасних ОМ число рівнів привілеїв (кілець захисту) збільшене. Так, у мікропроцесорах класу Pentium передбачено чотири рівні привілеїв.

Метод граничних регістрів. Даний вид захисту найбільш поширений. Метод припускає наявність у процесорі двох *граничних регістрів*, вміст яких визначає нижню і верхню межі області пам'яті, куди програма має право доступу. Заповнення граничних регістрів проводиться операційною системою під час завантаження програми. Під час кожного звернення до пам'яті перевіряється, чи потрапляє використовувана адреса у встановлені межі. Таку перевірку, наприклад, можна організувати на етапі перетворення віртуальної адреси у фізичну. У разі порушення межі доступ до пам'яті блокується і

формується запит переривання, що викликає відповідну процедуру операційної системи. Нижню межу дозволеної області пам'яті визначає сегментний регістр. Верхня межа підраховується операційною системою відповідно до розміру розміщуваного в ОП сегмента.

У розглянутій схемі необхідно, щоб в ОМ підтримувалися два режими роботи: привілейований і користувацький. Запис інформації в граничні регістри можливий лише в привілейованому режимі.

Метод ключів захисту. Метод дозволяє організувати захист несуміжних областей пам'яті. Пам'ять умовно ділиться на блоки однакового розміру. Кожному блоку ставиться у відповідність деякий код, названий *ключем захисту пам'яті*. Кожній програмі, у свою чергу, привласнюється *код захисту програми*. Умовою доступу програми до конкретного блока пам'яті служить збіг ключів захисту пам'яті і програми або рівність одного з цих ключів нулю. Нульове значення ключа захисту програми дозволяє доступ до всього адресного простору і використовується тільки програмами операційної системи. За розподіл ключів захисту програми відповідає операційна система. Ключ захисту програми зазвичай представлений у вигляді окремого поля *слова стану програми*, яке зберігається в спеціальному регістрі. Ключі захисту пам'яті зберігаються в спеціальній пам'яті. Під час кожного звернення до ОП спеціальна комбінаційна схема проводить порівняння ключів захисту пам'яті і програми. У разі збігу доступ до пам'яті дозволяється. Дії у разі незбігання ключів залежать від того, який вид доступу заборонений: під час запису, під час читання або в обох випадках. Якщо з'ясувалося, що даний вид доступу заборонений, то так само, як і в методі граничних регістрів, формується запит переривання і викликається відповідна процедура операційної системи.

3.11. ЗОВНІШНЯ ПАМ'ЯТЬ

Зовнішню пам’ять утворюють зовнішні (відносно МП та системної плати) пристрої комп’ютера, вона використовується для довготривалого зберігання інформації. Зокрема, в зовнішній пам’яті зберігається все програмне забезпечення комп’ютера. Вона містить різноманітні запам’ятовуючи пристрої. Найпоширеніші – накопичувачі на жорстких та гнучких магнітних дисках (НЖМД та НГМД), призначення яких – зберігання великих обсягів інформації, запис і видача інформації за запитом ОЗП. Як пристрої зовнішньої пам’яті застосовуються також запам’ятовуючі пристрої на касетній магнітній стрічці (стримери), накопичувачі на оптичних дисках, флеш-пам’ять.

Основа зовнішній пам’яті – електромеханічні запам’ятовуючи пристрої з рухомим носієм інформації, на поверхню якого наноситься магнітний матеріал із властивостями залишкової намагніченості.

Сучасні накопичувачі на жорстких магнітних дисках - (НЖМД) отримали назву “вінчестер” за асоціацією з рушницею калібру 30/30. Такий же самий формат мали перші накопичувачі цього типу – 30 доріжок на 30 секторів.

Габаритні розміри (форм-фактор) сучасних вінчестерів такі:

- горизонтальні — понад 1.8; 2.5; 3.5; 5.25 дюймів;
- вертикальні — Full Height 3.25";

Half-Height 1.63";

Low Profile 1 дюйм

(1 дюйм =2.54 см).

Механізм вінчестера поміщений у металевий корпус. Кришка вінчестера охороняє його від забруднення, ушкоджень, електромагнітних полів.

Усередині металевої коробочки обертається жорстко закріплений на осі шпиндельного двигуна диск або пакет, зібраний з декількох дисків. Кутова швидкість обертання шпиндельного двигуна сучасних дисководів така: 5400, 7200, 10000, 12000 обер./хв.

Повітря від надлишкового тиску стравлюється з корпусу через мініатюрний клапан, що дозволяє вирівняти внутрішній і зовнішній тиск. Внутрішні потоки повітря переганяються через фільтр.

У вінчестері є компоненти високоточної (прецизійної) механіки й електронні вузли (рис. 3.29). Елементи електроніки розташовуються головним чином на знімній платі електроніки. На платі розташовані: контролер інтерфейса, процесор (іноді кілька), ПЗП з програмою керування дисководом, буферна пам'ять, сервоголівка, логіка читання. Певна частина програми керування диском може бути записана на ньому самому.

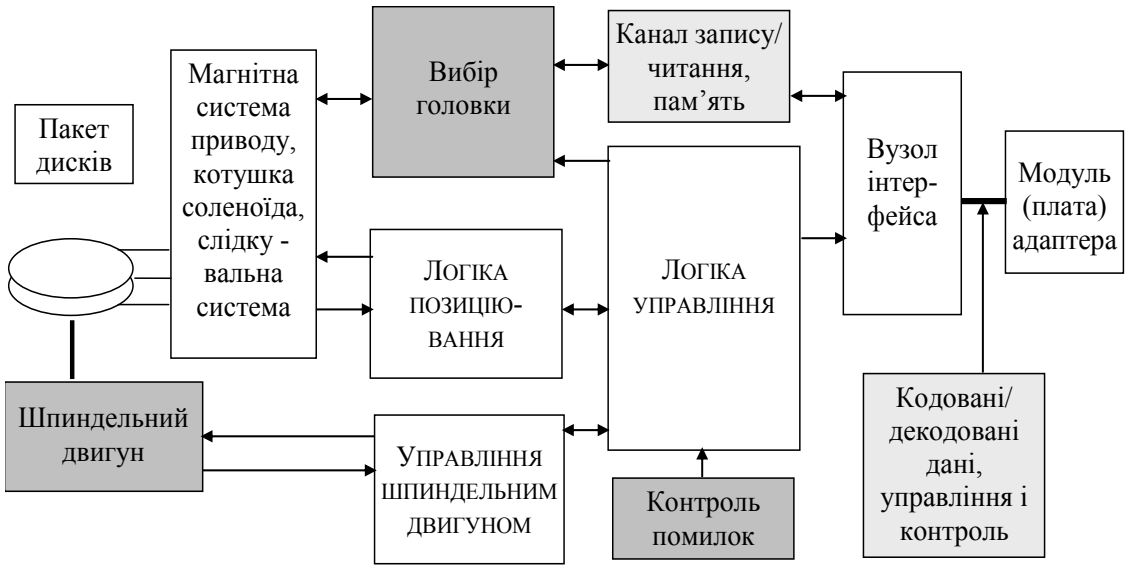


Рис. 3.29. Блок – схема НЖМД

На доріжках (треках) дисків зберігаються дані і службова інформація, необхідна для синхронізації роботи дисковода. Треки - це концентричні кільця,

логічно розташовані на поверхні дисків. Однойменні треки всіх наявних у пакеті поверхонь можна простромити уявними циліндрами. Ці «циліндри» використовуються як адреси при звертанні до даних.

Записують дані на диск і читають їх магнітні голівки з дуже малою вагою. Кілька таких голівок об'єднані в блок, керований із загального спеціального центру дисковода – автоматичної сервосистеми, що стежить. Під час позиціювання голівки (тобто послідовного пошуку потрібної доріжки) сервосистема, що стежить, читає цифрову інформацію – серводані, що записані в службові поля синхронізації доріжок диска. Серводані порівнюються з інформацією про місце призначення голівки, отриманої дисководом з контролера.

Цифрова інформація про різницю між номерами того циліндра, де знаходиться голівка, і того, де вона повинна знаходитися, надходить на обробку в так званий контур зворотного зв'язку. Цифровий сигнал у логіці позиціювання перетворюється в аналоговий, а потім виробляється постійний електричний струм певної величини і полярності.

За конструкцією позиціонер може бути соленоїдним або приводом обертання, або лінійним двигуном. Лінійний двигун всовує блок голівок у дисковий пакет не по тангенціальній кривій (як це робить привід обертання), а по прямій лінії. Котушка соленоїда позиціонера жорстко з'єднана з блоком голівок. Таким чином, рух котушки з визначеною швидкістю й у визначеному напрямку передається на всі наявні у дисководі голівки.

Завдяки надлишковому тиску, що виникає в граничних областях обертових пластин дисків, створюється аеродинамічний момент піднімальної сили голівок, і вони «плавають» - кожна над своєю поверхнею.

Система, що стежить, постійно інформує логіку позиціювання про те, де розташовуються голівки, і на котушку індуктивності соленоїда увесь час передаються все нові і нові значення величин постійного струму, а тому чим менша їх різниця, тим менший струм і, отже, швидкість руху блоку голівок. Такий пошук від доріжки до доріжки називається поперечним пошуком циліндра.

Може здатися, що голівка «прилипає» до доріжки. Це не так. Потрапивши на трек, голівка постійно прагне до його середньої лінії. Відхиляючись від цієї лінії, вона описує уздовж неї своєрідну синусоїду, а сервосистема, що стежить, миттєво реагує на зміну характеристик сигналу, що викликана таким відхиленням. Формується протифазний сигнал, і блок голівок переміщується в зворотному напрямку. Чим менше неузгодженість, тим точніше настроювання і вищий рівень корисного сигналу.

Системі, що стежить, постійно необхідна інформація про те, у якому циліндрі знаходиться блок голівок; визначається це за допомогою збережених на поверхні диска серводаних.

Серводані читаються при позиціюванні окремою голівкою зі спеціальної виділеної сервоповерхні або робочою голівкою зі службового запису (вбудованого сервозапису), що входить у формат сектора доріжки.

У дисководах з високою щільністю запису може використовуватися технологія самосинхронізації NCH (по clock head), що розроблена в IBM. Технологія NCH дозволяє замінити дорогий метод формування сервозаписів (servowriting), який вимагає юстировки приладів на заводі-виготовлювачі, використанням спеціального співпроцесора. Такий співпроцесор бере участь у формуванні серводаних і розміщенні їх між сегментами секторів дисків. За допомогою співпроцесора контролюється вирівнювання голівок щодо запису на доріжці при точному позиціюванні блока голівок. Це дозволяє набагато прискорити відстеження позиції голівки щодо запису і коректування її положення позиціонером. Співпроцесор реалізує алгоритм виправлення помилок, що через коливальні процеси час від часу виникають у сервосистемі. Завдяки співпроцесору скорочується обсяг дискового простору, що витрачається на сервозапис.

У сервосистемах високопродуктивних дисководів можна зустріти ще одну розробку фахівців IBM - технологію активного демпфірування блоку голівок. Метод цей розрахований на дисководи з високою щільністю запису і швидкістю обертання шпиндельного двигуна більше 7200 обер./хв. Сервосистема дозволяє коректувати положення позиціонера при механічних поперечних флуктуаціях через ряд різних дестабілізуючих факторів, а також уникнути механічного резонансу голівок. Метод цей полягає у використанні спеціального частотного фільтра.

Якщо ним виявлена одна з частот, що включена у таблицю критичних частот, у сервосистему направляється цифровий код, на підставі якого в котушку посилається постійний струм відповідної сили і полярності. Це дозволяє голівці точно розташуватися в межах запису на доріжці.

Оскільки, крім даних на доріжці записана і службова інформація, розрізняють ємність форматованого (formatted) диска (тобто за винятком зайнятого службовими записами простору) і неформатованого (formatless). Ємність неформатованого диска, природно, більша.

Шпиндельні двигуни приводу НЖМД бувають двофазними або трифазними, до того ж в останніх автоматично регулюється швидкість обертання. Для кріплення шпиндельних двигунів можуть використовуватися гідродинамічні підшипники (fluid dynamic bearing). Ця перспективна технологія, узята з області космічних розробок. У робочих частин гідродинамічного підшипника немає механічних контактів типу метал/метал. Вони розділені не кульками, а спеціальною масляною плівкою, що демпфірує внутрішні і зовнішні коливання, амортизує удари, знижує рівень шуму. Завдяки їй термін служби підшипника необмежений.

Автоматичне керування шпиндельним двигуном полягає в регулюванні його швидкості за допомогою *індуктивних* датчиків або *перетворювачів Холу.* Ефект Холу виникає в напівпровіднику, через який протікає струм і який знаходиться в електромагнітному полі із силовими лініями, що перпендикулярні руху струму в напівпровіднику. У цьому випадку на гранях напівпровідника, що перпендикулярні напрямку руху струму, виникає різниця потенціалів і напруга прямо залежить від величини магнітної індукції. На принципі ефекту Холу побудований датчик, що дозволяє перетворити швидкість обертання на валу в постійну напругу керування двигуном.

Керування даними. Канал керування даними включає тракт читання/запису, призначений для обміну кодованими даними між дисковим інтерфейсом, наприклад SCSI або ATA, і дисковою пам'яттю. У тракті читання/запису НЖМД виробляються імпульси струму, що надходять у голівки запису. З кожним таким імпульсом напрямок силових ліній магнітного потоку в зазорі сердечника голівки міняється на протилежний. Це відповідає елементу запису на феромагнітний носій і називається *переходом перемагнічування.*

Всі операції дисковод виконує під керуванням команд, які він одержує від адаптера. Команди керування, які генеруються адаптерами різних інтерфейсів, відрізняються рівнем складності. Наприклад, команди інтерфейса SCSI складніші, ніж команди, що направляються в дисковод адаптером EIDE. Вінчестер декодує команду і послідовно виконує кілька операцій. Він визначає, до якої області дискової пам'яті відбувається звертання, яка операція обміну виконується – запису чи читання.

Логіка вінчестера відповідним чином набудовує тракт запису/читання і систему позиціювання. Потім логіка керування дозволяє позиціонувати блок голівок, а у вузлі керування голівками вибирається та з них, що необхідна. На заключній стадії запису дані з буфера диска через підсилювачі, ланцюги формування і голівку записуються на диску. Читаються всі тією ж голівкою. Дані при цьому пересилаються в зворотному напрямку.

Перш ніж інформація займе своє місце на поверхні магнітного диска, вона кодується в контролері. У сучасних дисководах використовується метод кодування *RLL (Run Length Limited) - з обмеженням відстані між переходами перемагнічування* - або його удосконалений варіант *ARLL (Advanced RLL).*

Яким чином при кодуванні (декодуванні) логічні одиниці відокремлюються від нулів? Найпростіше кодування полягає в «підмішуванні» до корисного сигналу спеціальних синхросигналів. Під час передачі одиниць додатковий синхроімпульс множить частоту запису на два, а при передачі нуля частота залишається колишньою. Така частотна модуляція з подвоєнням частоти *2FM* і її різновид *MFM (Modified Frequency Modulation}* у вінчестерах уже не застосовується.

RLL - це теж один з типів частотної модуляції, доповнений спеціальним алгоритмом кодування не окремих імпульсів, а цілих груп. Вся інформація, що надходить на обробку в RLL-кодувальник, піддається груповому перекодуванню відповідно до даних зі спеціальної таблиці. Між окремими переходами перемагнічування дуже багато інформаційних інтервалів різної довжини. Тому якість магнітного запису на носій повинна виключати імовірність утрати навіть одного біта.

Цифри поруч з абревіатурою RLL відображають діапазон надмірності, що вводиться кодом. Наприклад, запис *RLL 8.9* означає, що між двома переходами перемагнічування може бути закодовано від восьми до дев'яти бітів інформації. Чим менший «перепад» цих значень, тим вища потужність сигналу і краща здатність вінчестера зберігати і відновлювати дані.

Завдяки технології запису з кодуванням RLL 8.9 і новій технологічній базі виготовлення дисків з надтонким покриттям і високій *коерцитивності* можна підвищити щільність запису і збільшити ємність дискової пам'яті. *(Коерцитивність* - це здатність речовини зберігати намагніченість при нульовій магнітній індукції.)

Технологія запису і читання інформації. Швидкодія дисковода залежить від швидкості обертання шпинделя, подовжньої і поперечної щільності запису і технологій, що забезпечують точність запису і читання інформації з магнітного носія на всіх етапах. Найнебезпечніша ділянка в цьому ланцюжку — механічна зв'язка *диск - голівка.*

Як підкладку чи *основу* диска вінчестера застосовують круглі поліровані алюмінієві або скляні пластини з золотим, срібним чи нікелевим покриттям. Вінчестер IBM Deskstar 75GXP - приклад накопичувача, диски якого виготовлені зі скляних пластин. Скляні пластини дозволяють одержати високу чистоту поверхні при мінімальних витратах на обробку матеріалу. У результаті, використовуючи сучасні технології запису інформації, на поверхню такого диска можна записувати дані з щільністю 14.3 Гбіт/дюйм^2.

На основу наноситься найтонший шар зберігання *даних (thin film media),* такий, як оксид кобальту або інший *феромагнетик.* При магнітному записі окремі частки основи утворять керовані магнітним потоком області намагніченості. Відстань між цими магнітними областями повинна бути якнайменшою: це дозволяє збільшити щільність запису.

Високої щільності запису і прийнятного співвідношення сигнал/шум можна досягти тільки при використанні тонкоплівного покриття товщиною близько 1 мкм. Крім того, носій повинний володіти високою *коерцитивністю* матеріалу.

При записі на феромагнетик кожне перемагнічування - це немов створення маленького магнітика, що має свої силові лінії. Уявимо собі, що сусідні елементарні магнітики при високій щільності запису стикаються настільки, що

їхні силові лінії впливають одна на одну. Це приведе до взаємної компенсації або перекручування магнітних полів і, як результат, до втрати даних. Щоб уникнути втрат даних, величину струму для запису варто зменшити. Висококоерцитивний носій при цьому може зберігати елементарні магнітики без ризику взаємодії однієї з одною сусідніх силових ліній. Однак рівень сигналу, збереженого на такому носії, буде дуже малий. Щоб одержати при читанні прийнятний рівень віддачі корисного сигналу, чутливість голівок повинна бути високою, а самі вони повинні бути якнайближче до носія.

Найпоширеніші технології створення тонкоплівкових покриттів — анодування і напилювання. *Анодування* — це осадження на підкладку электрогальванічним способом молекул металу, солі якого містяться в розчині. Товщина магнітного шару, що осаджується таким способом, - близько 2 мкм. *Напилювання* — це створення на підкладці шляхом безупинного осадження у вакуумі декількох тонких плівок — металу, феромагнітного носія і захисного вуглецевого покриття. Ця технологія дорожча, ніж анодування, але вона дозволяє формувати тонкоплівкові покриття пластин диска.

Технологією майбутнього стане використання як носія *нанокристалічної суперструктури,* чи *надрешітки.* Суть цього нового способу магнітного запису, запропонованого фахівцями IBM, полягає у побудові тривимірних структур молекулярної величини (4нм), виділених з розчину під дією магнітного поля. Для цього використовуються *атоми* заліза і платини, оточені плівкою ізолюючих органічних молекул.

У результаті термообробки нанесеної плівки залізо-платинові кристали утворять кристалічні ґрати. Стабільний магнітний заряд зберігається у вузлах ґрат, що дуже щільно упаковані і ніяк не впливають один на одного. Щільність запису на таку основу може досягти кількох терабітів даних на квадратний дюйм. Зберігати такий запис, на думку вчених, можна більше 10 років.

Між обертовими пластинами диска і плаваючими, завдяки зоні високого тиску, голівками є *повітряний зазор (head gap).* Пішли в небуття індуктивні тонкоплівкові класичні голівки з «висотою польоту» 1.4 мкм і більше. Зусиллями фахівців корпорацій IBM, Seagate і Fujitsu були впроваджені магніторезистивні голівки *(MP; Magnetic-Resistive, MR), що* дозволили зменшити зазор до 0.13 мкм і досягти щільності запису до 5 Гбит/дюйм^2. Серед переваг МР-голівок — мала вага, мініатюрність, нечутливість до швидкості обертання носія, невисока чутливість до шумових сигналів і висока — до корисного сигналу (переходів перемагнічування). Усе це стало можливим завдяки магніторезистивному ефекту Кельвіна, який полягає в зміні опору деяких феромагнітних сплавів під впливом магнітного поля. Через елемент МР- голівки, що читає, постійно проходить невеликий струм, а зміна магнітного поля під елементом змінює його опір і, отже, величину струму, що використовується для відтворення інформації.

МР-голівка складається з двох частин, об'єднаних загальним шаром. Секція запису подана тонкоплівковою індуктивною голівкою. Функції ж читання покладені на іншу секцію — *магніторезистивний сенсор (МР-сенсор}*. МР-сенсор виготовляється з залізо-нікелевої (NiFe) плівки; її опір у магнітному полі елементів запису на носії може мінятися.

Сенсор покритий двома шарами. Один захищає його від побічних магнітних полів; у другого — високий опір, і він є магнітопроникним шаром, що відокремлює елемент запису від елементу читання. Такий «пошаровий пиріг» називається *SAL (Soft Adjacent Layer).*

МР-сенсор дуже чутливий до індукції магнітного поля. Це дозволяє читати магнітні сигнали такого рівня, який був недоступний магнітним тонкоплівковим голівкам, що літають на тій же висоті.

Крім того, МР-голівки реагують не тільки на *переходи перемагнічування,* але і на рівні магнітних сигналів. Це дозволяє підвищувати не тільки подовжню, але і поперечну щільність запису. Для роботи при щільності більше 5 Гбіт/дюйм^2 призначені удосконалені МР-голівки *MRX (MR Extended).* Технологія їхнього виготовлення така ж, як і в МР-голівок, але вони мініатюрніші і піддаються більш ретельному вхідному контролю.

Щоб досягти щільності запису до 10 Гбіт/дюйм^2 і вище, застосовується технологія виготовлення *GMR-голівок (Giant MR). Я*кщо в звичайних МР-голівках використовується одна плівка-сенсор, то в голівках типу GMR сенсорів два. Вони розділені тонким електропровідним шаром. Така конструкція дозволяє підсилити ефект сенсорного датчика. У результаті чутливість доменів до слабких магнітних полів запису на носій підвищується, що дає можливість ще більше збільшити щільність запису.

Наступним етапом розвитку технології GMR-голівок стали голівки зі *спін-затвором (spin-valve heads).* Вони забезпечують збільшення ємності дискової пам'яті за рахунок подальшого росту чутливості магнітних голівок до слабких магнітних сигналів. Спін-технологія базується на GMR-ефекті посилення сигналів слабких магнітних полів. Спін-затвор складається з МР-сенсорів, виготовлених із семи-восьми шарів магнітного і немагнітного металів. Товщина кожного такого шару не перевищує п'яти атомів.

Існують і інші способи підвищення продуктивності вінчестерів. Наприклад, корпорацією Seagate Technology був запропонований *«вінчестер з оптичною підтримкою" OAW (Optically Assisted Winchester).* Диск OAW виготовлений з алюмінію або з пластику і покритий трьома шарами: аморфною (а не кристалічною) композицією рідкоземельних металів, що забезпечує високу поверхневу щільність запису, і двома захисними покриттями. Місце звичайної МР-голівки займає лазерний робочий вузол зі світловодом. Під час запису промінь лазера прогріває точку поверхні носія до температури сплаву Кюрі, і магнітні властивості прогрітої ділянки міняються. Запис займає всього

декілька наносекунд. Для читання лазер переключається в режим роботи зі зниженою потужністю. Намагніченість поверхні запису визначається за поляризацією відбитого лазерного променя. Подібна обробка інформації про вектор поляризації поверхневого шару застосовується в магнітооптичних дисках.

Підвищенню щільності запису і читанню даних з більш високою швидкістю сприяє технологія *часткового відгуку і максимальної правдоподібності — PRML (Portial Response Maximum Likelihood).* При традиційному способі запису/читання даних магнітними голівками кожна група бітів кодується відповідно до зміни частоти і фази сигналів. Позиції магнітного запису на носії відокремлюються один від одного за допомогою *переходів перемагнічування* областей носія. Саме на такі переходи реагують магнітні голівки. Для запису-читання використовуються ключові тригерні схеми. Вони працюють з логічними рівнями сигналів, що мають великі перепади амплітуд. Ці перепади гарантують ідентифікацію сигналів при читанні.

Відповідно до методу PRML використовується спеціальний алгоритм, що дозволяє згладити перепади рівнів сигналів. При читанні передбачається багаторазове читання кожного елемента запису і відновлення запису відповідно до методу максимальної правдоподібності.

Таким чином, вірогідність читаної інформації не так залежить від перепадів рівнів сигналів, і перекручування форми імпульсу (у межах допуску) не приведе до втрати даних. Це дозволяє збільшити щільність запису і підвищити ємність дискової пам'яті.

Надійність вінчестера. Як відомо, емпіричним показником надійності служить *середній час наробітку на відмовлення — MTBF (Mean Time Between Failures).* У сучасних вінчестерів вона складає від 300 тисяч до мільйона годин, якщо тільки через примхи долі або необережність дисковод випадково не упаде або не вдариться.

Довголіття дисковода скорочується через порушення електричних параметрів, через усілякі дестабілізуючі фактори — перегріви елементів, неправильне під’єднання пристроїв, збоїв живлення, розриву контактів і коротких замикань. Якогось одного загального засобу для захисту від усього цього немає. Проте основні виробники дисководів приділяють велику увагу питанням захисту пристроїв від згубних впливів зсередини і ззовні.

Одна з основних технологій забезпечення надійності складових комп'ютера — *SMART* (разом з її різновидами). Особливі датчики, встановлені у певних вузлах пристроїв комп'ютера, повідомляють системі моніторингу про зміну стану підсистем. Це дозволяє контролювати обстановку в різних вузлах і в периферійному устаткуванні комп'ютера і, залежно від сформованих умов експлуатації, підключати ті чи інші засоби захисту. До таких засобів відносяться:

- ***система захисту від ударів і вібрації*** *True Track Servo* (розроблена в IBM) — використовується для компенсації виходу голівок за межі робочого запису;
- ***система передбачення збоїв*** *Predictive Failure Analysis* (IBM) — стежить за робочими параметрами вінчестера;
- ***система термоконтролю*** *Drive TIP(Drive Temperature Indicator Processor;* розроблена в IBM) — спирається на моніторинг показань термодатчиків, розташованих у робочій зоні диска. Аналогічна система — *Thermal Monitor* — розроблена в Western Digital;
- ***діагностична система контролю за станом диска*** *Drive Fitness Test* (IBM) — працює разом із системою SMART.

Аналогічні функції створених у Quantum систем діагностики поверхні *Data Protection System* і *Seagate Sea Tools;*

- ***система захисту від ударних навантажень*** *(Shock Block;* розроблена в Maxtor) — удосконалений механізм підвіски голівок, що перешкоджає вертикальному переміщенню голівок при ударах і сприяє підвищенню ударостійкості голівок при їх установці;
- ***система діагностики помилок даних*** *Max Safe* (створений у Maxtor) — удосконалений спосіб *виявлення і корекції помилок даних ECC (Error Correction Code),* а також перевірка ширини зазору між голівкою і поверхнею диска. Моніторинг показань датчика ширини зазору між голівками і поверхнями дисків використовується також у системі Western Digital *Fly Height Monitor;*
- ***система захисту від тряски і ударів*** *Shock Protection System* (створена в Quantum) — запобігає вертикальному переміщенню голівок, розсіюючи ударну енергію по всьому механізму приводу голівок дисковода;
- ***захист від статичного розряду*** *Soft Sea Shield Cover* (розроблений у Seagate) — являє собою металеву пластину на

корпусі дисковода, що охороняє елементи пристрою від статичної електрики;

- ***система захисту від ударів*** *G-Force Protection* (Seagate) — дозволяє зменшити ризик ушкодження вузлів дисковода випадковими ударними навантаженнями. У цій технології використовуються: гідродинамічна підвіска шпиндельного двигуна; посилене кріплення дискових пластин; механізм сервокомпенсації можливих зсувів голівок щодо записів на доріжках під час удару; мініатюризація голівок; збільшення зазорів між голівками і поверхнями дисків, зазорів між диском і важелем механізму позиціювання голівок;
- ***система вбудованої діагностики і регенерації даних*** *Life Guard* (створена в Western Digital) — комплект сервісних утиліт, що включаються в програмне забезпечення дисководів WD. За результатами перевірки всіх

зайнятих даними секторів перекручена інформація коректується, перевіряється і пересилається на колишнє місце або в інший, не зіпсований сектор. При відмовленні кожного з вузлів дисковода відбувається примусове паркування голівок.

3.12. ТЕНДЕНЦІЇ РОЗВИТКУ ПАМ'ЯТІ КОМП'ЮТЕРА

Розвиток сучасної електронної технології, яке умовно почалося 50 років тому з винаходу транзистора і десятьма роками пізніше - твердотільної інтегральної схеми, як і раніше вписується у відкритий в 1965 році співробітником корпорації Intel Гордоном Муром закон: кожен новий чіп, вироблений через 18-24 місяці після попереднього, має приблизно вдвічі більшу ємність пам'яті. Даний рух має природним чином закінчитися через 10-15 років з огляду на необхідність переходу до використання взаємодії елементів на основі квантової механіки.

Дійсно, розміри елементів повинні бути порівнянні з нанометром (нм), рівним 10^{-9} метра, який в тисячу разів менше мікрометра (мкм), або, постарому, мікрона (мк).

В останні роки намітився реальний вихід із ситуації, заснований на досягненнях фізичних і хімічних наук. Розглянемо основні тенденції розвитку пам'яті комп'ютерів.

Комплект модулів пам'яті 128 ГБ (8 x 16 ГБ) DDR4

Компанія Corsair, світовий лідер високопродуктивних компонентів для ПК, оголосила перший в світі комплект модулів небуферірованої оперативної пам'яті DDR4 для персональних комп'ютерів із загальною ємністю 128 ГБ. Запропоновано дві серії: Vengeance LPX і Dominator Platinum.

Комплект нових модулів DDR4 від Corsair ємністю 128 ГБ (8 x 16 ГБ) розроблений для самих «свіжих» материнських плат серії Intel X99 з підтримкою XMP 2.0. Запропоновано комплекти для тактових частот 2666 і 2400 МГц, але очікується поліпшення цього параметра в самому найближчому майбутньому. Як і вся інша продукція Corsair, нові модулі пам'яті мають довічну гарантію.

XMP 2.0 на материнських платах - це Extreme Memory Profile (профіль екстремальної пам'яті) від Intel, який дозволяє змінювати часові параметри доступу до пам'яті, грубо кажучи, в трьох режимах: звичайний, ентузіаст (enthusiast) і екстремал (extreme), тобто розганяти модулі пам'яті на свій смак, але без порушення узгодженості декількох параметрів.

Пам'ять DDR4 має повну назву DDR4 SDRAM, тобто Duble Data Rate версія 4 Synchronous Dynamic Random Access Memory або четвертий за рахунком інтерфейс з подвоєною швидкістю передачі даних (DDR) синхронної динамічної пам'яті RAM. Слово «динамічна» в цій назві означає, що навіть з

поданням живлення відбувається розряд комірки пам'яті, тому необхідна спеціальна службова процедура «підзарядки» комірок, які давно не оновлювалися. Слово «синхронна» відзначає ще один недолік - команди подаються не в будь-який момент (асинхронний принцип роботи), а тільки по тактовою сигналу. Незважаючи на обидва недоліки, пам'ять SDRAM дозволяє знизити виробничі витрати і підвищити об’єм в порівнянні з іншими типами RAM, а з затримками виробники борються еволюційними методами, постійно підвищуючи тактову частоту (яка як раз і визначає синхронізацію з шиною). Основним трюком на цьому шляху залишається конвеєрна обробка запитів звернення до пам'яті SDRAM, що передбачає буферірування або відсутність буфера при зверненні до пам'яті для читання або запису.

Пам'ять DDR4 SDRAM з'явилася на ринку в другому кварталі 2014 року. Стандарт DDR4 дозволяє мати модуль пам'яті ємністю 128 ГБ (замість 16 ГБ в DDR3), причому передбачається напруга живлення 1,2 В з тактовою частотою 1600 ... 3200 МГц (порівняйте: 800 ... 2400 МГц з напругою 1,5 або 1,65 В для DDR3). Тобто, стандарт DDR4 залишає величезні перспективи для подальшого зростання характеристик пам'яті, якщо зіставити з оголошеними модулями компанії Corsair.

Резистивна пам'ять RRAM

RRAM - мемристор - пристрій, здатний змінювати опір в залежності від величини струму, який проходить. Сама назва елемента є похідним від англійських слів memory (пам'ять) і resistor (електричний опір), тобто по суті прилад має здатність "пам'ятати" обсяг заряду, який встиг пройти по ланцюгу до моменту відключення живлення.

Мемристор був запропонований в 1971 році професором Леоном Чуа з університету Каліфорнії в Берклі. Довгий час мемристор вважався лише теоретичною концепцією, нездійсненною на практиці, хоча деякі вчені і пророкували цьому гіпотетичному елементу велике і славне майбутнє. Сам Леон Чуа стверджував, що рано чи пізно мемристор, поряд з конденсатором, резистором і котушкою індуктивності, стане четвертим базовим елементом електронних схем. Але головна роль мемристора, на думку професора Чуа, полягає в радикальній зміні принципів створення електронних схем. Аж до теперішнього часу наріжним каменем схемотехніки було співвідношення між напругою і зарядом на тих чи інших елементах схеми, однак більш перспективним підходом, стверджує професор, є розгляд співвідношення між зарядом і швидкістю зміни напруги.

Лабораторний зразок мемристора був створений в 2008 році колективом вчених на чолі з Р.С. Вільямсом в дослідницькій лабораторії фірми Hewlett-Packard. Явище гістерезису, що спостерігається в мемристорі, дозволяє використовувати його в якості елементу пам'яті.

У порівнянні з сучасними типами пам'яті мемристори володіють важливими перевагами. Оскільки вони енергонезалежний і мають високу швидкодію, запам'ятовуючі пристрої на їх основі дозволять замінити флеш-пам'ять і пам'ять DRAM. Завантаження відбуватиметься миттєво, минаючи зчитування інформації з "повільних" жорстких дисків.

На думку Грега Снайдера (фахівець компанії HP), мемристор стане одним з основних елементів нанопристроїв, емулює роботу людського мозку (мініатюрні нанопристрої будуть об'єднані в єдину мережу, а мемристор стане елементом, відповідальним за "пам'ять" штучного інтелекту).

Як стверджується, створення нового елемента може стати найбільш значною подією десятиліття в мікроелектроніці і привести до кардинальних змін в технології зберігання інформації, оскільки мемристор здатний зберігати дані без витрат енергії на протязі тривалого часу. Мікросхеми пам'яті, побудовані на базі мемристоров, забезпечать можливість моментального включення комп'ютерів за рахунок відмови від необхідності початкового завантаження, зниження енергоспоживання мобільних пристроїв та інші перспективи.

На думку HP, нова технологія цілком може претендувати на роль універсальної пам'яті майбутнього, яка одночасно замінить використовувану зараз динамічну пам'ять з довільним доступом і флеш-пам'ять.

Мемристори дозволять створювати чіпи пам'яті майбутнього, які здатні зберігати дані без електрики протягом тривалого періоду часу, що дозволить уникнути довгого процесу завантаження комп'ютерів, підвищити їх продуктивність, а також багаторазово знизити енергоспоживання електронної техніки.

Магніторезистивна пам’ять (MRAM)

MRAM - оперативна пам'ять, в якому, на відміну від інших типів пам'яті, інформація зберігається не у вигляді електричних зарядів або струмів, а в магнітних елементах. Останні сформовані з двох феромагнітних шарів, розділених тонким шаром діелектрика. Магніторезистивна пам'ять володіє швидкодією, яку можна порівняти з пам'яттю типу SRAM, такий же щільністю комірок, але меншим енергоспоживанням, ніж у пам'яті типу DRAM. Вона швидша і не деградує в часі в порівнянні з Flash-пам'яттю.

PRAM-пам'ять (phase change random access memory)

Мова йде про так звану "пам'ять зі зміною фазового стану" (phase change memory, PCM або PRAM), заснованої на здатності халькогенида (chalcogenide) під впливом нагріву і електричних полів змінювати свій стан з непровідного аморфного в провідний кристалічний. Даний вид пам'яті прекрасно справляється як із зберіганням великих об’ємів даних, так і зі зберіганням

виконуваного коду, уявляючи, таким чином, дивовижний сплав флеш-пам'яті і динамічної пам'яті з довільним доступом.

При нагріванні і наступному охолодженні халькогенід швидко переходить з стабільного аморфного в стабільний кристалічний стан. У аморфному стані коефіцієнт відбивання матеріалу невеликий, а опір великий, в кристалічному - коефіцієнт віддзеркалення великий, а опір малий. Це і дозволяє зберігати в пам'яті такого типу логічні "0" і "1". Використовуваний в сучасних схемах пам'яті халькогенід - сплав германію, сурми і телуру. Комірка пам'яті складається з верхнього електрода, шару халькогенида і резистивного нагрівального елементу. Так само, як і в MRAM, при зчитуванні даних вимірюється опір комірки пам'яті, але на відміну від MRAM відношення опорів велике - більше 100. При запису даних халькогенід нагрівається до температури, що перевищує його точку плавлення, і потім швидко охолоджується, тобто переходить в аморфну фазу. Щоб перевести матеріал в кристалічний стан, комірка нагрівається до температури нижче точки плавлення і витримується при ній протягом ~ 50 нс.

До переваг PCM-пам'яті відносяться: проста структура, мала площа комірки пам'яті, можливість неруйнівного зчитування і селективного перезапису даних без стирання, мала споживана потужність і велика кількість циклів перезапису - більш 10^{13}. PCM-матриці, які не потребують при виготовленні високо температурних процесів, легко об'єднувати з кремнієвими логічними пристроями.

Лідерами дослідного виробництва PCM в даний час є Samsung і спільне підприємство Intel і STMicroelectronics − компанія Numonyx.

Освоюючи PRAM-пам'ять, Samsung сподівається в майбутньому зайняти лідируючі позиції на цьому ринку. Samsung Electronics Co. Ltd розпочала поставки першого повністю функціонального прототипу PRAM-чіпа пам'яті (Phase-change Random Access Memory, або пам'ять з довільним доступом на основі фазових перетворень) ємністю 512 Мбіт.

Нова технологія дозволяє перезаписувати дані без попереднього затирання вже існуючих, що виливається в 30-кратну перевагу в швидкості над традиційною флеш-пам'яттю. Очікується десятикратне збільшення терміну служби PRAM-чіпів. Нові чіпи компактніше і, що важливо, на 10% дешевше.

Сегнетоэлектрическая энергонезависимая память (FRAM)

FRAM за своїм устроєм схожа з DRAM (конденсатор на основі сегнетоелектріка). Запис відбувається шляхом зміни вектора поляризації сегнетоелектричного шару різницею потенціалів між електродами. Серед переваг FRAM перед Flash-пам'яттю можна виділити низьке енергоспоживання, швидкий запис інформації і суттєво збільшене максимальне число циклів

перезапису, що перевищує 10^{14}. До недоліків FRAM відносять набагато більш низьку щільність запису, обмежену ємність і більш високу вартість.

Одним зі світових лідерів в області розробки і виробництва електронних компонентів самого різного призначення за запатентованою технологією створення енергонезалежних сегнетоелектричних ОЗП (FRAM) є Ramtron International. У число ліцензіатів або партнерів по освоєнню технології FRAM входять такі великі компанії, як Texas Instruments, Fujitsu, Toshiba, Samsung, Hynix і ін. В їх роботах дані прогнози подальшого розвитку пристроїв FRAM, розглянуті перспективи заміни Flash-пам'яті і зроблено висновок про те, що пристрої перепрограмовуваної пам'яті на основі сегнетоелектриків мають всі шанси зайняти міцну конкурентоспроможну позицію серед інших пристроїв енергонезалежної пам'яті за умови, що будуть вирішені проблеми їх старіння і надійності, а також забезпечена можливість неруйнуючого зчитування інформації.

Полімерна пам'ять

Шведська компанія ThinFilm вважає, що пам'ять майбутнього буде заснована на пластмасі. Дуже тонкий лист полімеру, затиснутий між двома сітками крихітних електродів, являє собою матрицю пам'яті. В кожному перетині шахівниці електродів (один провід зверху шару полімеру, а інший - знизу його) створюється бістабільна комірка пам'яті. Електрична напруга, прикладена до даної комірки, може змінювати структуру полімеру, переводячи його з одного стабільного стану в інше. Альтернативні стани полімеру відповідають логічним нулю і одиниці. Даний стан може зберігатися досить тривалий час, так як зміна стану полімеру носить хімічний характер. З цієї ж причини стан комірки енергонезалежний.

Практично технологія створення пристроїв пам'яті даного виду може являти собою розкочування рулону полімеру і нагадувати роботу газетної друкарні. Особливо вражає щільність такої пам'яті. Якщо зараз одна комірка SRAM займає площу розміром 4-6 квадратних мікрометрів, пропонована технологія дозволить розмістити елемент пам'яті на площі близько однієї чверті квадратного мікрометра. Причина цього - відсутність у складі комірки активних елементів (транзисторів). Активні елементи адресування, зчитування і запису можуть знаходитися по периметру матриці пам'яті або, як альтернатива, вище або нижче її. Якщо врахувати, що сучасна пам'ять ємністю в один гігабіт вимагає застосування від 1,5 до 6,5 млрд. транзисторів, то система полімерної пам'яті обмежується приблизно половиною мільйона активних елементів. Для підвищення щільності пам'яті листи полімеру можуть бути складені стопкою. Згідно з розрахунками фахівців фірми ThinFilm, пристрій пам'яті розміром з кредитну картку, побудований за цією технологією, міг б зберігати 60 000 фільмів в стандарті DVD; або 126 років музики в стандарті MP3; або 400 000

компакт-дисків; або 250 мільйонів цифрових фотографій високої роздільної здатності.

Молекулярна пам'ять

Якщо попередній підхід до вдосконалення систем пам'яті можна умовно назвати еволюційним (шляхом масштабування провідники стають все тонше, також поступово тоншає шар полімеру), то воістину революційним є ідея використання в якості елементів електронної техніки молекул.

Вже відносно давно був створений прототип системи пам'яті, що використовує в якості комірок молекули протеїну, який називається бактеріородопсин (bacteriorhodopsin). Він має пурпурний колір, поглинає світло і присутній в мембрані мікроорганізму, з назвою halobacterium halobium. Цей мікроорганізм був виявлений в соляних болотах, де температура може досягати +150° С. Коли рівень вмісту кисню в навколишньому середовищі настільки низький, що для отримання енергії неможливо використовувати подих (окислювання), він для фотосинтезу використовує протеїн.

Бактеріородопсин був обраний тому, що фотоцикл (послідовність структурних змін, які молекула зазнає при реакції зі світлом) робить цю молекулу ідеальним логічним запам'ятовуючим елементом типу «&» або типу перемикача з одного стану в інший (тригер). Як показали дослідження, bR-стан (логічне значення «0») і Q-стан (логічне значення «1») є проміжними станами молекули і можуть залишатися стабільними протягом багатьох років.

Іншою важливою особливістю бактеріородопсина є те, що ці два стани мають спектри поглинання, які помітно відрізняються. Це дозволяє легко визначити поточний стан молекули за допомогою лазера, налаштованого на відповідну частоту. Дані, записані в такому пристрої зберігання даних, можуть зберігатися приблизно п'ять років.

Був побудований прототип системи пам'яті, в якому бактеріородопсин запам'ятовує дані в тривимірній матриці. Така матриця являє собою кювету (прозора посудина), заповнену поліакридним гелем, в який поміщений протеїн. Кювета має довгасту форму розміром 1x1x2 дюйма. Протеїн, який знаходиться в bR-стані, фіксується в просторі при полімеризації гелю. Кювету оточують батарея лазерів і детекторна матриця, побудована на базі приладу, що використовує принцип зарядової інжекції (CID - Charge Injection Device), які служать для запису і читання даних.

При запису даних для перекладу молекул у Q-стан спочатку треба використовувати жовтий «сторінковий» лазер. Просторовий світловий модулятор (SLM), який, як говорилося раніше, являє собою LCD-матрицю, що створює маску на шляху променя, викликає виникнення активної (збудженої) площини в матеріалі усередині кювети. Ця енергоактивна площина є сторінкою даних, яка може вміщати масив розміром 4096x4096 комірок.

Для повернення протеїну в стан спокою використовується червоний лазер, що записує і розташовується під прямим кутом по відношенню до жовтого. Другий SLM також управляється матрицею двійкових даних і, таким чином, створює на шляху променя відповідну маску, тому опроміненню піддадуться тільки певні точки сторінки. Молекули в цих місцях перейдуть у Q-стан і будуть представляти двійкову одиницю. Частина сторінки повернеться в початкове bR-стан і буде представляти виконавчі нулі. Для того щоб прочитати дані, треба знову використовувати сторінковий лазер, який переводить читану сторінку в Q-стан. Це робиться для того, щоб в подальшому за допомогою відмінності в спектрах поглинання ідентифікувати виконавчі нулі і одиниці.

Для стирання даних досить короткого імпульсу синього лазера, щоб повернути молекули з Q-стану в початковий bR-стан. Синє світло не обов'язково повинно йти від лазера – можна стерти всю кювету за допомогою звичайної ультрафіолетової лампи.

Запропонована система по швидкодії близька до напівпровідникової пам'яті. Теоретично кювета, що містить протеїн, може вмістити близько одного терабіта даних. Обмеження на ємність пов'язані, в основному, з проблемами лінзової системи і якістю протеїну.

Така молекулярна пам'ять «першого покоління» вже має певні переваги в порівнянні з традиційною напівпровідниковою пам'яттю. По-перше, вона заснована на протеїні, що виробляється у великій кількості і за невисокою ціною, чому сприяють досягнення генної інженерії. По-друге, система може функціонувати в більш широкому діапазоні температур, ніж існуюча напівпровідникова пам'ять. По-третє, така пам'ять енергонезалежна. Нарешті, кювети з даними можна довго і безпечно зберігати.

КОНТРОЛЬНІ ПИТАННЯ

1. Які операції визначає поняття «Звернення до ЗП»?
2. Чим викликана необхідність побудови системи пам'яті за ієрархічним принципом?
3. Поясніть принцип побудови адресного ЗП.
4. Як здійснюється пошук інформації в асоціативному ЗП?
5. Поясніть призначення маски в асоціативному ЗП.
6. Як здійснюється запис і зчитування інформації в стековому ЗП?
7. Охарактеризуйте можливі варіанти побудови блочної пам'яті.
8. Які можливості щодо скорочення часу доступу до інформації надає блочна організація пам'яті?
9. Чим обумовлена ефективність розшарування пам'яті?

10. Чим відрізняються сторінковий, швидкий сторінковий і пакетний режими доступу до пам'яті?

11. Яку функцію виконує ЗЕ динамічної пам'яті і як він функціонує?

12. Яку мінімальну кількість ліній повинен містити стовпець ІМС пам'яті?

13. Поясніть призначення управляючих сигналів у мікросхемі пам'яті.

14. Як функціонує динамічний ОЗП?

15. Чим обумовлена необхідність регенерації вмісту динамічних ОЗП?

16. Охарактеризуйте основні сфери застосування статичних і динамічних ОЗП.

17. Дайте порівняльну характеристику різних типів асинхронних ОЗП.

18. Який вплив на асинхронний режим роботи пам'яті надає синхронний характер роботи контролера пам'яті?

19. У чому основні відмінні риси SDRAM архітектури SDR і DDR?

20. У чому полягає основа архітектури Rumbus-пам'яті (RDRAM)?

21. Поясніть призначення і логіку роботи кеш-пам'яті.

22. Якими засобами забезпечується віртуалізація пам'яті?

23. Яка частина віртуальної адреси залишається незмінною у разі його перетворення у фізичну адресу?

24. Чим обумовлена необхідність захисту пам'яті?

25. Назвіть основні елементи блок-схеми НЖМД.

26. Як функціонує “вінчестер”?

27. Як виконується керування даними у “вінчестері”?

28. Опишіть технологію запису і читання інформації у “вінчестері”.

29. Дайте характеристику основних засобів захисту “вінчестера”.

30. Охарактеризуйте основні тенденції розвитку запам’ятовуючих пристроїв комп’ютерів.

4. АРХІТЕКТУРА ПРОЦЕСОРІВ

4.1. ПРИЗНАЧЕННЯ ТА КЛАСИФІКАЦІЯ ПРОЦЕСОРІВ

Процесор – пристрій, що здійснює процес автоматичної обробки даних і програмне управління цим процесом. Процесор дешифрує й виконує команди програми, організовує звертання до оперативної пам'яті, в потрібних випадках ініціює роботу каналів вводу/виводу і периферійних пристроїв, приймає та обробляє запити переривання, які надходять з пристроїв комп'ютера і ззовні. За виконуваними функціями процесор є центральним пристроєм ОМ. Постійне прагнення до підвищення швидкодії ОМ призвело до створення великої різноманітності процесорів, які відрізняються за своєю структурою, призначенням, за способом організації обчислювального процесу та ін. Весь парк процесорів можна класифікувати, наприклад, за такими ознаками:

За використовуваною системою числення:

- процесори, що працюють у позиційній системі;
- процесори, що працюють у непозиційній системі (наприклад, у системі залишкових класів – СЗК).

За способом обробки розрядів:

- з паралельною обробкою розрядів;
- з послідовною обробкою;
- із змішаною обробкою (послідовно-паралельною).

За складом операцій:

- процесори загального призначення;
- проблемно-орієнтовані;
- спеціалізовані.

За місцем процесора в системі:

- центральний процесор (ЦП);
- співпроцесор;
- периферійний процесор;
- канальний процесор (контролер каналу вводу/виводу);
- процесорний елемент (ПЕ) багатопроцесорної системи.

За організацією операційного пристрою:

- з операційним пристроєм процедурного типу (І-процесори, М-процесори);
- процесори з блоковим операційним пристроєм;
- процесори з конвеєрним операційним пристроєм (з арифметичним конвеєром).

За організацією обробки адрес:

- із загальним операційним пристроєм;
- із спеціальним (адресним) операційним пристроєм.

За типом операндів:

- скалярний процесор;
- векторний процесор;
- з можливістю обробки і скалярних, і векторних даних.

За логікою управління процесором:

- з жорсткою логікою управління;
- з мікропрограмним управлінням.

За складом (повнотою) системи команд:

- RISC;
- CISC.

За організацією управління потоком команд / способом заванаження виконавчих пристроїв:

- з послідовною обробкою команд;
- з конвеєром команд;
- суперскалярні процесори;
- процесори з довгим командним словом (VLIW) і так далі.

Як всяка класифікація, приведена вище класифікація не може вважатися повною, оскільки кількість типів процесорів достатньо велика і за своєю архітектурою процесори дуже різноманітні.

4.2. ПРИНЦИПИ ПОБУДОВИ ЕЛЕМЕНТАРНОГО ПРОЦЕСОРА

Вище було відзначено, що конкретні типи ОМ містять у своєму складі процесори, побудовані за різними схемами, і процесори великих ЕОМ істотно відрізняються від процесорів міні- і мікро-ЕОМ (про супер-ЕОМ і говорити не доводиться).

Проте основні принципи побудови процесорів загалом однакові, причому найнаочніше їх можна продемонструвати на прикладі простого мікропроцесора.

Раніше розглядалися дії над числами (додавання, віднімання, множення), представленими в різній формі. Було підкреслено, що всі ці дії здійснюються за допомогою *елементарних* операцій, що виконуються в певній *послідовності*.

До таких елементарних операцій відносяться:

- запис числа в регістр;
- інвертування вмісту розрядів регістра;
- пересилка вмісту регістрів;
- зсув вмісту регістра;
- додавання кодів;

- порозрядні логічні операції або аналіз розрядів;
- операція лічби С+1 або С-1 (інкремент або декремент).

Приклад.

Операція множення реалізується з допомогою:

- аналізу розряду множника;
- підсумовування;
- зсуву.

Всі ці дії виконуються в пристрої, що називається процесором, який складається з двох пристроїв – *операційного* (ОПр) і *управляючого* (ПУ).

ОПр – виконує вказані елементарні операції.

ПУ – управляє ОПр, задаючи необхідну послідовність виконання цих операцій.

Узагальнена структура будь-якого процесора зображена на рис. 4.1.

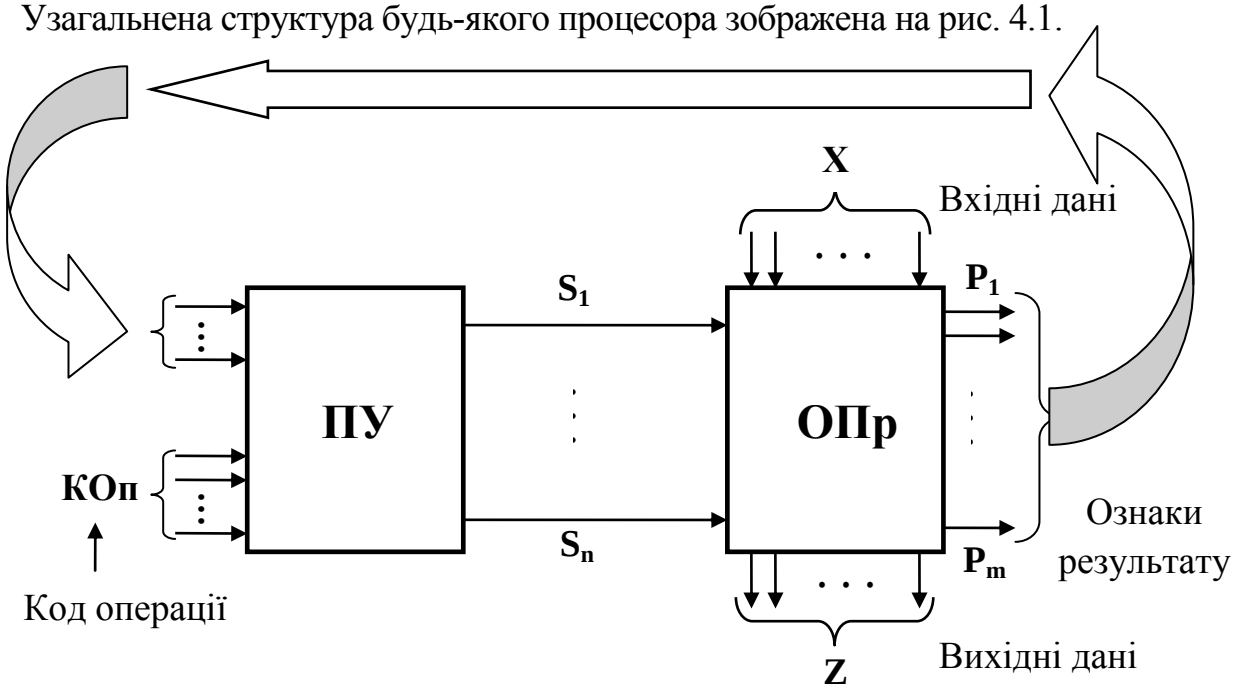


Рис. 4.1. Узагальнена структура процессора

Як вузли ПУ і ОПр включають регістри, лічильники, суматори, мультиплексори, дешифратори і так далі. Крім того, нормальне функціонування процесора і всієї ЕОМ можливо тільки за наявності високостабільних імпульсних послідовностей, які формуються, як правило, з однієї імпульсної послідовності, що виробляється кварцевим генератором. Ці *тактові* імпульсні послідовності синхронізують роботу вузлів процесора, а іноді і всієї ЕОМ.

Кожна елементарна операція, що виконується в одному з вузлів ОПр протягом одного тактового періоду, називається *мікрооперацією.*

У певні тактові періоди одночасно можуть виконуватися декілька мікрооперацій, наприклад: Рг ← 0, Лч ← (Лч) – 1 і так далі. Така сукупність несуперечливих мікрооперацій називається *мікрокомандою*, а набір

мікрокоманд, призначений для вирішення завдання, називається *мікропрограмою*.

Якщо в ОПр передбачена можливість виконання *n* різних мікрооперацій, то з ПУ повинно виходити *n* управляючих ланцюгів S1, ..., Sn, кожний з яких відповідає своїй мікрооперації. Через те що ПУ визначає мікропрограму, тобто які і в якій тимчасовій послідовності повинні виконуватися мікрооперації, він отримав назву *мікропрограмного автомата*. Відповідно ОПр часто називають *операційним автоматом*.

Формування управляючих сигналів S1, ..., Sn може залежати як від зовнішніх сигналів КОп (команди асемблера), так і від стану вузлів ОПр, які визначаються повідомними сигналами ознак стану P1, ..., Pm, що поступають з виходу ОПр на відповідні входи ПУ.

Як уже наголошувалося, ОПр виконує над початковими даними різні арифметичні і логічні операції, тому ОПр найчастіше називають *арифметико-логічним пристроєм* (АЛП).

Ділення будь-якого процесора на програмний і операційний автомати достатньо очевидно і не викликає особливих труднощів у розумінні. Проте структурні схеми навіть простих реальних процесорів крім АЛП і ПУ містять ще ряд вузлів (регістри, лічильники, дешифратори та ін.), які начебто не відносяться ні до АЛП, ні до ПУ.

Для усунення плутанини в подальшому матеріалі необхідно зробити ряд зауважень:

1. У абсолютній більшості випадків пристрої обробки цифрової інформації мають багаторівневу структуру.

Це означає, що ПУ і ОПр можуть самі розпадатися на пари ПУ' і ОП'р, які, у свою чергу, також можуть розпадатися на відповідні ПУ і ОПр. Все залежить від ступеня деталізації розгляду даного цифрового пристрою. Цей *принцип багаторівневості* справедливий для всіх пристроїв ЕОМ.

Більше того, ці міркування справедливі в цілому для ЕОМ, яку можна розкласти на ряд *віртуальних* (що здаються) машин і з кожною працювати на відповідному рівні. У загальному випадку сучасні універсальні ЕОМ мають шість рівнів:

- рівень проблемно-орієнтованої мови;
- процедурно-орієнтована мова;
- асемблерний рівень (мова асемблера);
- рівень операційної системи (мова операційної системи);
- традиційний машинний рівень (мова машинних команд);
- мікропрограмний рівень (мова мікрокоманд).

Машинні мови двох нижніх рівнів є цифровими, і програми на них складаються з довгих числових послідовностей, дуже незручних для людини,

але зрозумілих машині. Все більш високі рівні містять слова і абревіатуру, що зручніше для людини.

2. Із сказаного виходить, що тільки найпростіші процесори мають один рівень і можуть бути в чистому вигляді розкладені на ПУ і ОПр, що складаються з комбінаційних логічних схем, здатних виконувати елементарні арифметико-логічні операції.

3. В даний час немає строгого визначення АЛП, що викликає деяку плутанину при користуванні різною літературою. АЛП зазвичай позначають так, як показано на рис. 4.2. При цьому одні автори мають на увазі під АЛП тільки комбінаційні логічні схеми, здатні виконувати операції двійкового підсумовування (тобто фактично двійковий суматор), інші – цілий комплекс схем для виконання арифметико-логічних операцій.

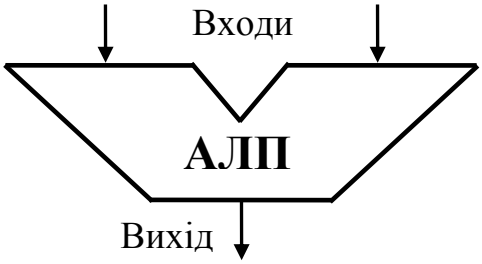


Рис. 4.2. Графічне позначення простого АЛП

4. У загальному випадку поняття мікрооперації і мікропрограми *відносні* і вимагають конкретизації рівня розгляду процесора, оскільки один такт верхнього рівня може включати декілька тактів нижнього рівня.

5. Під час вивчення основних принципів побудови елементарних процесорів вважатимемо:

- процесор має один рівень;
- процесор користується однією тактовою послідовністю;
- значок АЛП (див. рис. 4.2) позначає комплекс комбінаційних схем, здатних виконувати двійкове додавання, зсув двійкового числа, прості порозрядні логічні операції;
- вузли мікропроцесора, що не відносяться безпосередньо до схеми управління, вважатимемо допоміжними вузлами АЛП, або, точніше, вузлами, що забезпечують нормальне функціонування АЛП.

4.3. ПРИСТРІЙ УПРАВЛІННЯ

4.3.1. Функції центрального пристрою управління

Пристрій управління (ПУ) обчислювальної машини реалізує функції управління ходом обчислювального процесу, забезпечуючи автоматичне виконання команд програми. Процес виконання програми в ОМ є послідовністю машинних циклів. Деталізуємо основні цільові функції

управління, що реалізуються пристроєм у ході типового машинного циклу [25]. Для простоти приймемо, що ОМ забезпечує одноадресну систему команд. При цьому, зокрема, вважається, що до початку виконання двохоперандної арифметичної команди другий операнд уже знаходиться в процесорі.

Першим етапом у машинному циклі є *вибірка команди* з пам'яті (етап ВК). Цільову функцію цього етапу позначатимемо як ЦФ-ВК.

За вибіркою команди йде етап декодування її операційної частини (коду операції). Для простоти поки розглядатимемо цей етап як складову частину етапу ВК.

Друга цільова функція – *формування адреси наступної команди*. На це виділяється спеціальний такт роботи – етап ФАНК, якому відповідає цільова функція ЦФ-ФАНК.

Далі слідує етап *формування виконавчої адреси операнда* або адреси переходу (етап ФВА), на якому ПУ реалізує функцію ЦФ-ФВА. Функція має стільки модифікацій, скільки способів адресації (СА) передбачено в системі команд ОМ.

На четвертому етапі реалізується цільова функція *вибірки операнда* (ЦФ-ВО) з пам'яті за виконавчою адресою, сформованою на попередньому етапі.

Нарешті, на останньому етапі машинного циклу дії задаються цільовою функцією *виконання операції* – ЦФ-ВОп. Очевидно, що кількість модифікацій ЦФ-ВОп дорівнює кількості операцій, наявних в системі команд ОМ.

Порядок проходження цільових функцій повністю визначає динаміку роботи пристрою управління і всієї ОМ у цілому.

4.3.2. Структура пристрою управління

Як уже наголошувалося раніше, процес функціонування ОМ складається з послідовності елементарних дій в її вузлах. Такі елементарні перетворення інформації, виконувані протягом одного такту сигналів синхронізації, називаються *мікрооперaціями* (МО). Сукупність сигналів управління, що викликають одночасно виконувані мікрооперації, утворює *мікрокоманду* (МК). У свою чергу, послідовність мікрокоманд, що визначає зміст і порядок реалізації машинного циклу, прийнято називати *мікропрограмою*. Сигнали управління виробляються пристроєм управління, а точніше одним з його вузлів – *мікропрограмним автоматом* (МПА). Назва відображає те, що МПА визначає мікропрограму як послідовність виконання мікрооперацій.

Мікропрограми реалізації перерахованих раніше цільових функцій ініціюються *задаючим обладнанням*, яке виробляє необхідну послідовність сигналів управління і входить до складу управляючої частини ПУ.

Виконуються мікропрограми *виконавчим обладнанням*, що входить до складу основної пам'яті (для ЦФ-ВК і ЦФ-ВО) і операційного пристрою (для

ЦФ-ВОп). Виконавчим обладнанням для цільових функцій ЦФ-ФАНК, ЦФ-ФВА служить адресна частина пристрою управління. В узагальненій структурі ПУ (рис. 4.3) можна виділити дві частини: управляючу і адресну.

Управляюча частина ПУ призначена для координування роботи операційного блока ОМ, адресної частини пристрою управління, основної пам'яті та інших вузлів ОМ.

Адресна частина ПУ забезпечує формування адрес команд і виконавчих адрес операндів в основній пам'яті.

До складу управляючої частини ПУ входять:

- регістр команди (РгК), що складається з адресної (Адреса) і операційної (КОп, СА) частин;
- мікропрограмний автомат (МПА);
- вузол переривань і пріоритетів (ВПП).

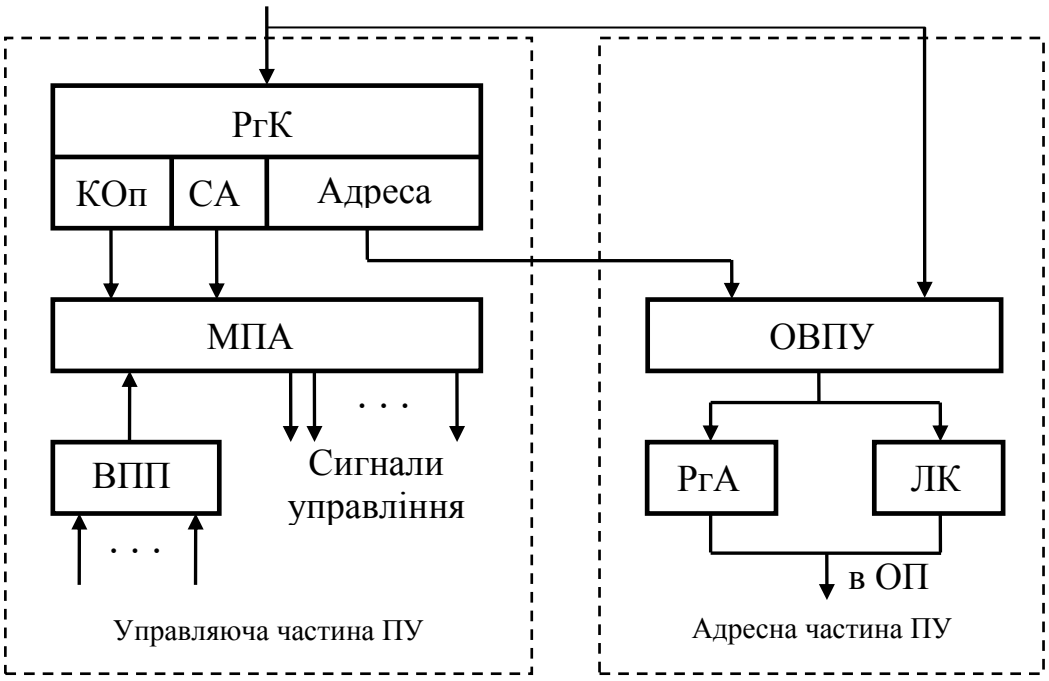


Рис. 4.3. Узагальнена структура пристрою управління

Регістр команди РгК призначений для прийому чергової команди з запам'ятовуючого пристрою. Мікропрограмний автомат на підставі результатів розшифровки операційної частини команди (КОп – код операції, СА – спосіб адресації) виробляє певну послідовність мікрокоманд, що викликають виконання всіх цільових функцій ПУ.

Залежно від способу формування мікрокоманд розрізняють *мікропрограмні автомати*:

- з жорсткою або апаратною логікою;
- з програмованою логікою.

Вузол переривань і пріоритетів дозволяє реагувати на різні ситуації, пов'язані як з виконанням робочих програм, так і із станом ОМ.

Адресна частина ПУ включає:

- операційний вузол пристрою управління (ОВПУ);
- регістр адреси (РгА);
- лічильник команд (ЛК).

Регістр адреси використовується для зберігання виконавчих адрес операндів, а *лічильник команд* – для вироблення і зберігання адрес команд. Вміст РгА і ЛК посилається в регістр адреси основної пам'яті (ОП) для вибірки операндів і команд відповідно.

ОВПУ, що називається інакше вузлом індексної арифметики або вузлом адресної арифметики, обробляє адресні частини команд, формуючи виконавчі адреси операндів, а також готує адресу наступної команди під час виконання команд переходу. Склад ОВПУ може бути аналогічний складу основного операційного пристрою ОМ (іноді в простих ОМ з метою економії витрат на обладнання ОВПУ поєднується з основним операційним пристроєм).

Пристрій управління центрального процесора має зазвичай яскраво виражену блокову архітектуру. В його складі можуть бути:

- блок управління пам'яттю (БУП);
- блок управління каналами (БУК);
- вузол організації прямого доступу до пам'яті.
- блок захисту пам'яті (БЗП);
- блок синхронізації (БС);
- блок таймування (БТ);
- блок зовнішніх зв'язків (БЗЗ, комплексування ЕОМ на рівні процесорів).

Вузол організації прямого доступу до пам'яті зазвичай реалізується у вигляді самостійного пристрою – контролера прямого доступу до пам'яті (КПДП). КПДП забезпечує поєднання в часі роботи операційного пристрою з процесом обміну інформацією між ОП і іншими пристроями ОМ, тим самим підвищуючи загальну продуктивність машини.

Досить часто регістри різних вузлів ПУ об'єднують в окремий вузол управляючих (спеціальних) регістрів пристрою управління.

Всю множину технологій, використовуваних під час реалізації мікропрограмних автоматів пристроїв управління, можна звести до двох категорій:

- МПА з «жорсткою» логікою або апаратурною реалізацією;
- МПА з програмованою логікою.

4.3.3. Мікропрограмний автомат з «жорсткою» логікою

ПУ, побудовані на жорсткій логіці, історично з'явилися першими.

Основною перевагою таких ПУ є їх швидкодія. Саме тому абсолютна більшість спеціалізованих процесорів, особливо призначених для обробки

інформації в режимі реального часу, мають ПУ на жорсткій логіці. Під спеціалізованими розуміються процесори, призначені для виконання вузького набору спеціальних функцій (обробка сигналів радіолокаційних станцій, перетворення Фур'є, матричні операції, обробка сигналів у швидкісних лініях зв'язку і так далі) з максимальною швидкістю.

Проте і в процесорах загального призначення з універсальними наборами команд ПУ на жорсткій логіці також використовуються дуже широко, особливо, як уже наголошувалося, для управління виконанням простих команд. Системи команд таких процесорів завжди фіксовані і не можуть бути змінені користу-вачем. Подібні ПУ іноді називають спеціалізованими. Спеціалізовані ПУ формують незмінні послідовності сигналів управління (СУ) блоком логічних схем.

Блок логічних схем складається з комбінаційних схем, регістрів, лічильників, дешифраторів та інших пристроїв, які виконують функції запам'ятовування поточного стану автомата, що визначає СУ, і формування наступного стану відповідно до вхідних ознак.

Мікропрограма в такому автоматі зберігається за рахунок системи жорстких зв'язків між вузлами ПУ. Для зміни мікропрограми потрібний демонтаж жорстких зв'язків і створення нової схеми.

Типова структура мікропрограмного автомата з жорсткою логікою управління показана на рис. 4.4.

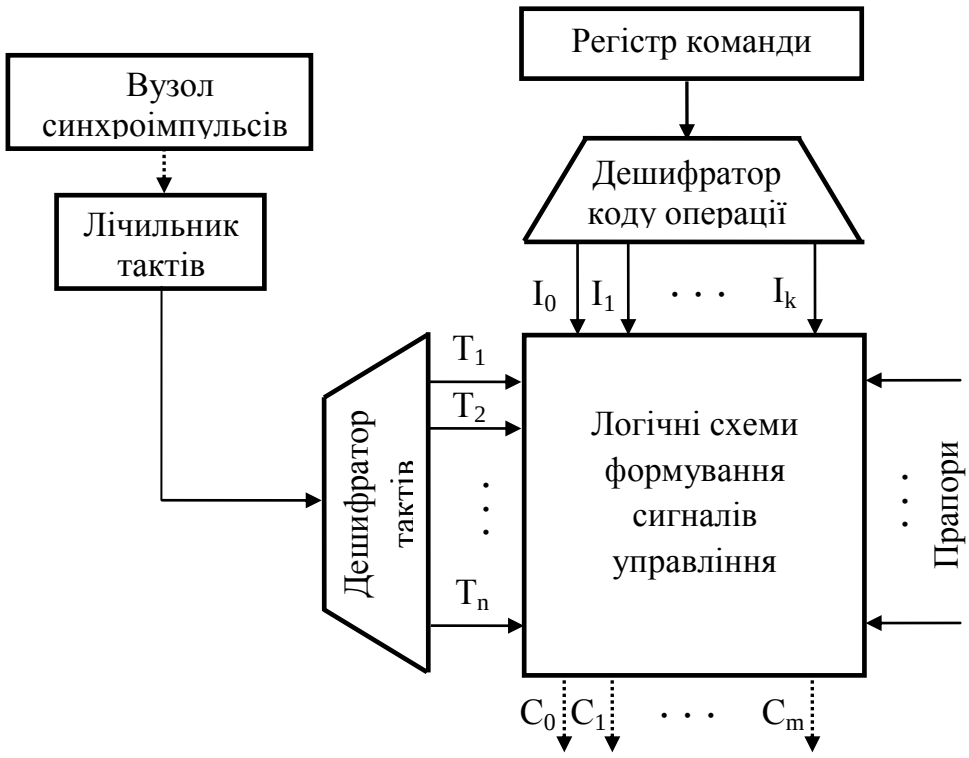


Рис. 4.4. Мікропрограмний автомат з жорсткою логікою

Початковою інформацією для ПУ служать: вміст регістра команди, прапори, тактові імпульси і сигнали управління.

Код операції, що зберігається в РгК, використовується для визначення того, які СУ і в якій послідовності повинні формуватися, при цьому, з метою спрощення логіки управління, бажано мати в ПУ окремий логічний сигнал для кожного коду операції (I_0, I_1, ..., I_k). Це може бути реалізовано за допомогою дешифратора. Дешифратор коду операції перетворить код *j*-ї операції, що поступає з регістра команди (РгК), в одиничний сигнал на *j*-му виході.

Машинний цикл виконання будь-якої команди складається з декількох тактів.

Сигнали управління, по яких виконується кожна мікрооперація, повинні вироблятися в строго певні моменти часу, тому всі СУ «прив'язані» до імпульсів синхронізації (СІ), що формуються вузлом синхроімпульсів. Період СІ повинен бути достатнім для того, щоб сигнали встигли розповсюдитися по трактах даних та інших ланцюгах. Кожен СУ асоціюється з одним з тактових періодів у рамках машинного циклу. Формування сигналів, що відзначають початок чергового тактового періоду, покладається на синхронізатор. Синхронізатор містить лічильник тактів, що здійснює підрахунок СІ. Вузол синхроімпульсів після завершення чергового такту роботи додає до вмісту лічильника тактів одиницю. До виходів лічильника підключений дешифратор тактів, з якого і знімаються сигнали тактових періодів: T_1, .., T_n. В *i*-му стані лічильника тактів, тобто під час *i*-го такту, дешифратор тактів виробляє одиничний сигнал на своєму *i*-му виході. У разі такої організації в ПУ повинен бути передбачений зворотний зв'язок, за допомогою якого після закінчення циклу команди лічильник тактів знову встановлюється в стан T_1.

Додатковим чинником, що впливає на послідовність формування СУ, є стан інформаційних сигналів (прапорів), що відображають хід обчислень, і сигнали з шини управління. Ця інформація також поступає на вхід ПУ, причому кожна лінія тут розглядається незалежно від інших.

Процес синтезу схеми МПА з жорсткою логікою називається структурним синтезом і розділяється на такі етапи:

- вибір типу логічних запам'ятовуючих елементів;
- кодування станів автомата;
- синтез комбінаційної схеми, що формує вихідні сигнали.

Щоб визначити спосіб реалізації МПА з жорсткою логікою, необхідно описати внутрішню логіку ПУ, що формує вихідні сигнали управління, як булеву функцію вхідних сигналів. Канонічний метод структурного синтезу МПА був запропонований В. М. Глушковим. Згідно з цим методом, завдання синтезу автомата зводиться до синтезу комбінаційної схеми шляхом побудови системи логічних функцій з подальшою їх мінімізацією.

Принцип побудови логічних схем формування управляючих сигналів пояснюється на рис. 4.5. Тут показаний фрагмент схеми, що забезпечує

вироблення управляючого сигналу C_k, в *i*-му і *S*-му тактах виконання команди з кодом операції *j*, причому сигнал C_k з'являється в *i*-му такті тільки у разі значень повідомчих сигналів $x_1=1$ і $x_3=1$, а в *S*-му такті завжди.

Таким чином, назва «жорстка логіка» обумовлена тим, що кожній мікропрограмі тут відповідає свій набір логічних схем з фіксованими зв'язками між ними. Під час реалізації простої системи команд вузли МПА з жорсткою логікою економічні і дозволяють забезпечити найбільшу швидкодію зі всіх можливих методів побудови МПА. Проте із зростанням складності системи команд відповідно ускладнюються і схеми автоматів з жорсткою логікою, внаслідок чого зменшується їх швидкодія.

Другим недоліком МПА з жорсткою логікою є те, що будь-які зміни або модифікації команд універсального процесора, що вимагають зміни мікропрограм, приведуть до зміни структури управляючого автомата, а отже, і топології його внутрішніх зв'язків. Під час виробництва спеціалізованих процесорів потрібна дуже широка номенклатура ПУ (по числу вирішуваних завдань) при відносно невеликій потребі в кожному конкретному типі. З погляду технології мікроелектронного виробництва процесорів у вигляді ВІС і НВІС вказаний недолік є досить істотним. Збільшується ціна кожного випущеного кристала процесора за рахунок збільшення витрат на розробку нових топологій ПУ і відладку технології їх виробництва.

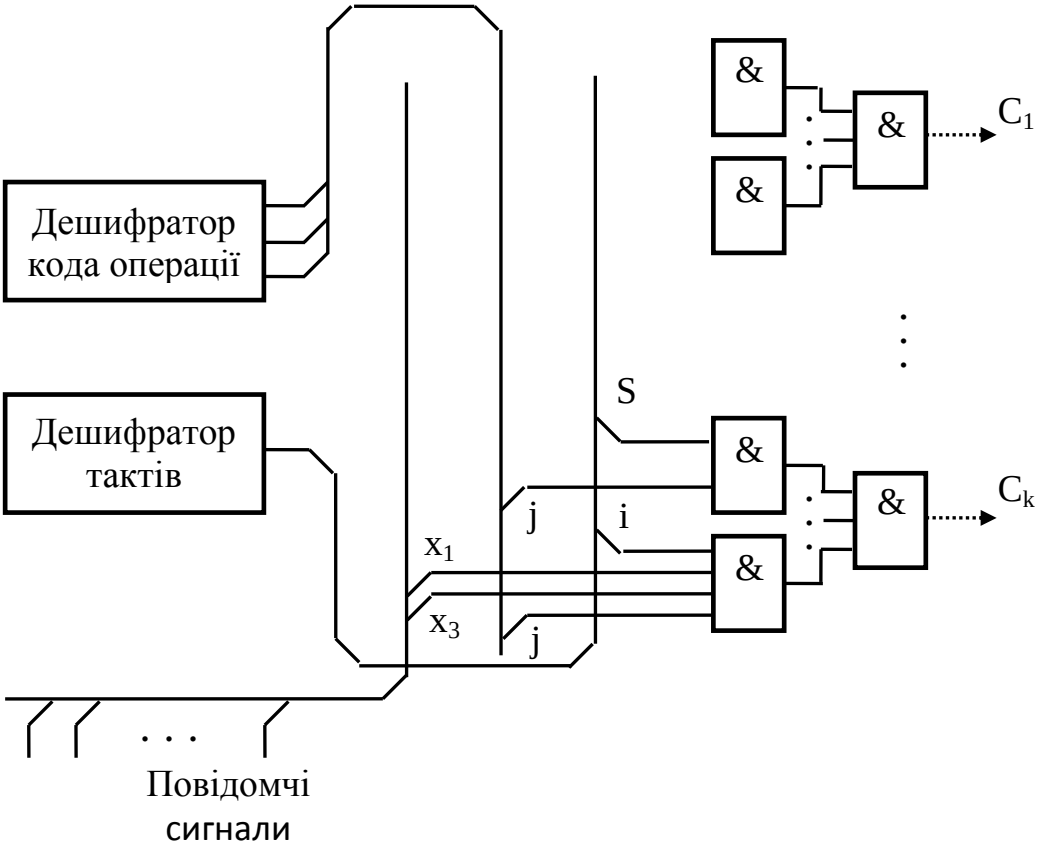


Рис. 4.5. Фрагмент схеми формування сигналів управління

Оптимальним вирішенням цієї проблеми з'явилася побудова ПУ на спеціалізованих логічних структурах з фіксованою топологією – програмованих логічних матрицях (ПЛМ). ПЛМ є шаруватою структурою, в кожному шарі якої зосереджені однотипні логічні елементи. Топологія зв'язків побудована таким чином, що на входи *кожного* елемента подальшого шару подаються вихідні сигнали *всіх* елементів попереднього шару. ПЛМ може виконуватися як окрема ВІС, так і формуватися усередині кристала процесора, будучи достатньо зручним елементом для створення управляючих автоматів.

Програмування матриці полягає в усуненні непотрібних зв'язків за допомогою фотошаблонів або випалюванням (подібно до того, як це робиться в ПЗП).

Програмуючи ПЛМ, можна реалізувати потрібні системи булевих функцій.

Це дозволяє будувати автомати складної управляючої структури. Через свою складність ПУ, як правило, описується великою кількістю булевих функцій багатьох змінних. Ці змінні, у свою чергу, часто бувають залежними, тому виявляється необхідною *сумісна* мінімізація системи булевих функцій, які реалізуються ПЛМ.

Розглянуте вище ілюструє тільки саму ідею побудови ПЛМ. Структура ж ВІС, що реально випускаються, достатньо різноманітна. Для побудови управляючих автоматів найбільш зручні ВІС, що містять разом з ПЛМ набір вихідних тригерів.

Наступним поколінням пристроїв типу ПЛМ є ПЛІС – програмовані логічні інтегральні схеми, що дозволяють програмно скомпонувати в одному корпусі електронну схему, еквівалентну схемі, що включає від декількох десятків до декількох сотень ІС стандартної логіки.

В даний час на світовому ринку домінують дві великі групи ПЛІС – EPLD і FPGA.

EPLD – багаторазово програмовані (для збереження конфігурації використовується ППЗП з ультрафіолетовим стиранням).

FPGA – багаторазово реконфігуровані (для збереження конфігурації використовується статичний ОЗП).

Фірми-виробники поставляють також повне інструментальне забезпечення для розробки і застосування пристроїв на базі EPLD і FPGA за допомогою персональних комп'ютерів.

4.3.4. Мікропрограмний автомат з програмованою логікою

В основі ідеї мікропрограмного автомата з програмованою логікою лежить той факт, що для ініціації будь-якої мікрооперації досить сформувати відповідний СУ на відповідній лінії управління, тобто перевести таку лінію в активний стан. Це може бути подано за допомогою двійкових цифр 1 (активний стан – є СУ) і 0 (пасивний стан – немає СУ). Для вказівки мікрооперацій, що виконуються в даному такті, можна сформувати управляюче слово, в якому

кожен біт відповідає одній лінії, що управляє. Таке управляюче слово називають *мікрокомандою* (МК). Таким чином, мікрокоманда може бути представлена управляючим словом зі своєю комбінацією нулів і одиниць. Послідовність мікрокоманд, що реалізують певний етап машинного циклу, утворює *мікропрограму*. В термінології англійською мовою мікропрограму часто називають *firmware*, підкреслюючи той факт, що це щось середнє між апаратурою (*hardware*) і програмним забезпеченням (*software*). Мікропрограми для кожної команди ОМ і для кожного етапу циклу команди розміщуються в спеціальному ЗП, що називається *пам'яттю мікропрограм* (ПМП). Процес формування СУ можна реалізувати, послідовно (з кожним тактовим імпульсом) витягуючи мікрокоманди мікропрограми з пам'яті і інтерпретуючи інформацію, що міститься в них, про сигнали управління.

Відмітною особливістю мікропрограмного автомата з програмованою логікою є зберігання мікрокоманд у вигляді кодів в спеціалізованому запам'ятовуючому пристрої – пам'яті мікропрограм. Кожній команді ОМ у цьому ЗП в явній формі відповідає мікропрограма, тому часто пристрої управління, до складу яких входить мікропрограмний автомат з програмованою логікою, називають мікропрограмними.

Типова структура мікропрограмного автомата зображена на рис. 4.6.

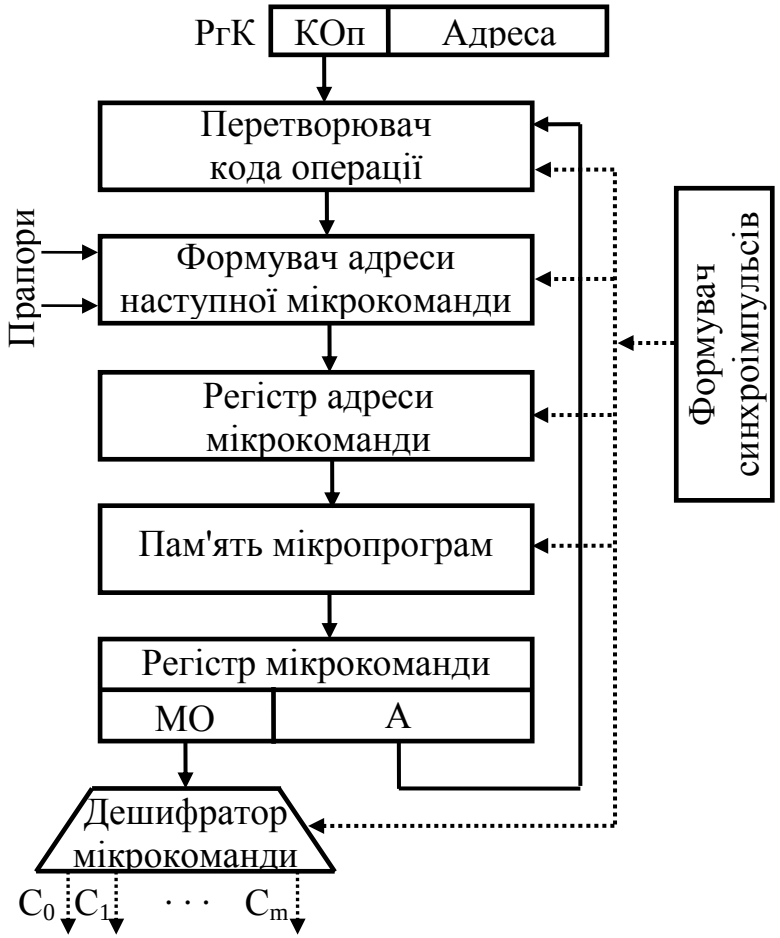


Рис. 4.6. Мікропрограмний автомат з програмованою логікою

У складі вузла присутні: пам'ять мікропрограм (ПМП), регистр адреси мікрокоманди (РгАМ), регістр мікрокоманди (РгМК), дешифратор мікрокоманд (ДшМК), перетворювач коду операції, формувач адреси наступної мікрокоманди (ФАНМ), формувач синхроімпульсів (ФСІ).

Запуск мікропрограми виконання операції здійснюється шляхом передачі коду операції з РгК на вхід перетворювача, в якому код операції перетвориться в початкову (першу) адресу мікропрограми A_n. Ця адреса поступає через ФАНМ в регістр адреси мікрокоманди. Вибрана за адресою A_n з ПМП мікрокоманда зано-ситься в РгМК. Кожна мікрокоманда в загальному випадку містить мікрооперaційну (МО) і адресну (А) частини.

Мікрооперaційна частина мікрокоманди поступає на дешифратор мікрокоманди, на виході якого утворюються управляючі сигнали C_i, які ініціюють виконання мікрооперaцій у виконавчих пристроях і вузлах ОМ. Адресна частина мікрокоманди подається у ФАНМ, де формується адреса наступної мікрокоманди $A_{мк}$. Ця адреса може залежати від адреси на виході перетворювача коду операції A_n, адресної частини поточної мікрокоманди A і значень інформаційних сигналів (прапорів) X, які поступають від виконавчих пристроїв. Сформована адреса мікрокоманди знову записується в РгАМ, і процес повторюється до закінчення мікропрограми.

Розрядність адресної (R_A) і мікрооперaційної (R_{MO}) частин мікрокоманди визначаються із співвідношень

$$R_A = \text{int}(\log_2 N_{MK}); \quad (4.1)$$

$$R_{MO} = \text{int}(\log_2 N_{CУ}), \quad (4.2)$$

де N_{MK} - загальна кількість мікрокоманд; $N_{CУ}$ - загальна кількість формованих сигналів управління.

У свою чергу, необхідна ємкість пам'яті мікропрограм дорівнює

$$E_{ПМП} = N_{MK}(R_A + R_{MO}) = N_{MK}(\text{int}(\log_2 N_{MK}) + \text{int}(\log_2 N_{CУ})).$$

Інформація про те, які сигнали управління повинні бути сформовані в процесі виконання поточної МК, в закодованому вигляді міститься в мікрооперaційній частині (МО) мікрокоманди. Спосіб кодування мікрооперaцій багато в чому визначає складність апаратних засобів пристрою управління і його швидкісні характеристики.

4.3.5. Кодування мікрокоманд

Вживані в мікрокомандах варіанти кодування сигналів управління можна звести до трьох груп: мінімальне кодування (горизонтальне мікропрог-рамування), максимальне кодування (вертикальне мікропрограмування) і групове кодування (змішане мікропрограмування). Структури мікропрограмних

автоматів у ході різних способів кодування мікрооперацій показані на рис. 4.7, 4.8 [25].

Під час горизонтального мікропрограмування (рис. 4.7, *а*) під кожен сигнал управління в мікрооперaційній частині мікрокоманди виділений один розряд ($R_{МО} = N_{СУ}$). Це дозволяє в рамках однієї мікрокоманди формувати будь-які поєднання СУ, чим забезпечується максимальний паралелізм виконання мікрооперацій. Крім того, відсутня необхідність у декодуванні МО і виходи регістра мікрокоманди можуть бути безпосередньо підключені до відповідних керованих точок ОМ. Широкому розповсюдженню горизонтального мікропрограмування проте перешкоджають великі витрати на зберігання мікрооперaційних частин мікрокоманд ($E_{МО} = N_{МК} \times N_{СУ}$), причому ефективність використання ПМП виходить низькою, оскільки при великому числі мікрооперацій в кожній окремій МК реалізується лише одна або декілька з них, тобто переважна частина розрядів МО містить нулі.

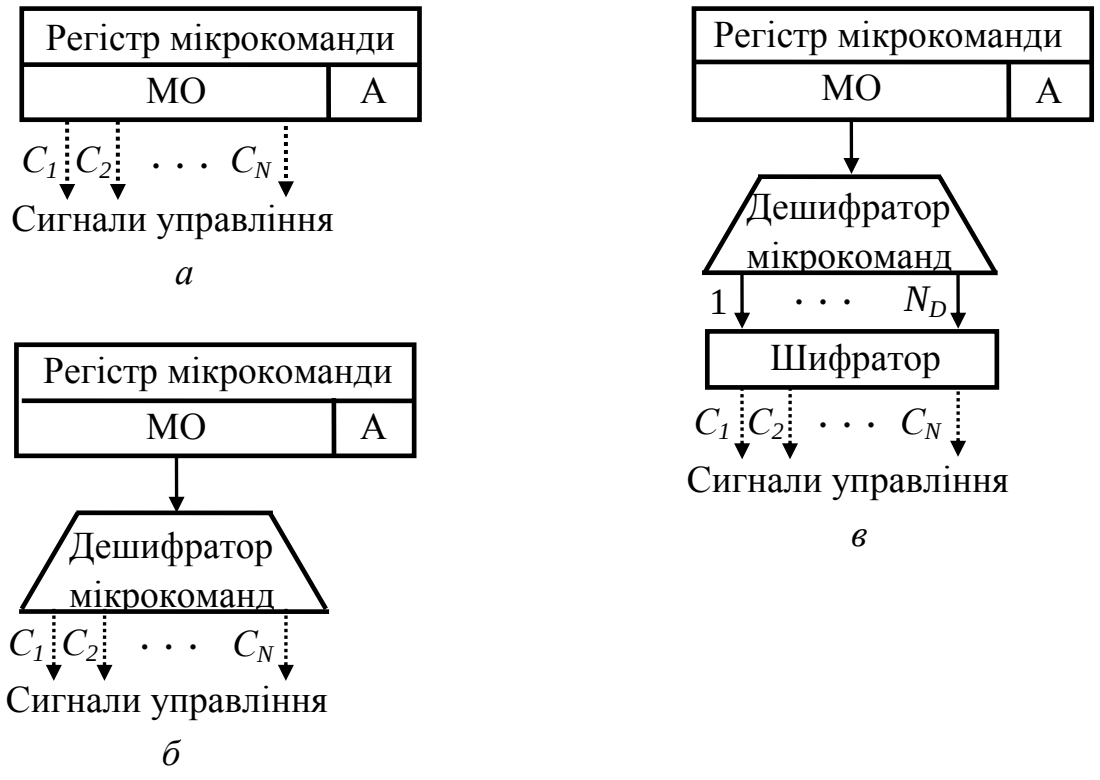


Рис.4.7. Структури МПА у ході різних способів кодування мікрооперацій:
а – мінімального; *б* – максимального;
в – максимального з шифратором

Під час максимального (вертикального) кодування (рис. 4.7, *б*) кожній мікрооперації привласнюється певний код, наприклад, її порядковий номер у повному списку можливих мікрооперацій. Цей код і заноситься в МО. Мікрооперaційна частина МК має мінімальну довжину, яка визначається як двійковий логарифм від числа управляючих сигналів (мікрооперацій) за

формулою (4.2). Такий спосіб кодування вимагає мінімальних апаратних витрат у ПМП на зберігання мікрокоманд, проте виникає необхідність у дешифраторі ДшМК, який повинен перетворити код мікрооперації у відповідний сигнал управління. У разі великої кількості СУ дешифратор вносить значну тимчасову затримку, а головне – в кожній МК указується лише один сигнал управління, який ініціює тільки одну мікрооперацію, за рахунок чого збільшуються довжина мікропрограми і час її реалізації.

Останній недолік усувається у разі підключення до виходів ДшМК шифратора (Ш) (рис. 4.7, *в*), що приводить до збільшення кількості СУ, які формуються одночасно. Природно, що крім кодів окремих мікрооперацій повинні бути передбачені коди, що представляють визначені комбінації мікрооперацій. Для підвищення універсальності шифратор доцільно будувати на базі запам'ятовуючого пристрою.

Варіант, представлений на рис. 4.7, *в*, раціональний, якщо

$$N_D << N_{MK} \;\; i \;\; R_{MO} = \text{int}(\log_2 N_D) < N_{CУ},$$

де N_D - кількість виходів дешифратора.

За цих умов апаратні витрати на зберігання мікрооперацiйних частин МК відносно малі:

$$E_{MO} = N_{MK} \times R_{MO} = N_{MK} \times \text{int}(\log_2 N_D).$$

Проте повна ємкість використовуваної пам'яті дорівнює

$$E_{MO} + E_{Ш} = N_{MK} \times \text{int}(\log_2 N_D) + N_D \times N_{CУ},$$

звідки ясно, що при близьких N_{MK} *і* N_D варіант втрачає сенс.

Мінімальне і максимальне кодування є двома крайніми точками широкого спектра можливих рішень задачі кодування СУ. Проміжне положення займає групове або змішане кодування.

Тут усі сигнали управління (мікрооперації) розбиваються на K груп

$$Y = \bigcup_{l=1}^{K} Y_l.$$

Залежно від принципу розбиття мікрооперацій на групи розрізняють горизонтально-вертикальне і вертикально-горизонтальне кодування (рис. 4.8).

В горизонтально-вертикальному методі (рис. 4.8, а) в кожну групу включаються взаємно несумісні сигнали управління (мікрооперації), тобто СУ, які ніколи не зустрічаються разом в одній мікрокоманді. При цьому сигнали, що зазвичай формуються в одному і тому ж такті, опиняються в різних групах. Усередині кожної групи сигнали управління кодуються максимальним (вертикальним) способом, а групи – мінімальним (горизонтальним) способом.

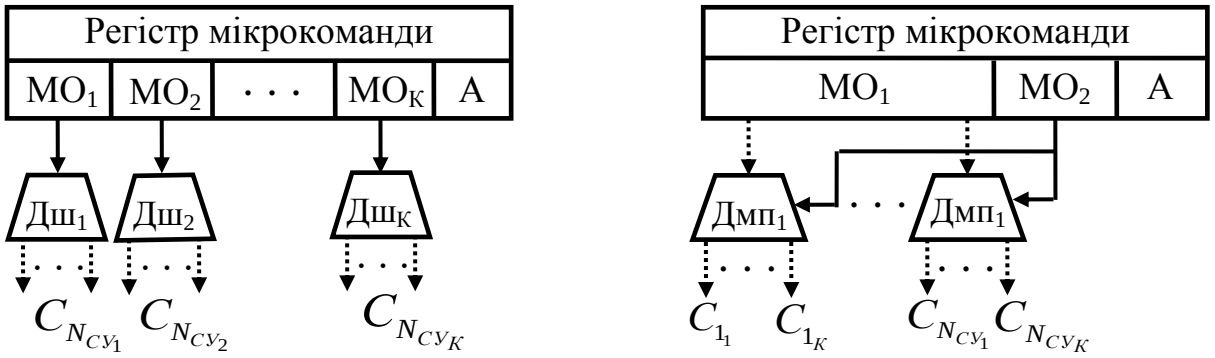


Рис.4.8. Структури МПА у ході різних способів кодування мікрооперацій:
а – горизонтально-вертикального; *б* – вертикально-горизонтального

Кожній групі Y_l виділяється окреме поле в мікрооперацiйній частині МК, загальна розрядність якої дорівнює

$$R_{MO} = \sum_{l=1}^{K} R_{MO_l} = \sum_{l=1}^{K} \text{int}(\log_2 N_{CУ_l}),$$

де $N_{CУ_l}$ – кількість СУ, які подаються l–$м$ полем (групою).

Загальна ємність пам'яті мікропрограм розглянутого варіанту кодування визначається з виразу

$$E_{ПМП} = N_{МК}(R_{MO} + R_A) = N_{МК}(\sum_{l=1}^{K} \text{int}(\log_2 N_{CУ_l}) + \text{int}(\log_2 N_{МК})).$$

У ході вертикально-горизонтального способу (рис. 4.8, *б*) вся множина сигналів управління (мікрооперацій) також ділиться на декілька груп, однак у групу включаються сигнали управління (мікрооперації), що найчастіше зустрічаються разом в одному такті. Поле МО ділиться на дві частини: $МО_1$ і $МО_2$. Поле $МО_1$, довжина якого дорівнює максимальній кількості сигналів управління (мікрооперацій) в групі, кодується горизонтально, а поле $МО_2$, що вказує на приналежність до певної групи, – вертикально. Із зміною групи міняються і керовані точки, куди повинні бути направлені сигнали управління з кожної позиції $МО_1$. Це досягається за допомогою демультиплексорів (Дмп), керованих кодом номера групи з поля $МО_2$.

У ході горизонтального, вертикального і горизонтально-вертикального способів кодування мікрооперацій кожне поле мікрокоманди несе фіксовані функції, тобто має місце *пряме кодування*. Під час *непрямого кодування* одне з полів відводиться для інтерпретації інших полів. Прикладом непрямого кодування мікрооперацій може служити вертикально-горизонтальне кодування.

Іноді використовується дворівневе кодування мікрооперацій. На першому рівні з вертикальним кодуванням вибирається мікрокоманда, поле МО якої є

адресою горизонтальної мікрокоманди другого рівня – *нанокоманди*. Даний спосіб поєднання вертикального і горизонтального мікропрограмування часто називають *нанопрограмуванням*. Метод припускає дворівневу систему кодування мікрооперацій і, відповідно, дворівневу організацію управляючої пам'яті.

Верхній рівень управління утворюють мікропрограми, що зберігаються в пам'яті мікропрограм. У мікрокомандах використовується вертикальне кодування мікрооперацій. Кожній мікрокоманді відповідає нанокоманда, що зберігається в управляючій пам'яті нижнього рівня – пам'яті нанокоманд. У нанокомандах застосовано горизонтальне кодування мікрооперацій. Саме нанокоманди використовуються для безпосереднього формування сигналів управління. Мікрокоманди замість закодованого номера СУ містять адресне посилання на відповідну нанокоманду.

Такий підхід, зберігаючи всі переваги горизонтального мікропрограмування, дозволяє значно скоротити сумарну ємність управляючої пам'яті. Річ у тому, що число різних поєднань сигналів управління, а отже, кількість «довгих» нанокоманд, зазвичай невелике. В той же час практично всі мікрокоманди багато разів повторюються в різних мікропрограмах, і заміна в мікрокомандах «довгого» горизонтального управляючого поля, на коротке адресне посилання дає відчутний ефект.

4.3.6. Адресація мікрокоманд

Переважно мікрокоманди в мікропрограмі виконуються послідовно, проте в загальному випадку черговість мікрооперацій не є фіксованою. З цієї причини в ПУ необхідно передбачити ефективну систему реалізації переходів. Переходи, як безумовні, так і умовні, є невід'ємною частиною будь-якої мікропрограми.

Під час виконання мікропрограми адреса чергової мікрокоманди відноситься до однієї з трьох категорій:

- визначається кодом операції команди;
- є наступною по порядку адресою;
- є адресою переходу.

Перший випадок має місце тільки один раз у кожному циклі ко-манди, відразу ж услід за її вибіркою. Як уже наголошувалося раніше, кожній команді з системи команд ОМ відповідає «своя» мікропрограма в пам'яті мікропрограм, тому перша дія, яку потрібно провести після вибірки команди, – перетворити код операції в адресу першої МК відповідної мікропрограми. Це може бути виконано за допомогою апаратного перетворювача коду операції (див. рис. 4.6). Такий перетворювач зазвичай реалізується у вигляді спеціального ЗП, такого, що зберігає початкові адреси мікропрограм в ПМП. Для вказівки того, як

повинна обчис-люватися адреса наступної МК, в мікрокоманді може бути виділене спеціальне однобитовое поле. Одиниця в цьому полі означає, що $A_{МК}$ повинна бути сформована на підставі коду операції. Подібна МК зазвичай розташовується в кінці мікропрограми етапу вибірки або в мікропрограмі аналізу операції.

Подальша черговість виконання мікрокоманд мікропрограми може бути задана шляхом вказівки в кожній МК адреси наступної мікрокоманди (примусова адресація) або шляхом автоматичного збільшення на одиницю адреси поточної МК (природна адресація) [7].

В обох випадках необхідно передбачити ситуацію, коли адреса наступної мікрокоманди залежить від стану інформаційних сигналів – *ситуацію переходу*. Для вказівки того, яка умова повинна бути перевірена, в МК вводиться поле умови переходу (УП). Поле УП визначає номер *i* інформаційного сигналу x_i, значення якого аналізується під час формування адреси наступної мікрокоманди. Якщо поле УП = 0, то ніякі умови не перевіряються і $A_{МК}$ або береться з адресної частини мікрокоманди (у разі примусової адресації), або формується шляхом додавання одиниці до адреси поточної мікрокоманди. Коли УП ≠ 0, то $A_{МК} = A + x_i$. У результаті цього здійснюється умовний перехід: якщо $x_i = 0$ – до мікрокоманди з адресою $A_{МК} = A$, а коли $x_i = 1$ – до мікрокоманди з адресою $A_{МК} = A + 1$. Описаний порядок формування адреси показаний на рис. 4.9 і має місце у разі примусової адресації з однією адресою.

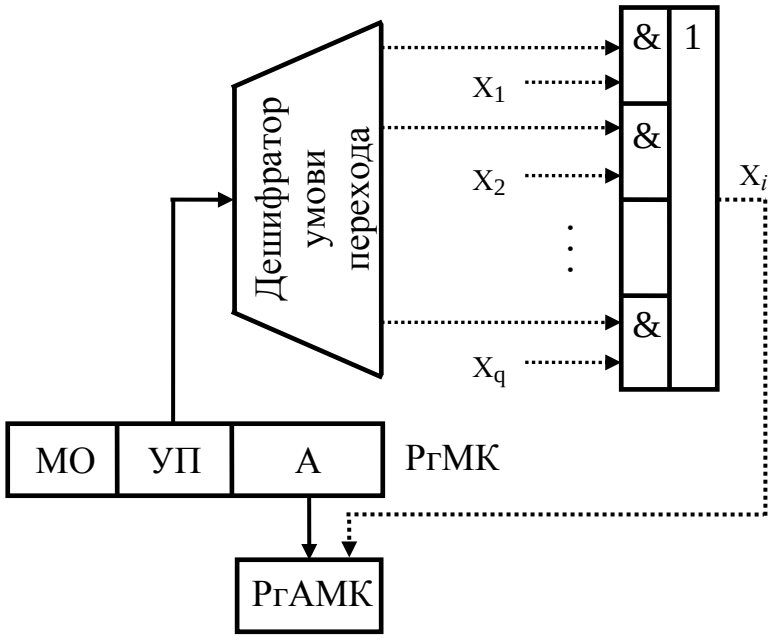


Рис. 4.9. Формування адреси у разі примусової адресації

Розглянутий спосіб дозволяє в кожній команді врахувати стан тільки одного з інформаційних сигналів. Гнучкіший підхід реалізований у ряді ОМ фірми IBM. Адреси мікрокоманди розбиваються на дві складові. Старші *n* розрядів зазвичай залишаються незмінними. В процесі виконання

мікрокоманди ці розряди просто копіюються з адресної частини МК в аналогічні позиції РгАМ, визначаючи блок з 2^n мікрокоманд у пам'яті мікропрограм. Останні (молодші) k розрядів РгАМ встановлюються в 1 або 0 залежно від того, перевірка яких інформаційних сигналів була задана в полі УП і в якому стані ці сигнали знаходяться. Такий метод дозволяє в одній мікрокоманді сформувати 2^k варіантів переходу.

На відміну від варіанта, показаного на рис. 4.9, можливий і інший підхід, у чомусь близький природній адресації, коли в адресній частині задається лише адреса можливого переходу (рис. 4.10). Адреса чергової МК формується шляхом додавання одиниці до адреси поточної мікрокоманди.

Головна перевага примусової адресації – висока універсальність і швидкодія. Тут зміна ділянки мікропрограми не зачіпає решти мікрокоманд, а поєднання в одній МК умовного переходу з формуванням сигналів управління зменшує загальний час виконання мікропрограми.

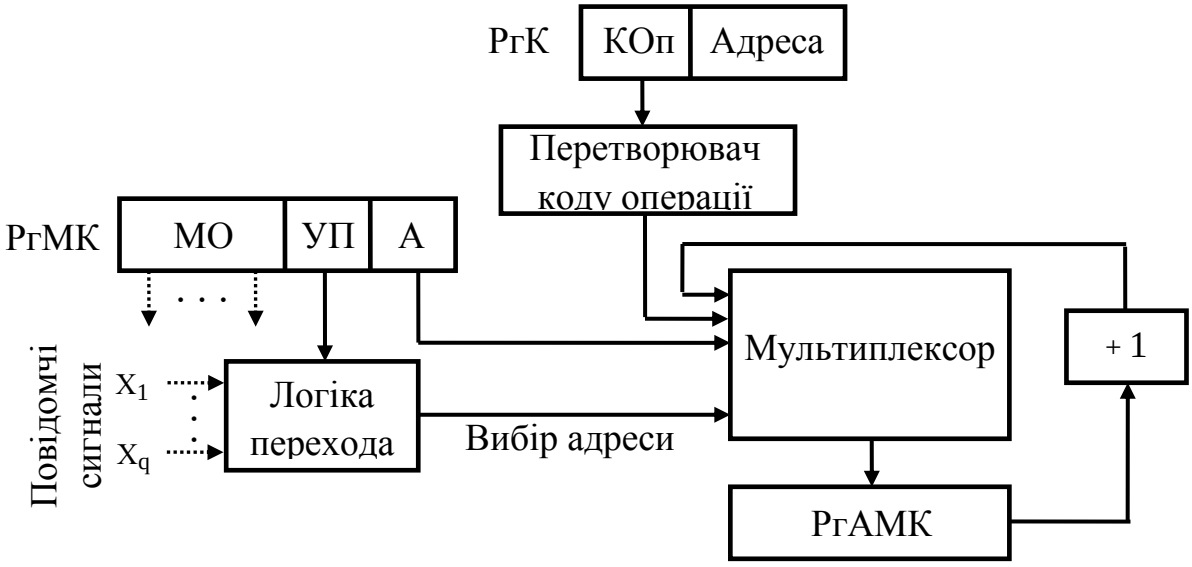


Рис. 4.10. Логіка управління переходом

Основний недолік примусової адресації – підвищення вимоги до ємності пам'яті для зберігання адрес МК:

$$E_A = N_{МК} (R_A + R_{УП}) = N_{МК} (\text{int}(\log_2 N_{МК}) + \text{int}(\log_2 L)),$$

де $R_{УП}$ – розрядність поля УП; L – загальна кількість повідомчих сигналів.

У разі природної адресації відпадає необхідність у введенні адресної частини в кожну МК. Мається на увазі, що мікрокоманди проходять у природному порядку і процес адресації реалізується лічильником адреси мікрокоманди (ЛчАМ). Для реалізації умовних і безумовних переходів використовуються спеціальні мікрокоманди, які складаються тільки з двох полів: адресного поля B і поля УП, яке визначає номер умовного переходу.

Таким чином, у разі природної адресації повинні застосовуватися МК двох типів: управляючі і операційні. Операційна мікрокоманда містить тільки

мікрооперaційну частину і не має адресної частини. Тип МК задається її першим розрядом: якщо *МК*(1) = 1, то це управляюча мікрокоманда.

Структура МПА з природною адресацією показана на рис. 4.11.

Видача сигналів управління $C_1, ..., C_m$ стробується сигналом $\overline{МК(1)}$, який приймає одиничне значення під час виконання операційної МК. Дешифратор умови переходу стробується сигналом *МК*(1), який рівний 1 під час обробки управляючої мікрокоманди. Адреса наступної МК утворюється на лічильнику ЛчАМ під час виконання мікрооперації +1ЛчАМ ЛчАМ = ЛчАМ + 1 або $П_2$ЛчАМ ЛчАМ = В.

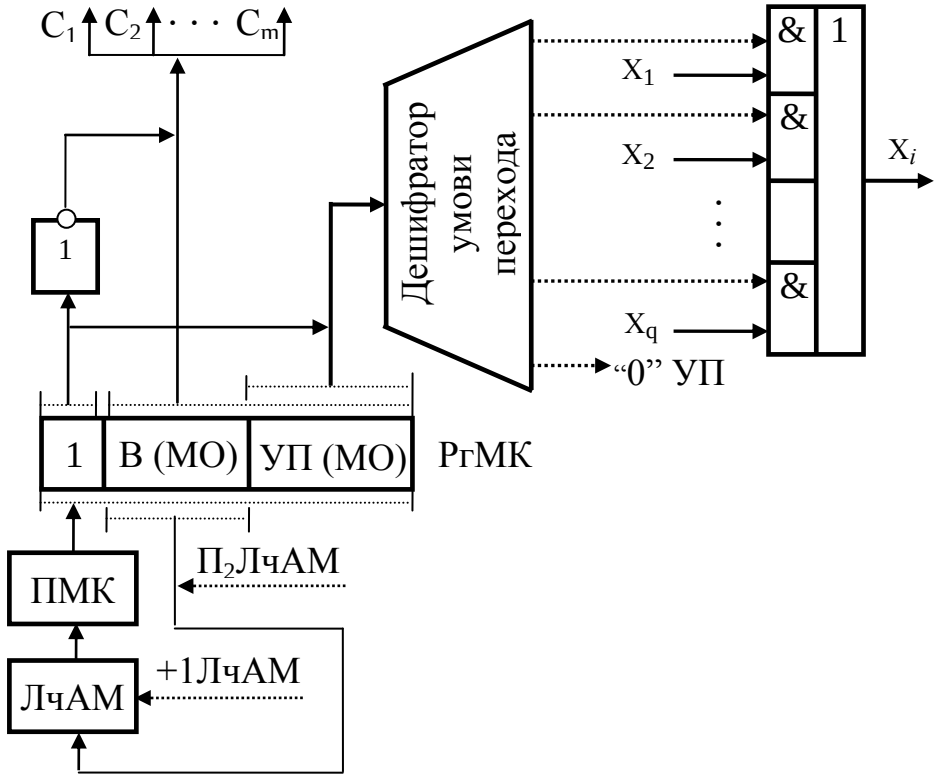


Рис. 4.11. Мікропрограмний автомат з природною адресацією

Перевага природної адресації – економія пам'яті мікропрограм, а основний недолік полягає в тому, що для будь-якого переходу потрібний повний тактовий період, тоді як при примусовій адресації перехід виконується одночасно з формуванням управляючих сигналів без додаткових звернень до управляючої пам'яті. Крім того, у разі сильно розгалужених мікропрограм потрібні великі додаткові витрати пам'яті.

Розглянуті способи адресації апаратно реалізує формувач адреси наступної мікрокоманди ФАНМ (рис. 4.6). ФАНМ є механізмом управління послідовністю виконання мікрокоманд.

4.4. ОПЕРАЦІЙНІ ПРИСТРОЇ

4.4.1. Логічна організація процесорів загального призначення

Раніше було показано, що різні арифметичні операції над числами (представленими, окрім, того в різному кодуванні) вимагають істотно різних послідовностей мікрооперацій.

Крім того, очевидно, що чим більше функціональніший електронний пристрій, тим складніша його структура (більше елементів) і тим повільніше він працює. З іншого боку, функції такого складного пристрою може виконати набір простіших і швидкодіючих пристроїв, проте апаратурні витрати і ціна будуть вищі.

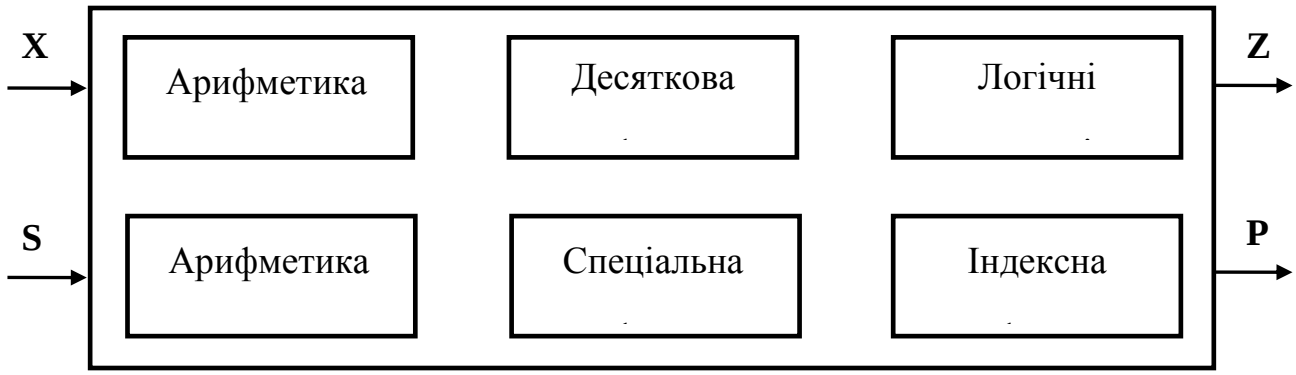
У загальному випадку операції, що виконуються в операційному пристрої (ОПр), можна розділити на такі групи:

- операції двійкової арифметики для чисел з фіксованою комою (ЧФК);
- операції двійкової (шістнадцяткової) арифметики для чисел з плаваючою комою (ЧПК);
- операції десяткової арифметики;
- логічні операції;
- операції індексної арифметики (у разі модифікації адрес команд);
- операції спеціальної арифметики: нормалізація чисел, арифметичний зсув (зсовуються тільки цифрові розряди без знакового), логічний зсув (зсовуються всі розряди) і так далі.

ЕОМ загального призначення зазвичай реалізують операції приведених вище груп, але роблять це порізному, залежно від типу АЛП, використовуваного в процесорі.

АЛП підрозділяються на блокові і багатофункціональні.

У блокових АЛП (рис. 4.12) перераховані групи операцій виконуються в окремих електронних блоках, при цьому підвищується швидкість роботи, оскільки блоки можуть паралельно виконувати відповідні операції.



X – вхідні дані; S – управляючі сигнали; Z – результат операції;
P – повідомлення про завершення роботи

Рис. 4.12. Блокові АЛП

Крім того, спеціалізований блок завжди виконує операції швидше, ніж універсальний переналаштовуваний блок.

Блокові АЛП характерні для великих ЕОМ, де головною є максимальна швидкодія, а не апаратні витрати і вартість. Прості співпроцесори в мікро-ЕОМ, що виконують операції з ЧПЗ, також можна розглядати як спеціалізовані блоки, тому АЛП мікро-ЕОМ із співпроцесорами можна іноді розглядати як блокові.

У багатофункціональних АЛП перераховані групи операцій виконуються одними і тими ж схемами, які комутуються потрібним чином залежно від необхідного режиму роботи. Такі АЛП характерні для міні- і мікро-ЕОМ, побудованих на простих процесорах.

Існують і інші структури АЛП (змішані), що знаходяться десь між блоковими і багатофункціональними.

Слід мати на увазі, що часто ЕОМ, побудовані на базі простих мікропроцесорів, мають АЛП, що дозволяють виконувати тільки операції двійкової арифметики над ЧФЗ і деякими логічними операціями. У цьому випадку решта груп операцій виконується спеціальними підпрограмами, що сильно знижує швидкість їх виконання.

Розглянемо дещо докладніше структуру АЛП простого процесора і визначимо мінімально необхідний набір пристроїв, з яких він складається. З викладеного вище витікає, що до складу такого АЛП повинен входити пристрій, який виконує операції двійкового додавання (суматор). Крім того, для зберігання операндів і результату необхідно мати принаймні три буферні регістри (регістри тимчасового зберігання). Проте в простому випадку результат операції можна записувати в один з регістрів тимчасового зберігання на місце одного з операндів. Цей регістр прийнято називати *акумулятором*, а процесор у цілому – процессором *акумуляторного типу*.

Структурна схема АЛП простого мікропроцесора акумуляторного типу зображена на рис. 4.13.

Акумулятор повинен обов'язково мати двонаправлений зв'язок з внутрішньою шиною даних процесора. Для виконання арифметико-логічних операцій необхідний пристрій, що виконує зсув двійкових чисел (зсовувач). І, нарешті, необхідний регістр, в якому зберігаються деякі ознаки результату виконаної операції, необхідні для функціонування ПУ (регістр ознак).

Вже наголошувалося, що АЛП в цілому і двійковий суматор мають одне позначення. Відповідно до зроблених раніше зауважень регістр тимчасового зберігання і акумулятор можна вважати допоміжними вузлами АЛП.

Операційний пристрій нерозривно пов'язаний з найбільш швидкодіючою пам'яттю ОМ – з локальною регістровою пам'яттю процесора. Виділення регістрів у окремий блок на схемі покликане підкреслити самостійне значення, якого набувають регістри в універсальних процесорах, – вони не просто є

частиною операційних пристроїв, а використовуються для зберігання різної інформації як під час обробки, так і під час вводу-виводу.

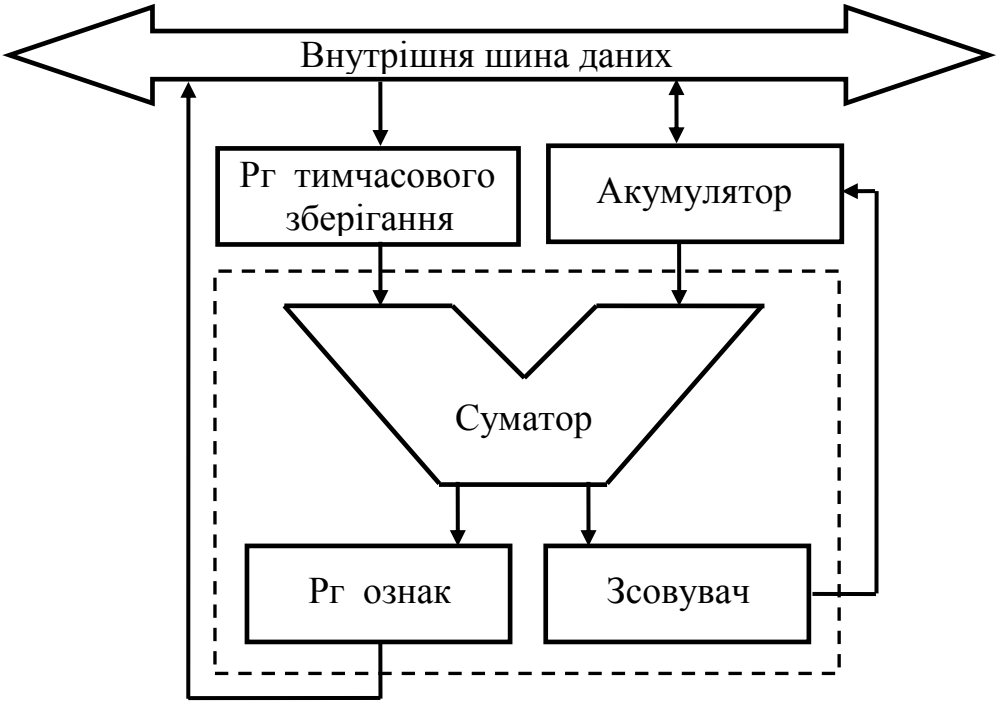


Рис. 4.13. Структурна схема простого АЛП акумуляторного типу

Цілочисельні регістри об'єднуються в блок регістрів загального призначення (РЗП), регістри з плаваючою комою – в окремий блок (у деяких процесорах ці багаторозрядні регістри використовуються і як векторні регістри в спеціальних режимах, в інших векторні регістри винесені в окремий блок). Окрім згаданих регістрів можна виділити набір спеціальних управляючих регістрів, використовуваних для управління режимами роботи процесора, функціонуванням його різних підсистем, управління пам'яттю і так далі.

Операційні пристрої процесорів можуть будуватися з більшим або меншим ступенем універсальності, можуть бути простішими, універсальними, такими, що вимагають великого об'єму мікрокоду для реалізації всіх необхідних алгоритмів операцій, або – складнішими і спеціалізованими, але за рахунок цього – продуктивнішими і такими, що не вимагають великого об'єму управляючого мікрокоду. Перші пристрої можна назвати пристроями процедурного типу, оскільки вони вимагають для реалізації якого-небудь алгоритму арифметичної операції виконання послідовності дій, заданої в часі (тобто процедури). Прикладом пристроїв процедурного типу можуть бути, до деякої міри, пристрої для виконання непрямого множення. Такі пристрої після невеликого доопрацювання можуть бути використані і для реалізації інших операцій (алгоритмів), наприклад, для звичайного додавання із знаком, для виконання ділення або операцій з плаваючою комою. У граничному випадку найбільш універсальною схемою може бути звичайний накопичувальний суматор, доповнений схемами виконання логічних операцій

Пристрої другого типу, які розраховані на апаратну реалізацію алгоритмів обчислень, можна назвати пристроями з жорсткою структурою.

З практичної точки зору найбільший інтерес являють операційні пристрої з жорсткою та магістральною структурами.

4.4.2. Операційні пристрої з жорсткою структурою

В ОПр з жорсткою структурою комбінаційні схеми жорстко розподілені між всіма регістрами. До кожного регістра відноситься свій набір комбінаційних схем, що дозволяють реалізувати певні мікрооперації. Приклад ОПр з жорсткою структурою, що забезпечує виконання операцій типу «додавання», приведений на рис. 4.14.

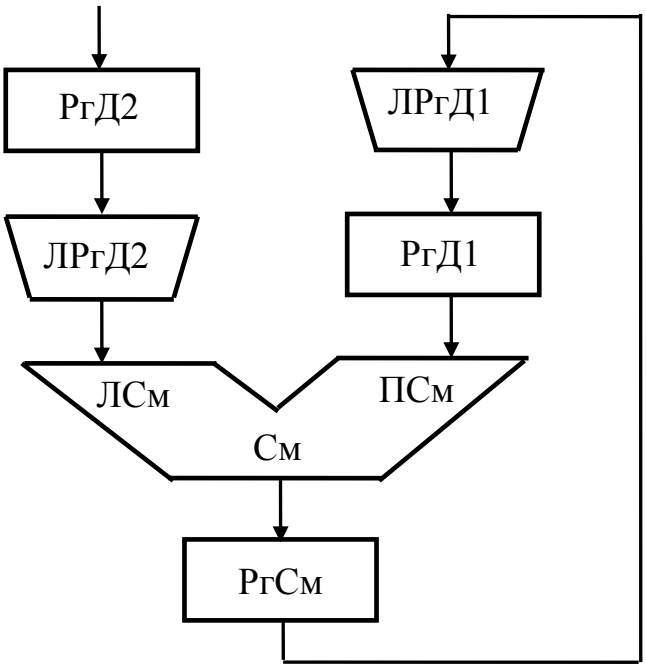


Рис. 4.14. Операційний пристрій із жорсткою структурою

До складу ОПр входять три регістри зі своїми логічними схемами:

- регістр першого доданку РгД1 і схема ЛРгД1;
- регістр другого доданку РгД2 і схема ЛРгД2;
- регістр суми РгСм і схема комбінаційного суматора См.

Логічна схема ЛРгД2 реалізує мікрооперації передачі другого доданка з РгД2 на лівий вхід суматора по відповідному сигналу управління:

- або прямим кодом;
- або інверсним кодом;
- або із зсувом на один розряд вліво.

Логічна схема ЛРгД1 забезпечує передачу результата з регістра РгСм в регістр РгД1 по відповідному сигналу управління:

- або прямим кодом;

- або із зсувом на один розряд вліво;
- або із зсувом на два розряди вправо.

Комбінаційний суматор См призначений для підсумовування (звичайного або по модулю 2) операндів, що поступили на його лівий (ЛСм**)** і правий (ПСм**)** входи. Результат підсумовування заноситься в регістр РгСм.

Моделлю ОПр з жорсткою структурою є так званий *І-автомат***.**

Апаратні витрати на ОПр з жорсткою структурою $C_{ж}$ можна оцінити за виразом

$$C_{ж} = nC_T N + 3\sum_{i=1}^{N} n_i \sum_{j=1}^{K} K_{ij} + \sum_{i=1}^{N} n_i \sum_{j=1}^{K} C_j k_{ij},$$

де N – кількість внутрішніх слів ОПр**;** n_1 ..., n_N – довжини слів; $n = (n_1 + ... + n_N)/N$ – середня довжина слова; k_{ij} – кількість мікрооперацій типу; $j = 1, 2,..., K$ (додавання, зсув, передача і т. п.), використовуваних для обчислень слів з номерами $i = 1, 2, ..., N$; C_T – ціна тригера; C_j – ціна однорозрядної схеми для реалізації мікрооперації j-го типу.

У приведеному виразі перший доданок визначає витрати на зберігання n-розрядних слів, другий – на зв'язки регістрів з комбінаційними схемами, а третій – сумарну вартість комбінаційних схем, що реалізують мікрооперації K типів над N словами.

Витрати часу на виконання операцій типу «додавання» в ОПр з жорсткою структурою дорівнюють

$$T_{ж} = t_B + t_C + t_П ,$$

де t_B – тривалість мікрооперації видачі операндів з регістрів;
t_C – тривалість мікрооперації «додавання»;
$t_П$ – тривалість мікрооперації прийому результата в регістр.

Перевагою ОПр з жорсткою структурою є висока швидкодія, недоліком – мала регулярність структури, що ускладнює реалізацію таких ОПр у вигляді великих інтегральних схем.

4.4.3. Операційні пристрої з магістральною структурою

В ОПр з магістральною структурою всі внутрішні регістри об'єднані в окремий вузол регістрів загального призначення (РЗП), а всі комбінаційні схеми – в операційний блок (ОБ), який часто асоціюють з терміном «арифметико-логічний пристрій».

Операційний блок і вузол регістрів сполучаються між собою з допомогою магістралей – звідси і назва «магістральний ОПр».

Приклад магістрального ОПр зображений на рис. 4.15.

До складу вузла РЗП тут входять *N* регістрів загального призначення, що підключаються до магістралей *A* і *B* через мультиплексори *MUX A* і *MUX B*. Кожен з мультиплексорів є керованим комутатором, що сполучає вихід одного з РЗП з відповідною магістраллю.

Номер регістра, що підключається, визначається адресою *a* або *b*, яка подається на адресні входи мультиплексора з пристрою управління. По магістралях *A* і *B* операнди поступають на входи операційного блока, до яких підключається комбінаційна схема, що реалізує необхідну мікрооперацію (по сигналу управління з ПУ).

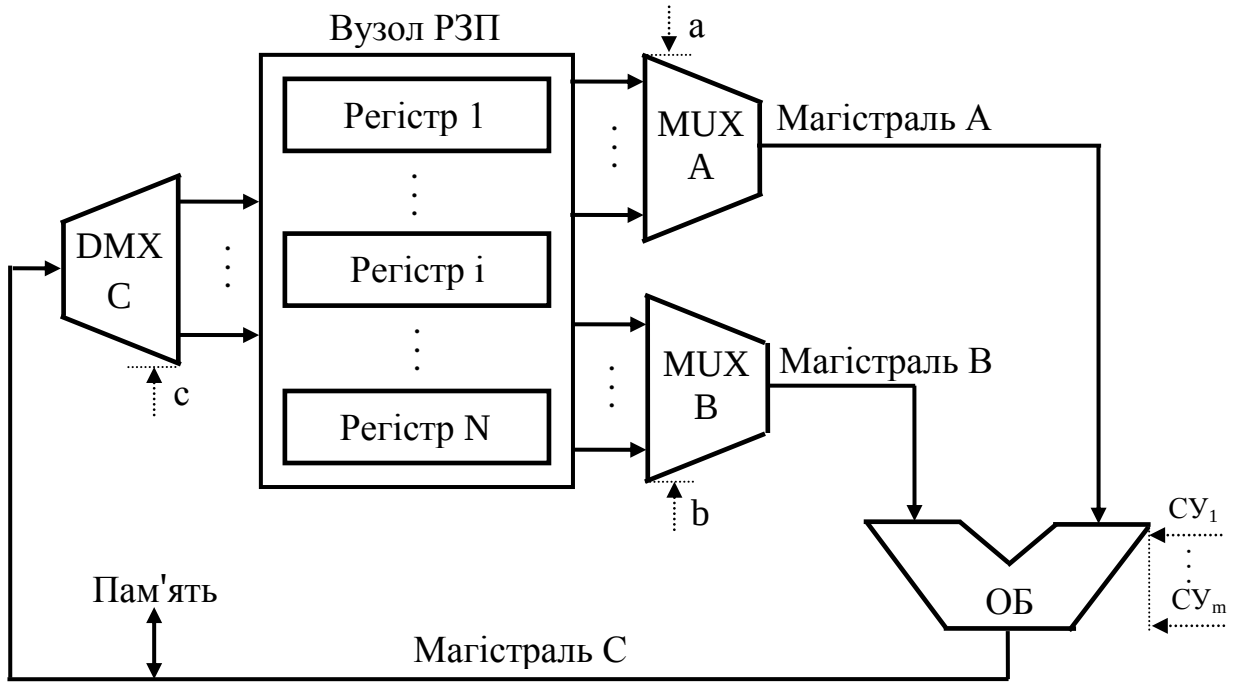


Рис. 4.15. Магістральний операційний пристрій

Таким чином, будь-яка мікрооперація ОБ може бути виконана над вмістом будь-яких регістрів ОПр. Результат мікрооперації по магістралі *C* заноситься через демультиплексор *DMX C* в конкретний регістр вузла РЗП. Демультиплексором є керований комутатор, що має один інформаційний вхід і *N* інформаційних виходів. Вхід підключається до виходу із заданою адресою *c*, яка поступає на адресні входи *DMX C* з ПУ.

Моделлю ОПр з магістральною структурою є *М-автомат. М-автоматом* називається модель ОПр, що створена на основі принципу об'єднання комбінаційних схем і реалізує в кожному такті тільки одну мікрооперацію.

Використовуючи позначення, введені в попередньому розділі, вираз для оцінки апаратних витрат на магістральний ОПр можна записати в такому вигляді:

$$C_M = nC_T N + 3n(N+K) + n\sum_{j=1}^{K} C_j,$$

де перший доданок визначає витрати на N регістрів, другий – витрати на зв'язки вузла РЗП і ОБ, а третій – сумарну ціну ОБ.

Із порівняння виразів для витрат витікає, що магістральна структура економічніша за жорстку структуру, якщо

$$З(N + K - M) < \sum_{i=1}^{N}\sum_{j=1}^{K} C_j K_{ij} - \sum_{j=1}^{K} C_j ,$$

де $M = \sum_{i=1}^{N}\sum_{j=1}^{K} K_{ij}$ кількість мікрооперацій, що реалізовуються ОП з жорсткою структурою.

З урахуванням останньої нерівності можна сформулювати таку сильну умову економічності магістральних структур:

$$M > N + K.$$

Витрати часу на додавання в магістральних ОПр більші, ніж в ОПр з жорсткою структурою:

$$T_M = t_B + t_C + t_П + t_{MUX} + t_{DMX} = t_Ж + t_{MUX} + t_{DMX} ,$$

де t_{MUX} – затримка на підключення операндів з РЗП до ОБ;
t_{DMX} – затримка на підключення результату до РЗП.

Основною перевагою магістральних ОПр є висока універсальність і регулярність структури, що полегшує їх реалізацію на кристалах ІС. Взагалі, магістральна структура ОПр в сучасних ОМ застосовується найчастіше.

4.4.4. Алгоритми виконання арифметичних операцій у цілочисельних операційних пристроях

Для більшості сучасних ОМ загальноприйнятим є такий формат з фіксованою комою (ФК), коли кома фіксується праворуч від молодшого розряду коду числа. З цієї причини відповідні операційні пристрої називають цілочисельними ОПр. У формі з ФК можуть бути представлені як числа без знака, коли все n позицій числа відводяться під значущі цифри, так і зі знаком. У останньому випадку старший (n – 1)-й розряд числа займає знак числа (0 – плюс, 1 – мінус), а під значущі цифри відведені розряди з (n – 2)-го по 0-й. Під час запису негативних чисел використовується доповнюючий код, який для числа N визначається за такою формулою:

$$N_Д = 2^n - N = (2^n - 1) - N + 1 = \overline{N} + 1.$$

Цілочисельний ОПр повинний забезпечувати виконання таких арифметичних операцій над числами без знака і зі знаком: *додавання/віднімання; множення; ділення.*

Додавання і віднімання

На рис. 4.16 наводяться приклади додавання цілих чисел, поданих у доповнюючому коді (нагадаємо, що під час додавання в доповнюючому коді знаковий розряд бере участь в операції нарівні з цифровими).

Під час додавання *n*-розрядных двійкових чисел (біт знака і $n - 1$ значущих цифр) можливий результат, який містить *n* значущих цифр.

Ця ситуація відома як *переповнення*. «Зайвий» біт займає позицію знака, що приводить до некоректності результату. Природно, що ОП повинний виявляти факт переповнення і сигналізувати про нього. Для цього використовується наступне правило: *якщо додаються два числа і вони обидва позитивні або обидва негативні, переповнення має місце тоді і тільки тоді, коли знак результату протилежний знаку доданків*.

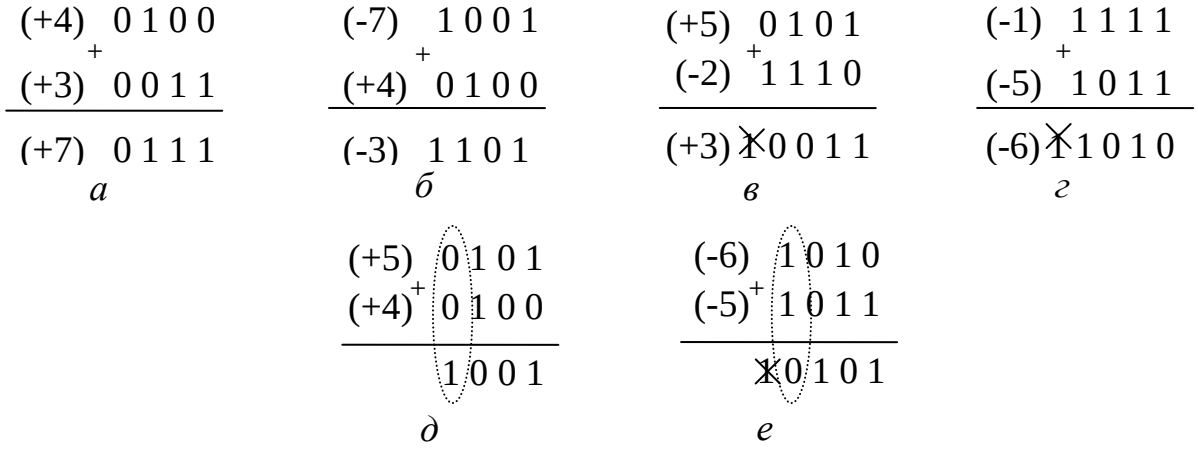


Рис. 4.16. Приклади виконання операції додавання в доповнюючому коді:
а, б, в, г – додавання без виникнення переповнення;
д, е – додавання з переповненням

Рисунки 4.16, *д* і 4.16, *е* показують приклади переповнення. Звернемо увагу, що переповнення не завжди супроводжується перенесенням із знакового розряду.

Віднімання виконується відповідно до правила: *для віднімання одного числа (від'ємника) з іншого (зменшуваного) необхідно взяти доповнення від'ємника і додати його до зменшуваного*. Під доповненням тут розуміється від'ємник з протилежним знаком, поданий у доповнюючому коді. Віднімання ілюструється прикладами (рис. 4.17). Два останні приклади (рис. 4.17, *д* і 4.17, *е*) демонструють раніше розглянуте правило виявлення переповнення.

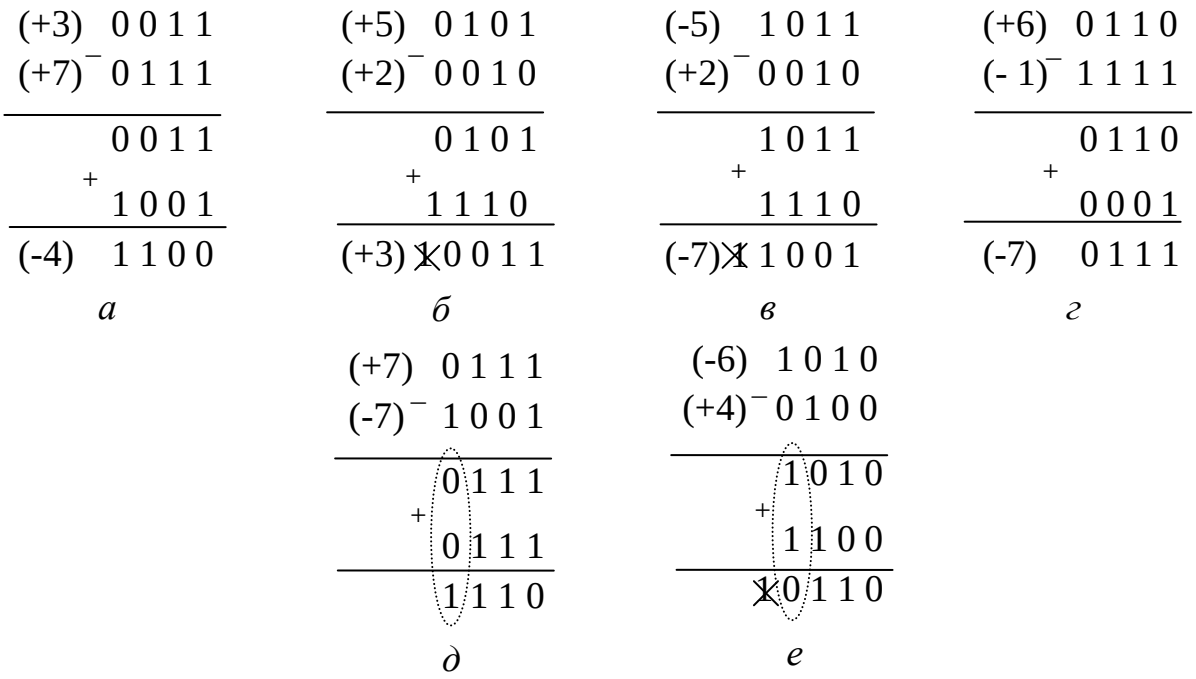


Рис. 4 .17. Приклади виконання операції віднімання в доповнюючому коді:
а, *б, в, г* - віднімання без виникнення переповнення;
д, е - віднімання з переповненням

Щоб спростити виявлення ситуації переповнення, часто застосовується так званий *модифікований доповнюючий код*, коли для зберігання знака відводяться два розряди, причому обидва беруть участь в арифметичній операції нарівні з цифровими розрядами.

В нормальній ситуації обидва знакових розряди містять однакові значення. Відмінність у вмісті знакових розрядів служить ознакою виниклого переповнення (рис. 4.18).

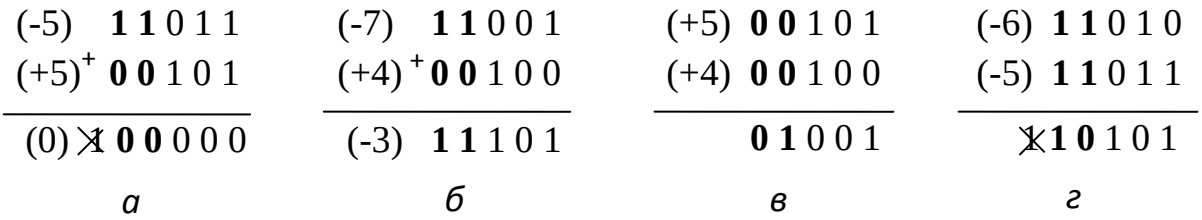


Рис. 4.18. Приклади виконання операції додавання в модифікованому доповнюючому коді: *а, б* - переповнення немає; *в, г* - виникло переповнення

На рис. 4.19 показана можлива структура операційного блока (ОБ) для додавання і віднімання чисел із знаком у форматі з фіксованою комою.

Центральною ланкою пристрою є *n*-разрядный двійковий суматор.

Операнд *A* поступає на вхід суматора без змін. Операнд *B* заздалегідь пропускається через схеми додавання по модулю 2, тому вид коду *B*, що поступає на інший вхід суматора, залежить від виконуваної операції. Якщо задана операція додавання (управляючий код – 0), то результат на виході ОБ визначається виразом $S = A + B$.

Якщо задана операція віднімання (управляючий код – 1), на вхід суматора подаються інверсні значення всіх розрядів B і, крім того, на вхід перенесення в молодший розряд суматора C_{in} поступає 1.

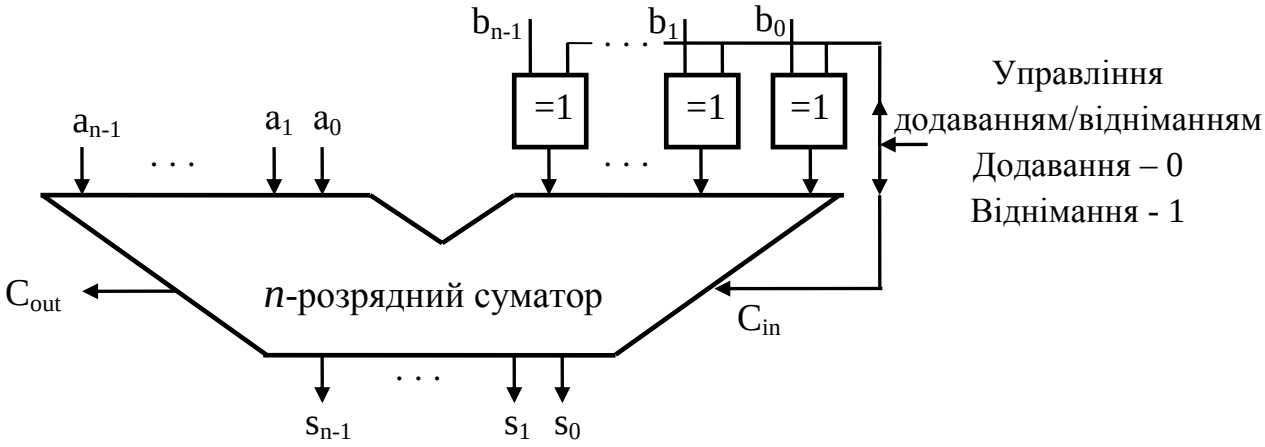


Рис. 4.19. Структура операційного блока для додавання і віднімання

В результаті на виході ОБ буде $S = A + \overline{B} + 1$, що відповідає додаванню до A числа B з протилежним знаком, тобто відніманню.

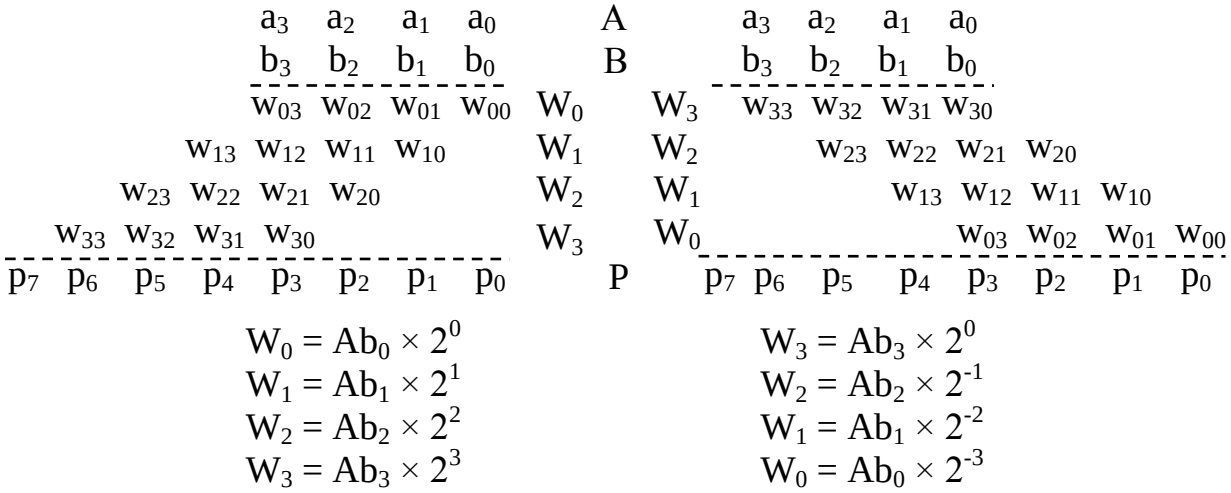
Цілочисельне множення

У порівнянні з додаванням і відніманням, множення – складніша операція як у разі програмного, так і в разі апаратного втілення. В ОМ застосовуються різні алгоритми реалізації операції множення і, відповідно, декілька схем побудови операційних блоків, що забезпечують виконання операції множення.

Традиційна схема множення схожа на відому процедуру запису «в стовпчик». Обчислення добутку P (p_{2n-1} p_{2n-2} **...** p_1 p_0) двох n-розрядних двійкових чисел без знака A (a_{n-1} a_{n-2} **...** a_1 a_0) і B (b_{n-1} b_{n-2} **...** b_1 b_0) зводиться до формування часткових добутків (ЧД) W_i, поодинці на кожну цифру множника, з подальшим підсумовуванням отриманих ЧД. Перед підсумовуванням кожен частковий добуток повинен бути зсунутим на один розряд щодо попереднього згідно з вагою цифри множника, якій цей ЧД відповідає. Оскільки операндами є двійкові числа, обчислення ЧД спрощується – якщо цифра множника b_i дорівнює 0, то W_i теж дорівнює 0, а при $b_i = 1$ частковий добуток дорівнює множеному ($W_i = A$). Множення двох n-розрядних двійкових чисел $P = A \times B$ приводить до отримання результату, що містить $2n$ бітів. Таким чином, алгоритм множення припускає послідовне виконання двох операцій – додавання і зсуву (рис. 4.20).

Підсумовування ЧД зазвичай проводиться не на завершальному етапі, а в міру їх отримання. Це дозволяє уникнути необхідності зберігання всіх ЧД, тобто скорочує апаратні витрати. Згідно з даною схемою пристрій множення

припускає наявність регістрів множеного, множника і суми часткових добутків, а також суматора ЧД і схем зсуву.



$W_0 = Ab_0 \times 2^0$
$W_1 = Ab_1 \times 2^1$
$W_2 = Ab_2 \times 2^2$
$W_3 = Ab_3 \times 2^3$

$W_3 = Ab_3 \times 2^0$
$W_2 = Ab_2 \times 2^{-1}$
$W_1 = Ab_1 \times 2^{-2}$
$W_0 = Ab_0 \times 2^{-3}$

Рис. 4.20. Загальна схема множення із зсувом суми часткових добутків вліво або вправо

Підсумовування ЧД зазвичай проводиться не на завершальному етапі, а в міру їх отримання. Це дозволяє уникнути необхідності зберігання всіх ЧД, тобто скорочує апаратні витрати. Згідно з даною схемою пристрій множення припускає наявність регістрів множеного, множника і суми часткових добутків, а також суматора ЧД і схем зсуву.

Залежно від способу отримання суми часткових добутків (СЧД) можливі чотири варіанти реалізації «традиційної» схеми множення [7]:

1. Множення починається з молодших розрядів множника, і зсув суми часткових добутків відбувається вправо у разі нерухомого множеного.

2. Множення починається із старших розрядів множника, і зсув суми часткових добутків відбувається вліво у разі нерухомого множеного.

3. Множення починається з молодших розрядів множника, і зсув множеного відбувається вліво у разі нерухомої суми часткових добутків.

4. Множення починається із старших розрядів множника, і зсув множеного відбувається вправо у разі нерухомої суми часткових добутків.

Варіанти із зсувом множеного на практиці не використовуються, оскільки для їх реалізації регістр множеного, регістр СЧД і суматор повинні мати розрядність *2n*, тому зупинимося на найбільш розповсюдженому 1-му варіанті, який має назву *алгоритм із зсувом вправо* (*алгоритм А).*

Множення чисел без знака

Загальну процедуру традиційного множення спочатку розглянемо стосовно чисел без знака, тобто таких чисел, в яких всі *n* розрядів представляють значущі цифри.

Процедура множення ілюструється прикладом обчислення добутку 10 × 11 (рис. 4.21).

A		1 0 1 0
B	×	1 0 1 1
0		0 0 0 0
$W_0 = Ab_0$	+	1 0 1 0
$P_0 = 0 + W_0$		0 1 0 1 0
$P_0 \times 2^{-1}$	⇒	0 1 0 1 0
$W_1 = Ab_1$	+	1 0 1 0
$P_1 = P_0 \times 2^{-1} + W_1$		0 1 1 1 1 0
$P_1 \times 2^{-1}$	⇒	0 1 1 1 1 0
$W_2 = Ab_2$	+	0 0 0 0
$P_2 = P_1 \times 2^{-1} + W_2$		0 0 1 1 1 1 0
$P_2 \times 2^{-1}$	⇒	0 0 1 1 1 1 0
$W_3 = Ab_3$	+	1 0 1 0
$P_3 = P_2 \times 2^{-1} + W_3$		0 1 1 0 1 1 1 0
$P_3 \times 2^{-1}$	⇒	0 1 1 0 1 1 1 0

Рис. 4.21. Приклад множення із зсувом суми часткових добутків вправо

Алгоритм зводиться до таких кроків:

1. Початкове значення суми часткових добутків приймається рівним нулю.

2. Аналізується чергова цифра множника (аналіз починається з молодшої цифри). Якщо вона дорівнює одиниці, то до СЧД додається множене, інакше (цифра дорівнює нулю) додавання не проводиться.

3. Виконується зсув суми часткових добутків вправо на один розряд.

4. Пункти 2 і 3 послідовно повторюються для всіх цифрових розрядів множника.

Алгоритм може бути реалізований за допомогою схеми, яка показана на рис. 4.22.

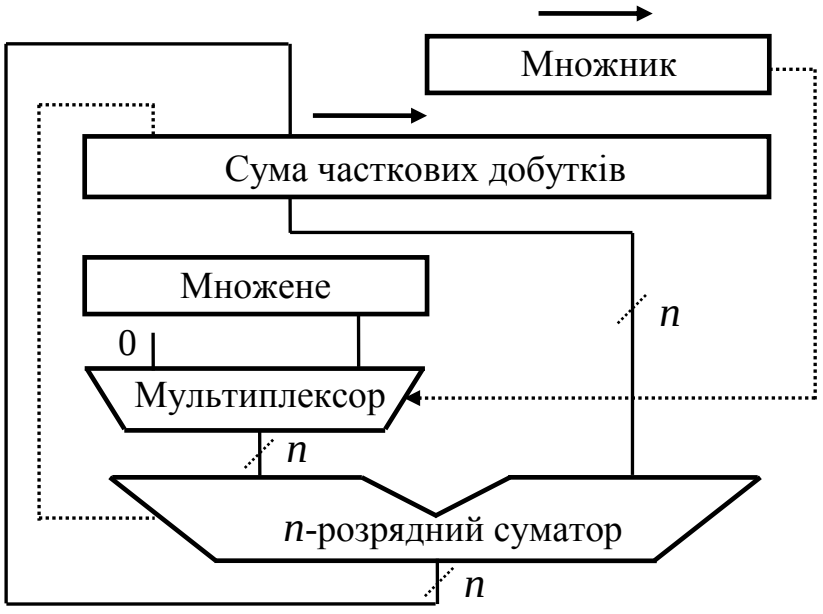


Рис. 4.22. Схема пристрою множення за алгоритмом *A*

Спочатку множене і множник заносяться в n-розрядні регістри множеного (РМн) і множника (РМк) відповідно, а всі розряди 2*n*-розрядного регістра суми часткових добутків (РЧД) встановлюються в 0. Множення відбувається за *n* кроків. На кожному кроці, залежно від стану молодшого розряду регістра множника, який керує мультиплексором, на один з входів *n*-разрядного суматора подається або множене, або 0. На другий вхід поступає вміст *n* старших розрядів РЧД. Новий частковий добуток з суматора пересилається в старші розряди РЧД. Далі вміст РЧД зсовується на один розряд вправо, причому в старший розряд регістра, що звільнився, заноситься значення переносу із старшого розряду суматора. Оскільки мультиплексор управляється молодшим розрядом РМк, то і вміст цього регістра також зсовується на один розряд вправо. Описана послідовність повторюється *n* разів.

Економічнішим у плані апаратури є інше рішення, коли замість двох регістрів – *n*-розрядного РМк і 2*n*-розрядного РЧД – використовується один комбінований 2*n*-розрядний регістр. Множник спочатку заноситься в молодші *n* розрядів цього регістра, а старші розряди обнуляються. В міру зсуву вправо молодші, вже проаналізовані розряди множника виштовхуються з регістра, звільняючи місце для чергової цифри СЧД. Зазвичай такий регістр будується з двох *n*-розрядных регістрів, об'єднаних ланцюгами зсуву. Додатково відзначимо, що якщо чергова цифра множника дорівнює 1, то для обчислення суми ЧД потрібні операції додавання і зсуву, а при нульовій цифрі множника в принципі можна обійтися без додавання, обмежившись тільки зсувом. Це, природно, вимагає деякої видозміни схеми.

Множення чисел із знаком

Дещо складніше йде справа з множенням чисел із знаком, коли *n*-розрядні співмножники містять знак (у старшому розряді слова) і *n*–1 значущу цифру. В подальшому умовимося відокремлювати знаковий розряд крапкою, не забуваючи, проте, що знаковий розряд бере участь в операції разом з цифровими розрядами.

Найбільш очевидна думка – набути абсолютних значень операндів і перемножити їх як числа без знака. Справедливість такого рішення видно з прикладу, приведеного на рис. 4.23, де показаний процес множення чисел +13 і +10.

У всіх ОМ загальноприйнято подавати числа із знаком у формі з фіксованою комою в доповнюючому коді. Додатні числа в цьому уявленні не відрізняються від запису в прямому коді, від'ємні записуються у вигляді $2^n - x$, де x – фактичне значення числа. В двійковій системі запис від'ємного числа в доповнюючому коді зводиться до інвертування всіх цифрових розрядів числа, представленого в прямому коді, і додавання одиниці до молодшого розряду зворотного коду, що вийшов після інвертування.

A	×	0. 1 1 0 1	+13
B		0. 1 0 1 0	+10
0		0. 0 0 0 0	
$W_0 = Ab_0$	+	0. 0 0 0 0	
$P_0 = 0 + W_0$		0. 0 0 0 0	
$P_0 \times 2^{-1}$	⇒	0. 0 0 0 0 0	
$W_1 = Ab_1$	+	0. 1 1 0 1	
$P_1 = P_0 \times 2^{-1} + W_1$		0. 1 1 0 1 0	
$P_1 \times 2^{-1}$	⇒	0. 0 1 1 0 1 0	
$W_2 = Ab_2$	+	0. 0 0 0 0	
$P_2 = P_1 \times 2^{-1} + W_2$		0. 0 1 1 0 1 0	
$P_2 \times 2^{-1}$	⇒	0. 0 0 1 1 0 1 0	
$W_3 = Ab_3$	+	0. 1 1 0 1	
$P_3 = P_2 \times 2^{-1} + W_3$		1. 0 0 0 0 0 1 0	
$P_3 \times 2^{-1}$	⇒	0. 1 0 0 0 0 1 0	+130

Рис. 4.23. Множення чисел при додатних співмножниках

Зупинемося на особливостях операції множення у випадку різних поєднань знаків співмножників. Перша з них виявляється під час виконання операції арифметичного зсуву вправо для суми часткових добутків – цифрові позиції, що звільнилися у разі зсуву, повинні заповнюватися не нулем, а значенням знакового розряду зсунутого числа. Тут, проте, слід враховувати, що це правило заповнення цифрових розрядів, що звільнилися, починає діяти лише з моменту, коли серед аналізованих розрядів множника з'являється перша одиниця.

Множене довільного знака, множник додатний. У разі від'ємного множеного процедура множення протікає аналогічно розглянутому, з урахуванням зробленого зауваження про арифметичний зсув СЧД (рис. 4.24).

A	×	1. 0 0 1 1	−13
B		0. 1 0 1 0	+10
0		0. 0 0 0 0	
$W_0 = Ab_0$	+	0. 0 0 0 0	
$P_0 = 0 + W_0$		0. 0 0 0 0	
$P_0 \times 2^{-1}$	⇒	0. 0 0 0 0 0	
$W_1 = Ab_1$	+	1. 0 0 1 1	
$P_1 = P_0 \times 2^{-1} + W_1$		1. 0 0 1 1 0	
$P_1 \times 2^{-1}$	⇒	1. 1 0 0 1 1 0	
$W_2 = Ab_2$	+	0. 0 0 0 0	
$P_2 = P_1 \times 2^{-1} + W_2$		1. 1 0 0 1 1 0	
$P_2 \times 2^{-1}$	⇒	1. 1 1 0 0 1 1 0	
$W_3 = Ab_3$	+	1. 0 0 1 1	
$P_3 = P_2 \times 2^{-1} + W_3$		10. 1 1 1 1 1 1 0	
$P_3 \times 2^{-1}$	⇒	1. 0 1 1 1 1 1 1 0	−130

Рис. 4.24. Множення чисел з від'ємним множеним і додатним множником

Оскільки результат множення від'ємний, він виходить у доповнюючому коді.

Множене довільного знака, множник від'ємний. На рис. 4.25 і 4.26 приведені приклади множення додатного і від'ємного множеного на від'ємний множник.

A		0. 1 1 0 1	+13
B	×	1. 0 1 1 0	−10
0		0. 0 0 0 0	
$W_0 = Ab_0$	+	0. 0 0 0 0	
$P_0 = 0 + W_0$		0. 0 0 0 0	
$P_0 \times 2^{-1}$	⇒	0. 0 0 0 0 0	
$W_1 = Ab_1$	+	0. 1 1 0 1	
$P_1 = P_0 \times 2^{-1} + W_1$		0. 1 1 0 1 0	
$P_1 \times 2^{-1}$	⇒	0. 0 1 1 0 1 0	
$W_2 = Ab_2$	+	0. 1 1 0 1	
$P_2 = P_1 \times 2^{-1} + W_2$		1. 0 0 1 1 1 0	
$P_2 \times 2^{-1}$	⇒	0. 1 0 0 1 1 1 0	
$W_3 = Ab_3$	+	0. 0 0 0 0	
$P_3 = P_2 \times 2^{-1} + W_3$		0. 1 0 0 1 1 1 0	
$P_3 \times 2^{-1}$	⇒	0. 0 1 0 0 1 1 1 0	
	+	1. 0 0 1 1	Корекція
		1. 0 1 1 1 1 1 1 1 0	−130

Рис. 4.25. Множення чисел у разі додатного множеного і від'ємного множника

A		1. 0 0 1 1	−13
B	×	1. 0 1 1 0	−10
0		0. 0 0 0 0	
$W_0 = Ab_0$	+	0. 0 0 0 0	
$P_0 = 0 + W_0$		0. 0 0 0 0	
$P_0 \times 2^{-1}$	⇒	0. 0 0 0 0 0	
$W_1 = Ab_1$	+	1. 0 0 1 1	
$P_1 = P_0 \times 2^{-1} + W_1$		1. 0 0 1 1 0	
$P_1 \times 2^{-1}$	⇒	1. 1 0 0 1 1 0	
$W_2 = Ab_2$	+	1. 0 0 1 1	
$P_2 = P_1 \times 2^{-1} + W_2$		0. 1 1 0 0 1 0	
$P_2 \times 2^{-1}$	⇒	1. 0 1 1 0 0 1 0	
$W_3 = Ab_3$	+	0. 0 0 0 0	
$P_3 = P_2 \times 2^{-1} + W_3$		1. 0 1 1 0 0 1 0	
$P_3 \times 2^{-1}$	⇒	1. 1 0 1 1 0 0 1 0	
	+	0. 1 1 0 1	Корекція
		0. 1 0 0 0 0 0 1 0	+130

Рис. 4.26. Множення чисел у разі від'ємного множеного і від'ємного множника

Оскільки множник від'ємний, він записується в доповнюючому коді: $[B]_{Д} = 2^n - |B|$, і в цифрових розрядах коду буде подане число $2^{n-1} - |B|$. У разі типового множення (як у випадку $B \geq 0$) отримаємо $= A \times (2^{n-1} - |B|) = -|B| \times A + A \times 2^{n-1}$. Псевдодобуток більше дійсного добутка P на величину $A \times 2^{n-1}$, що і необхідно враховувати під час формування остаточного результату. Для цього перед останнім зсувом з отриманого псевдодобутку необхідно відняти надлишковий член. У приклади множення є згадана корекція результату.

Розглянуті процедури множення чисел із знаком у принципі можуть бути реалізовані за допомогою раніше розглянутого пристрою (див. рис. 4.22). На практиці для перемножування чисел із знаком застосовують інші алгоритми. Найбільш поширеним з них є алгоритм Бута, що має додаткову перевагу, – він прискорює процес множення в порівнянні з розглянутим раніше.

В основі алгоритму Бута [25] лежить таке співвідношення, характерне для послідовностей двійкових цифр:

$$2^m + 2^{m-1} + \cdots + 2^k = 2^{m+1} - 2^k ,$$

де m і k – номери крайніх розрядів у групі з послідовних одиниць.

Наприклад, $011110 = 2^5 - 2^1$. Це означає, що за наявності в множнику груп з декількох одиниць (комбінацій вигляду 011, 110), послідовне додавання до СЧД множеного з наростаючою вагою (від 2^k до 2^m) можна замінити відніманням з СЧД множеного з вагою 2^k і додаванням до СЧД множеного з вагою 2^{m+1}.

Як видно, алгоритм припускає три операції: зсув, додавання і віднімання. Крім скорочення числа додавань (віднімань) у нього є ще одна перевага – він у рівній мірі застосовний до чисел без знака і зі знаком.

Розглянуті алгоритми відносилися до подання чисел з фіксованою комою, тобто, як це прийнято в більшості ОМ, до цілих чисел. При перемножуванні чисел із знаком необхідно брати до уваги, що добуток двох n-розрядних чисел із знаком (знак і $n-1$ значущий розряд) може мати $2(n-1)$ значущих розрядів і для його зберігання зазвичай використовують регістр подвійної довжини ($2n$ розрядів). Оскільки число ітерацій в операції множення визначається кількістю цифрових розрядів множника, кінцевий результат може розміщуватися в розрядній сітці подвійного слова невірно. Молодший розряд добутку цілих чисел, що має вагу 2^0, розміщується в позиції подвійного слова, яка має вагу 2^1. Таким чином, для правильного розташування добутку в розрядній сітці подвійного слова необхідний додатковий зсув вправо. Такий зсув можна врахувати як в апаратурі помножувача, так і програмним способом.

З іншого боку, при перемножуванні правильних дробів додатковий зсув не потрібний [25]. Цю обставину необхідно брати до уваги під час побудови помножувача для чисел у формі з плаваючою комою, де мантиси, що беруть

участь в операції, подані в нормалізованому вигляді, тобто правильними дробами.

Під час множення правильних дробів часто обмежуються результатом, що має одинарну довжину. В цьому випадку може застосовуватися або відкидання зайвих розрядів, або округлення.

Прискорення цілочисельного множення

Методи прискорення множення можна умовно розділити на апаратні і логічні. Ті та інші вимагають додаткових витрат обладнання, які під час використання апаратних методів зростають із збільшенням розрядності співмножників. Апаратні способи приводять до ускладнення схеми помножувача, але не зачіпають схеми управління. Додаткові витрати обладнання під час реалізації логічних методів не залежать від розрядності операндів, але схема управління помножувача при цьому ускладнюється. На практиці прискорення множення часто досягається комбінацією апаратних і логічних методів.

Логічні методи прискорення множення. Логічні підходи до прискорення множення можна поділити на дві групи:

- методи, що дозволяють зменшити кількість додавань у ході множення;
- методи, що забезпечують обробку декількох розрядів множника за крок.

Реалізація і тих і інших вимагає введення додаткових ланцюгів зсуву в регістри.

Розглянемо другу групу логічних методів [25].

Зупинимося на множенні з обробкою за крок двох розрядів множника (IBM 360/370). В принципі це ефективніша версія алгоритму Бута. Аналіз множника починається з молодших розрядів. Залежно від вхідної дворозрядної комбінації передбачаються такі дії:

- 00 – звичайний зсув на два розряди вправо суми часткових добутків (СЧД);
- 01 – до СЧД додається одинарне множене, після чого СЧД зсовується на 2 розряди вправо;
- 10 – до СЧД додається подвоєне множене, і СЧД зсовується на 2 разряди вправо;
- 11 – з СЧД віднімається одинарне множене, і СЧД зсовується на 2 розряди вправо. Отриманий результат повинен бути скоректований на наступному кроці, що фіксується в спеціальному тригері ознаки корекції.

Оскільки в разі пари 11 з СЧД віднімається одинарне множене замість збільшення потрійного, для коректування результату до СЧД перед виконанням зсуву треба було б додати збільшене вчетверо множене. Але після зсуву на два розряди вправо СЧД зменшується в чотири рази, так що на наступному кроці досить додати одинарне множене. Це враховується під час обробки наступної

пари розрядів множника, шляхом обробки пари 00 – як 01, пари 01 – як 10, 10 – як 11, а 11 – як 00. В останніх двох випадках фіксується ознака корекції.

Після обробки кожної комбінації вміст регістра множника і суматора часткових добутків зсовується на 2 розряди вправо. Даний метод множення вимагає коректування результату, якщо старша пара розрядів множника дорівнює 11 або 10 і стан ознаки корекції одиничний. У цьому випадку до отриманого добутку повинне бути додане множене.

Апаратні методи прискорення множення. Традиційний метод множення за рахунок зсувів і додавань, навіть у разі його апаратної реалізації, не дозволяє досягти високої швидкості виконання операції множення. Зв'язано це, головним чином, з тим, що під час додавання до СЧД чергового часткового добутку перенос повинен розповсюдитися від молодшого розряду СЧД до старшого. Затримка через розповсюдження переносу відносно велика, причому вона повторюється під час додавання кожного ЧД.

Один із способів прискорення множення полягає в зміні системи кодування співмножників, за рахунок чого можна скоротити кількість підсумовуваних часткових добутків. Прикладом такого підходу може служити алгоритм Бута.

Ще один ресурс підвищення продуктивності помножувача – використання ефективніших способів підсумовування ЧД, що виключають витрати часу на розповсюдження переносів. Досягається це за рахунок подання ЧД у надмірній формі, завдяки чому підсумовування двох чисел не пов'язане з розповсюдженням переносу уздовж всіх розрядів числа. Найбільш уживаною формою такого надмірного кодування є так звана форма *із збереженням переносу*. В ній кожен розряд числа подається двома бітами *cs*, відомими як перенос (*c*) і сума (*s*). Під час підсумовування двох чисел у формі із збереженням переносу перенос розповсюджується не далі, чим на один розряд. Це робить процес підсумовування значно швидшим, ніж у разі додавання з розповсюдженням переносу уздовж всіх розрядів числа.

Нарешті, третя можливість прискорення операції множення полягає в паралельному обчисленні всіх часткових добутків. Якщо розглянути загальну схему множення [25], то неважко відмітити, що окремими розрядами ЧД є добутки виду $a_i b_j$, тобто добуток певного біта множеного на певний біт множника. Це дозволяє обчислити всі біти часткових добутків одночасно, за допомогою n^2 схем «І». Під час перемножування чисел в доповнюючому коді окремі розряди ЧД можуть мати вид $\overline{a_i} b_j$, $a_i \overline{b_j}$ або $\overline{a_i b_j}$. Тоді елементи «І» замінюються елементами, що реалізують відповідну логічну функцію.

Таким чином, апаратні методи прискорення множення зводяться:

- до паралельного обчислення часткових добутків;
- до скорочення кількості операцій додавання;
- до зменшення часу розповсюдження перенесень під час підсумовування часткових добутків.

Всі три підходи в будь-якому їх поєднанні зазвичай реалізуються за допомогою комбінаційних пристроїв.

Паралельне обчислення ЧД має багато різновидів. Відмінності виявляються в основному в способі підсумовування отриманих часткових добутків, і з цих позицій використовувані схеми множення можна поділити на *матричні* і *з деревовидною структурою*. В обох варіантах підсумовування здійснюється за допомогою масиву взаємозв'язаних одноразрядних суматорів. У матричних помножувачах суматори організовані у вигляді матриці, а в деревовидних вони реалізуються у вигляді дерева того або іншого типу.

Відмінності в рамках кожної з цих груп виражаються в кількості використовуваних суматорів, їх вигляді і способі розповсюдження переносів, що виникають у процесі додавання.

В *матричних помножувачах* підсумовування здійснюється матрицею суматорів, що складається з послідовних лінійок (рядків) одноразрядних суматорів із збереженням переносів (СЗП). В міру руху даних вниз по масиву суматорів кожен рядок СЗП додає до СЧД черговий частковий добуток. Оскільки проміжні СЧД представлені в надмірній формі із збереженням переносу, у всіх схемах, аж до останнього рядка, де формується остаточний результат, розповсюдження переносу не відбувається. Це означає, що затримка в помножувачах відштовхується тільки від «глибини» масиву (числа рядків суматорів) і не залежить від розрядності операндів, якщо тільки в останньому рядку матриці, де формується остаточна СЧД, не використовується схема з послідовним переносом.

Разом з високою швидкодією важливою перевагою матричних помножувачів є їх регулярність, що особливо істотно під час реалізації таких помножувачів у вигляді інтегральної мікросхеми. З іншого боку, подібні схеми займають велику площу на кристалі мікросхеми, причому із збільшенням розрядності співмножників ця площа збільшується пропорційно квадрату числа розрядів. Друга проблема з матричними помножувачами – низький рівень утилізації апаратури. У міру руху СЧД вниз кожен рядок задіюється лише одноразово, коли її перетинає активний фронт обчислень. Ця обставина, проте, може використовуватись для підвищення ефективності обчислень шляхом конвеєризації процесу множення, при якій в міру звільнення рядка суматорів остання може бути використана для множення чергової пари чисел.

Цілочисельне ділення

Ділення дещо складніша операція, ніж множення, але базується на тих же принципах. Основу складає загальноприйнятий спосіб ділення за допомогою операцій віднімання або додавання і зсуву (рис. 4.27).

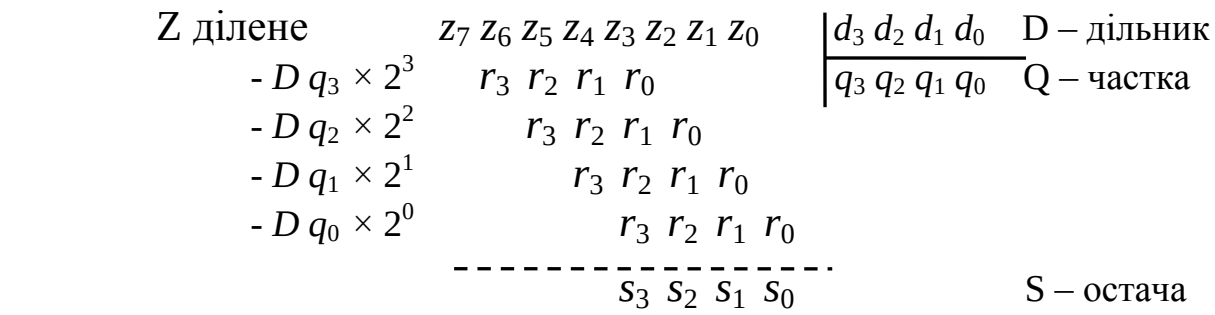


Рис. 4.27. Загальна схема операції ділення

Задача зводиться до обчислення частки Q і залишку S:

$$Q = \text{int}\left(\frac{Z}{D}\right), \qquad S = Z - QD, \qquad S < D.$$

Ділення виражається як послідовність віднімань дільника спочатку з діленого, а потім з часткових залишків (ЧЗ), що утворюються в процесі ділення. Ділене Z ($z_{2n-1}\ z_{2n-2}\ \ldots\ z_1\ z_0$) зазвичай записується подвійним словом ($2n$ розрядів), дільник D ($d_{2n-1}\ d_{2n-2}\ \ldots\ d_1\ d_0$), частка Q ($q_{2n-1}\ q_{2n-2}\ \ldots\ q_1\ q_0$) і залишок S ($s_{2n-1}\ s_{2n-2}\ \ldots\ s_1\ s_0$) мають розрядність n.

Операція виконується за n ітерацій і може бути описана таким чином:

$$S^{(i)} = 2S^{(i-1)} - q_{n-i}(2^n D), \text{ якщо } S^{(0)} = Z \text{ і } S^{(n)} = 2^n S;$$

$$q_{n-i} = \begin{cases} 1, \text{якщо } (2S^{(i-1)} - 2^n D) \geq 0, \\ 0, \text{якщо } (2S^{(i-1)} - 2^n D) < 0. \end{cases}$$

Після n ітерацій виходить

$$S^{(n)} = 2^n S^{(0)} - Q(2^n D) = 2^n[Z - (Q \times D)] = 2^n S.$$

Частка від ділення $2n$-розрядного числа на n-розрядне може містити більше, ніж n розрядів. У цьому випадку виникає переповнення, через що перед виконанням ділення необхідна перевірка умови

$$Z < (2^n - 1)D + D = 2^n D$$

З виразу виходить, що переповнення не буде, якщо число, що міститься в старших n розрядах діленого, менше дільника.

Крім цієї вимоги, перед початком операції необхідно виключити можливість ситуації ділення на 0.

Реалізувати ділення можна двома основними способами:

- з нерухомим діленим і зсовуваним вправо дільником;
- з нерухомим дільником і зсовуваним вліво діленим.

Недоліком першого способу є потреба мати в пристрої ділення суматор і регістр подвійної довжини. Другий спосіб дозволяє будувати пристрій делення з суматором одинарної довжини. Нерухомий дільник *D* зберігається в регістрі одинарної довжини, а ділене *Z*, що зсовується відносно *D*, знаходиться в двох таких же регістрах. Цифри частки *Q*, що утворюються, заносяться в розряди одного з регістрів *Z*, які звільняються при зсуві розрядів *Z*.

Нижче на прикладі чисел без знаку розглядаються два основні алгоритми цілочисельного ділення.

Ділення з відновленням залишку. Найбільш очевидний алгоритм носить назву алгоритму *ділення з нерухомим дільником і відновленням залишку*. Він дуже схожий на загальноприйнятий спосіб ділення стовпчиком.

На рис. 4.28 показаний процес ділення з відновленням залишку числа 41 на 7.

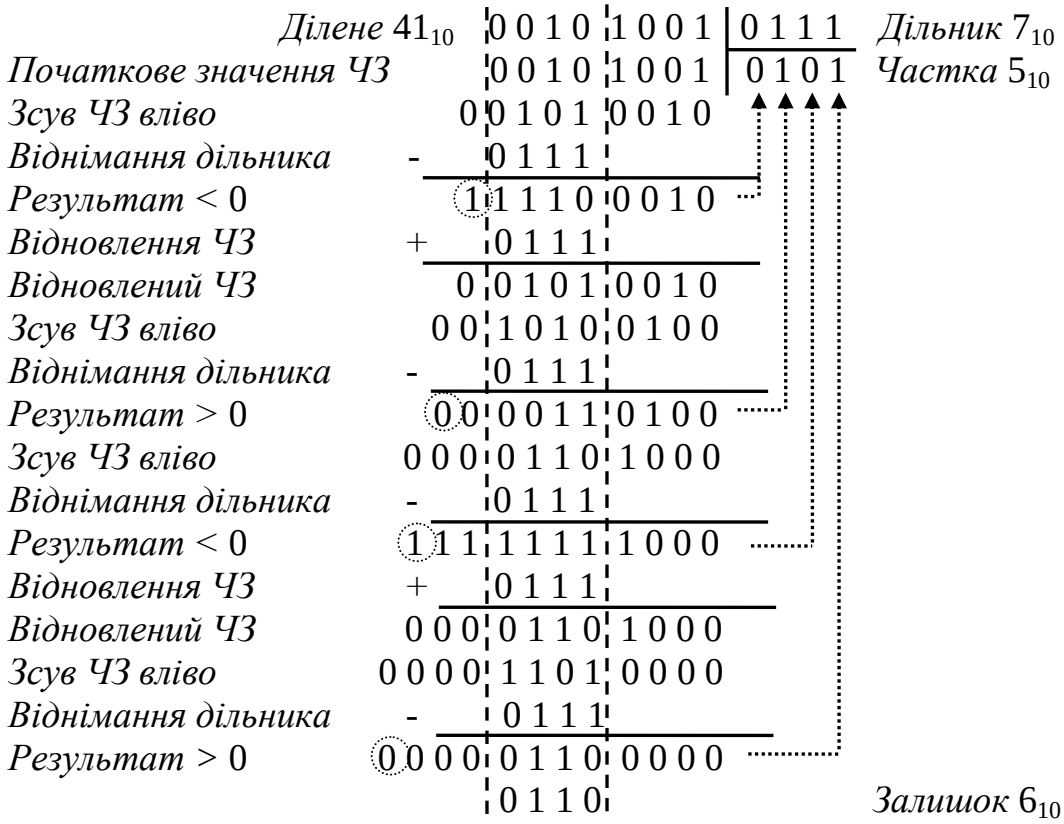


Рис. 4.28. Приклад ділення з відновленням залишку

Ділення без відновлення залишку. Недолік розглянутого алгоритму полягає в необхідності виконання на окремих кроках додаткових операцій додавання для відновлення часткового залишку. Це збільшує час виконання ділення, яке в цьому випадку може мінятися залежно від конкретного поєднання кодів операндів. Через вказані причини реальні пристрої ділення

будуються на основі алгоритму *ділення з нерухомим дільником без відновлення залишку*.

Процес ділення без відновлення залишку для раніше розглянутого прикладу демонструється на рис. 4.29.

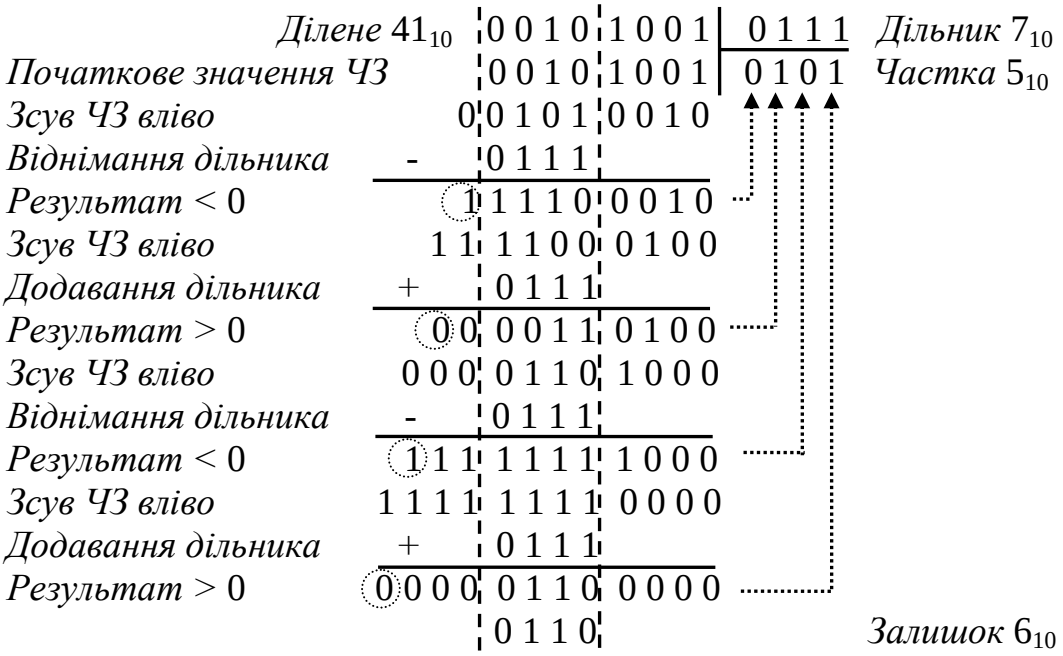


Рис. 4.29. Приклад ділення без відновлення залишку

Приведемо опис цього алгоритму.

1. Початкове значення часткового залишку вважається рівним старшим розрядам діленого.

2. Частковий залишок подвоюється шляхом зсуву на один розряд вліво. При цьому в молодший разряд ЧЗ, що звільняється під час зсуву, заноситься чергова цифра частки.

3. Із зсунутого часткового залишку віднімається дільник, якщо залишок додатний, і до зсунутого часткового залишку додається дільник, якщо залишок від'ємний.

4. Чергова цифра модуля частки дорівнює одиниці, коли результат віднімання додатний, і нулю, якщо він від'ємний.

5. Пункти 2…4 послідовно виконуються для отримання всіх цифр модуля частки.

Як бачимо, пункти 1, 2, 5 повністю збігаються з відповідними пунктами попереднього алгоритму ділення.

Ділення чисел із знаком

Як і у разі множення, ділення чисел із знаком може бути виконане шляхом переходу до абсолютних значень діленого і дільника, з подальшим

привласненням частки знака «плюс» у разі збігу знаків діленого і дільника або «мінус» – інакше.

Ділення чисел, представлених у доповнюючому коді, можна здійснювати не переходячи до модулів. Розглянемо необхідні для цієї зміни в алгоритмі без відновлення залишку.

Оскільки ділене і дільник не обов'язково мають однакові знаки, то дії з частковим залишком (додавання або віднімання D) залежать від знаків залишку і дільника і визначаються вмістом табл. 4.1:

Таблиця 4.1

Операція, яка виконується в черговій ітерації ділення

Знак залишка	Знак дільника	Дія
+	+	Віднімання дільника
+	−	Додавання дільника
−	+	Додавання дільника
−	−	Віднімання дільника

- Якщо знак залишку збігається із знаком дільника, то чергова цифра частки − 1, інакше − 0.
- Якщо $Z > 0$ і $D < 0$, частку необхідно збільшити на 1.
- Якщо $Z < 0$ і $D > 0$, то при ненульовому залишку від ділення частку потрібно збільшити на одиницю.
- Якщо $Z < 0$ і $D < 0$, то у разі нульового залишку від ділення частку потрібно збільшити на одиницю.

Залишок завжди приводиться до додатного числа, тобто якщо після закінчення ділення він від'ємний, до нього слід додати модуль дільника.

Пристрій ділення

Розглянутий алгоритм ділення без відновлення залишку може бути реалізований за допомогою пристрою, схема якого приведена на рис. 4.30.

Процедура починається із занесення діленого в $2n$-розрядний регістр діленого (РДн) і дільника в n-розрядний регістр дільника (РДк). В лічильник циклу (ЛчЦ – на схемі не показаний), який служить для підрахунку кількості отриманих цифр частки, поміщається початкове значення, яке дорівнює n.

На кожному кроці вміст регістра діленого (РДн) і регістра частки (РЧ) зсовується на один розряд вліво. Залежно від поєднання знаків часткового залишку і дільника визначається значення чергової цифри частки і необхідна дія: віднімання або додавання дільника. Віднімання дільника проводиться за допомогою додавання доповнюючого коду дільника. Перетворення в доповнюючий код здійснюється за рахунок передачі дільника на вхід суматора зворотним (інверсним) кодом з подальшим додаванням одиниці до молодшого розряду суматора.

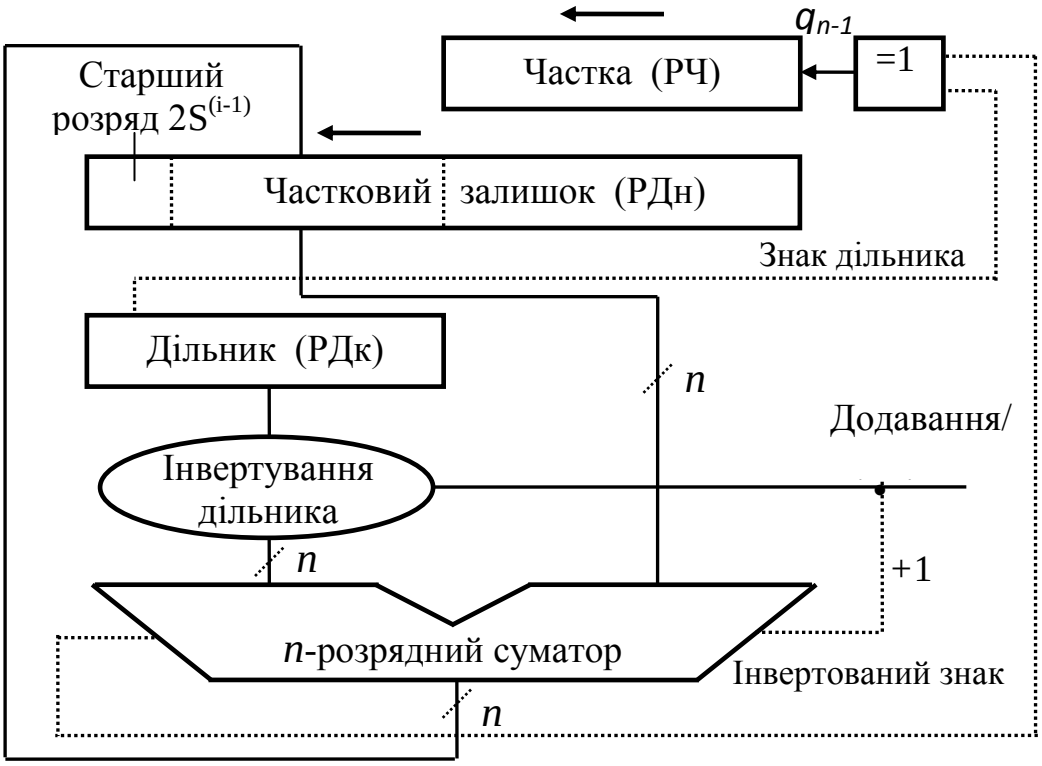


Рис. 4.30. Схема ділення за алгоритмом без відновлення залишку

Описана процедура повторюється до вичерпання всіх цифр діленого, про що свідчить нульовий вміст лічильника циклів ЛчЦ. Після закінчення операції ділення частка розташовується в регістрі частки, а в регістрі діленого буде залишок від ділення.

На завершальному етапі, якщо це необхідно, проводиться коректування отриманого результату.

На практиці для накопичення і зберігання частки замість окремого регістра використовують молодші розряди регістра діленого, що звільняються в процесі зсувів.

Прискорення цілочисельного ділення

Слід зазначити, що операція ділення надає не дуже багато шляхів для своєї оптимізації за часом. Проте певні можливості для прискорення ділення існують, і їх можна звести до таких:

- заміна дільника зворотною величиною, з подальшим її множенням на ділене;
- скорочення часу обчислення часткових залишків в традиційних методах ділення (з відновленням або без відновлення залишку) за рахунок прискорення операцій підсумовування (віднімання);
- скорочення часу обчислення за рахунок зменшення кількості операцій підсумовування (віднімання) під час розрахунку ЧЗ;
- обчислення частки в надмірній системі числення.

Детальніше розглянемо перший з перерахованих підходів, всі інші фактично є модифікаціями традиційного способу ділення.

Операцію множення можна проводити порівняно швидко, якщо взяти на озброєння комбінаційні схеми паралельного множення. Дану обставину можна використовувати, замінивши операцію ділення на D множенням на

$$Q = \frac{1}{D} = Z \times \frac{1}{D}.$$

У цьому випадку проблема зводиться до ефективного обчислення $1/D$. Зазвичай задача вирішується одним з двох методів: за допомогою ряду Тейлора або методу Ньютона-Рафсона. В обох випадках основний час витрачається на множення, тому даний метод прискорення ділення має сенс за наявності швидких схем множення.

У разі реалізації першого методу дільник D подається у вигляді: $D = 1 + X$. Тоді для двійкового подання D можна записати:

$$\frac{1}{D} = (1 - X) \times (1 + X^2) \times (1 + X^4) \times (1 + X^8) \times (1 + X^{16})...$$

Метод був використаний у моделі 91 обчислювальної машини IBM 360 для обчислення 32-розрядної величини $1/D$. Можливі значення співмножників у правій частині виразу витягувалися з таблиці ємністю 28 байт, яка зберігалася в пам'яті. Операція обчислення $1/D$ вимагає шести множень.

Обчислення величини $1/D$ методом Ньютона-Рафсона зводиться до знаходження кореня рівняння

$$f(X) = \frac{1}{X} - D = 0,$$

тобто $X = 1/D$. Рішення може бути отримане із залученням рекурентного співвідношення: $X_{i+1} = X_i\,(2 - X_i\,D)$. Кількість ітерацій визначається необхідною точністю обчислення $1/D$. Реалізація методу для n-розрядних чисел вимагає $2\ \mathrm{int}\,(\log_2 n) - 1$ операцій множення.

Загалом, заміна операції ділення на множення характерніша для чисел з плаваючою комою.

Можливості *прискорення обчислення часткових залишків* значно обмежені і пов'язані в основному з прискоренням операцій додавання (віднімання). Способи досягнення цієї мети нічим не відрізняються від тих, що застосовуються, наприклад, під час виконання множення. Це різні прийоми для прискорення розповсюдження переносів, матричні схеми додавання і тому подібне.

В основі *третьої групи методів прискорення операції ділення*, згідно з приведеною вище класифікацією, лежить так званий алгоритм SRT [25]. Свою назву алгоритм отримав по прізвищах авторів (Sweeney, Robertson, Tocher), що

розробили його незалежно один від одного приблизно в один і той же час. Цей алгоритм є модифікацією ділення без відновлення залишку. В стандартній процедурі на кожному кроці крім зсуву часткового залишку проводиться додавання або віднімання дільника. В SRT-алгоритмі зсув ЧЗ також є в кожній ітерації, проте додавання або віднімання, залежно від ЧЗ, що виходить, на окремих кроках може не виконуватися, що, природно, позитивно впливає на швидкодію ділення.

Алгоритм був орієнтований на операції над мантисами чисел з плаваючою комою і спирається на ту обставину, що мантиси в таких числах нормалізовані. Вперше SRT-алгоритм був реалізований у моделі 91 обчислювальної
машини IBM 360. В даний час він широко застосовується в блоках обробки чисел з плаваючою комою, зокрема в мікропроцесорах фірми Intel.

Найбільш поширені методи прискорення операції ділення засновані на застосуванні алгоритмів, де ділення виконується в *надмірних системах числення*, тобто в системах числення, відмінних від двійкової. Це означає, що цифри частки можуть мати більше, ніж два значення, наприклад {-1,0,1}, як це було в алгоритмі множення Бута, або {-2,-1,0,1,2}. В таких системах одне і те ж число може бути записане декількома способами, через що системи називають надмірними. Чергова цифра частки в надмірній системі числення, залежно від бази цієї системи, відповідає двом або більше цифрам у двійковому уявленні частки, і для потрібної кількості двійкових цифр частки і залишку потрібно менше ітерацій. У той же час реалізація такого підходу веде до ускладнення апаратури дільника, зокрема, надбудовується логіка визначення операції, що виконується в черговій ітерації. Для цієї мети до складу пристрою ділення включається спеціальна пам'ять, що зберігає таблицю, яка визначає необхідні дії, залежно від поточної комбінації цифр у частковому залишку і дільнику. Проте виграш у швидкодії є вирішальним моментом.

Операційні пристрої з плаваючою комою

Операції над числами у форматі з плаваючою комою (ПК) мають істотні відмінності від аналогічних операцій цілочисельної арифметики, тому їх зазвичай реалізують за допомогою самостійного операційного пристрою. Як і цілочисельний ОПр, операційний пристрій для чисел у форматі ПК як мінімум повинен забезпечувати виконання чотирьох арифметичних дій: додавання, віднімання, множення і ділення.

Після ухвалення стандарту IEEE 754 негласною вимогою до всіх ОМ є забезпечення операцій з числами, поданими у форматах, які визначені даним стандартом. Нагадаємо основні положення запису чисел у стандарті IEEE 754. Мантиси чисел *M* подаються у нормалізованому вигляді, при цьому діє прийом прихованого розряду, коли старша цифра мантиси, яка завжди дорівнює

одиниці, в записі числа відсутня, тобто в полі мантиси старшою є друга старша цифра нормалізованої мантиси.

На відміну від загальноприйнятої умови нормалізації S = |*M*| < 1, в стандарті IEEE 754 використовується умова 1 = |M| < 2.

Запис числа містить зміщений порядок, тобто порядок, збільшений на величину зміщення, яке в стандарті IEEE 754 для одинарного формату дорівнює 127, а для подвійного − 1023.

З урахуванням перерахованих особливостей арифметичну операцію над числами у форматі з плаваючою комою можна записати у вигляді:

$$Z_M \times 2^{Z_{СП}} = (\pm X_M \times 2^{X_{СП}}) \Diamond (\pm Y_M \times 2^{Y_{СП}}),$$

де X_M, Y_M, Z_M − нормалізовані мантиси операндів і результату;
$X_{СП}$, $Y_{СП}$, $Z_{СП}$ − зміщені порядки операндів і результату;
$\Diamond$ − знак арифметичної операції.

При всіх відмінностях у виконанні різних арифметичних операцій підготовчий і завершальний етапи у всіх випадках збігаються, через що має сенс розглянути їх окремо.

Підготовчий етап. Першою особливістю операційних пристроїв для чисел з плаваючою комою є те, що в них операції над трьома складовими чисел з ПК (знаками, мантисами і порядками операндів) виконуються роздільно: блоком обробки знаків (БОЗ), блоком обробки порядків (БОП) і блоком обробки мантис (БОМ). Для зберігання операндів і результату в ОП передбачені відповідні регістри. Хоч ці регістри можуть бути фізично реалізовані у вигляді єдиних пристроїв, кожен з них логічно розглядати як сукупність трьох регістрів: знака, порядку і мантиси. Таким чином, на етапі завантаження операндів у регістри ОП здійснюється «розпаковка» чисел з ПК, їх розбиття на три складові. У ході такого розпаковування в старшому розряді регістра мантиси відновлюється одиниця, яка в записі числа була відсутня (була прихована).

На попередньому етапі може бути також виконана перевірка на рівність нулю одного або обох операндів (у стандарті IEEE 754 для представлення нульового значення використовується такий запис числа, в якому нулю дорівнюють усі розряди порядку). Це дозволяє виключити непотрібні операції. Так, в операціях множення і ділення, якщо нулю рівні множник, множене або ділене, як результат відразу ж можна прийняти нульове значення, обійшовши наказані даними операціями дії.

Завершальний етап. Дії на завершальному етапі виконання будь-якої арифметичної операції ідентичні і зводяться до виявлення нульового значення мантиси (втрати значимості мантиси), нормалізації мантиси, виявлення від'ємного переповнювання порядку, "упаковки" складових результату.

Нульове значення мантиси може вийти в результаті операції, наприклад під час складання або віднімання мантис. Другою причиною може стати зсув мантиси вправо для усунення переповнення. В обох випадках має місце ситуація *втрати значимості мантиси*, і результат операції приймається рівним нулю. Для стандарту IEEE 754 це означає, що всі цифри порядку результату необхідно обнулити, а також те, що нормалізацію мантиси результату проводити не потрібно.

Нормалізація мантиси результату зводиться до послідовного її зсуву вліво до тих пір, поки старшу позицію не займе одиниця. Кожен зсув супроводжується зменшенням на одиницю порядку результату. В ході зменшення порядок може стати від'ємним, що для зміщених порядків свідчить про отримання числа, неуявного в даному форматі. В такій ситуації результат приймається рівним нулю і одночасно формується ознака *втрати значимості порядку*.

На завершення мантиса результату округляється і, якщо це передбачено форматом ПК, з неї видаляється прихований розряд.

В останній фазі здійснюється «упаковка» всіх складових результату (знака, порядку і мантиси), після чого сформований результат заноситься у вихідний регістр ОП.

Додавання і віднімання

В арифметиці з плаваючою комою додавання і віднімання − складніші операції, ніж множення і ділення. Обумовлено це необхідністю вирівнювання порядків операндів. Алгоритм додавання і віднімання включає такі основні фази:

1. Підготовчий етап.
2. Визначення операнда, що має менший порядок, і зсув його мантиси вправо на число розрядів, яке дорівнює різниці порядків операндів.
3. Прирівнювання порядку результату більшому з порядків операндів.
4. Додавання або віднімання мантис і визначення знака результату.
5. Перевірку на переповнювання.
6. Завершальний етап.

Додавання і віднімання виконуються ідентично, але у разі віднімання необхідно змінити знак другого операнда на протилежний.

Множення

На початковому етапі множення чисел з ПК проводиться перевірка на рівність нулю одного із співмножників. Якщо один з операндів дорівнює нулю, як результат видається 0, поданий у даному форматі чисел з ПК. Наступний крок − додавання порядків. Результатом дій з порядками може стати як переповнення порядку, так і втрата значимості. У обох випадках виконання

операції припиняється і видається відповідне повідомлення (виникає переривання).

Якщо порядок результату лежить у допустимих межах, на наступному кроці проводиться перемножування мантис з урахуванням їх знака, яке виконується так само, як для чисел з фіксованою комою. Під час розміщення добутку мантис у розрядній сітці необхідно враховувати, що мантиси подані не цілими числами, а правильними дробами. Хоч результат множення мантис має подвоєну в порівнянні з операндами довжину, він округляється до довжини поля мантиси.

На останньому кроці проводиться нормалізація і компоновка результату, аналогічно тому, як це має місце під час додавання і віднімання.

Ділення

Спочатку також проводиться перевірка на 0. Якщо нулю дорівнює дільник, залежно від реалізації видається повідомлення про ділення на 0, або як результат приймається нескінченність. Коли нулю дорівнює ділене, результат також приймається рівним нулю.

Далі виконується віднімання порядку дільника з порядку діленого, що приводить до видалення зсуву з порядку результату. Отже, для отримання зміщуного порядку результату до різниці повинен бути доданий зсув. Після виконання цих дій необхідна перевірка на переповнювання порядків і втрату значимості.

Наступний крок − ділення мантис, за яким йдуть нормалізація, округлення і компоновка числа з мантиси і порядку.

Реалізація логічних операцій

Крім арифметичних дій ОПр будь-якої обчислювальної машини припускає виконання основних логічних операцій і зсувів. Найчастіше такі операції реалізуються додатковими схемами, що входять до складу цілочисельних ОПр.

До базових логічних операцій відносяться: логічне заперечення (НІ), логічне додавання (АБО) і логічне множення (І). Цей набір, як правило, доповнюють операцією додавання по модулю 2 (виключаюче АБО). Можлива структура операційного блока (ОБ) для виконання логічних операцій показана на рис. 4.31.

Булева змінна в ОМ подається одним бітом, проте на практиці логічні операції в ОПр виконуються відразу над сукупністю логічних змінних, об'єднаних у рамках одного байта або машинного слова, причому над всіма бітами цього слова виконується одна і та ж операція. Оскільки кожен біт сукупності логічних змінних розглядається як незалежна змінна, ніякі перенесення між розрядами не формуються.

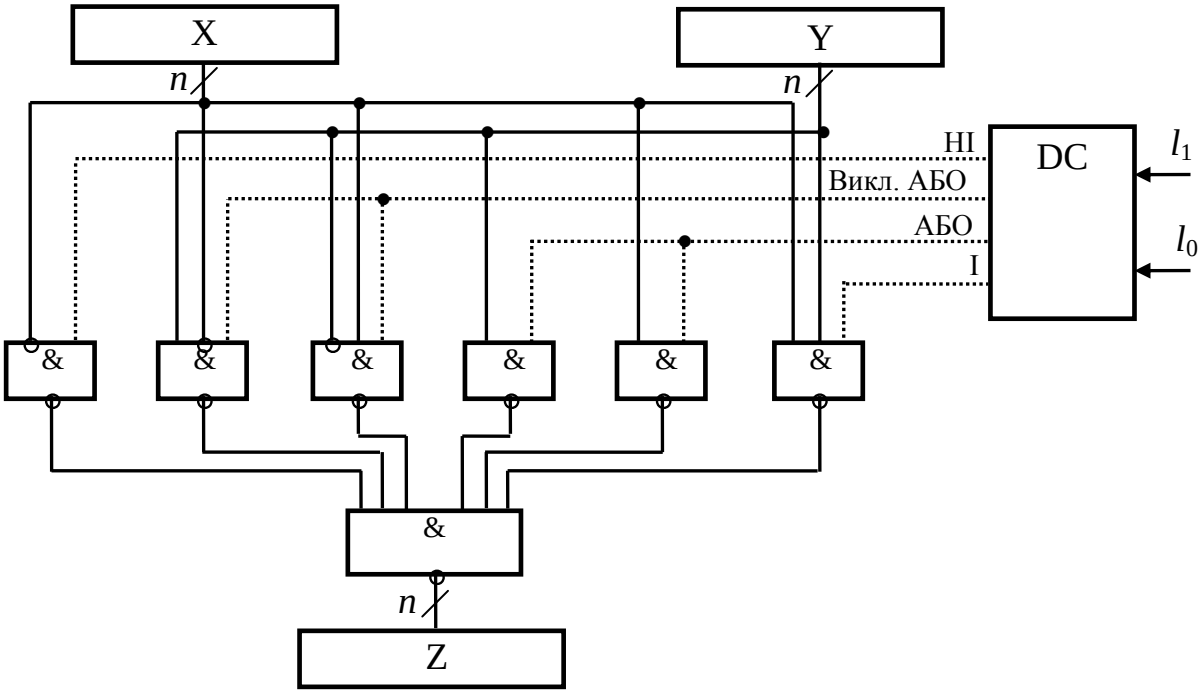


Рис. 4.31. Структура операційного блока для виконання логічних операцій

Вибір потрібної операції здійснюється за допомогою дворозрядного управляючого коду $L(l_1 l_0)$. З урахуванням управляючого коду, вихідна функція Z може бути описана виразом

$$Z = \overline{l_1} \wedge \overline{l_0} \wedge \overline{X} \vee \overline{l_1} \wedge l_0 \wedge (\overline{X} \wedge Y \vee X \wedge \overline{Y}) \vee l_1 \wedge \overline{l_0} \wedge (X \vee Y) \vee l_1 \wedge l_0 \wedge (X \wedge Y).$$

4.5. ОСНОВНІ НАПРЯМКИ В АРХІТЕКТУРІ ПРОЦЕСОРІВ

4.5.1. Конвеєризація обчислень

Вдосконалення елементної бази вже не приводить до кардинального зростання продуктивності ОМ. Комп'ютер у цілому являє собою складну логічну схему, і, як правило, в певні періоди тактових сигналів одні вузли цієї схеми працюють, а інші ні. Тому для підвищення швидкості обробки інформації схему потрібно побудувати так, щоб у ній працювала одночасно максимальна кількість вузлів. Перспективнішими в цьому плані являються архітектурні прийоми, серед яких один з найбільш значущих – конвеєризація (рис. 4.32).

Для пояснення ідеї конвеєра спочатку звернемося до рис. 4.32, *а*, де показаний окремий функціональний блок (*ФБ*). Початкові дані поміщаються у вхідний регістр $Рг_{вх}$, обробляються у функціональному блоці, а результат обробки фіксується у вихідному регістрі $Рг_{вих}$. Якщо максимальний час обробки у *ФБ* дорівнює T_{max}, то нові дані можуть бути занесені у вхідний регістр $Рг_{вх}$ не раніше, ніж опісля T_{max}.

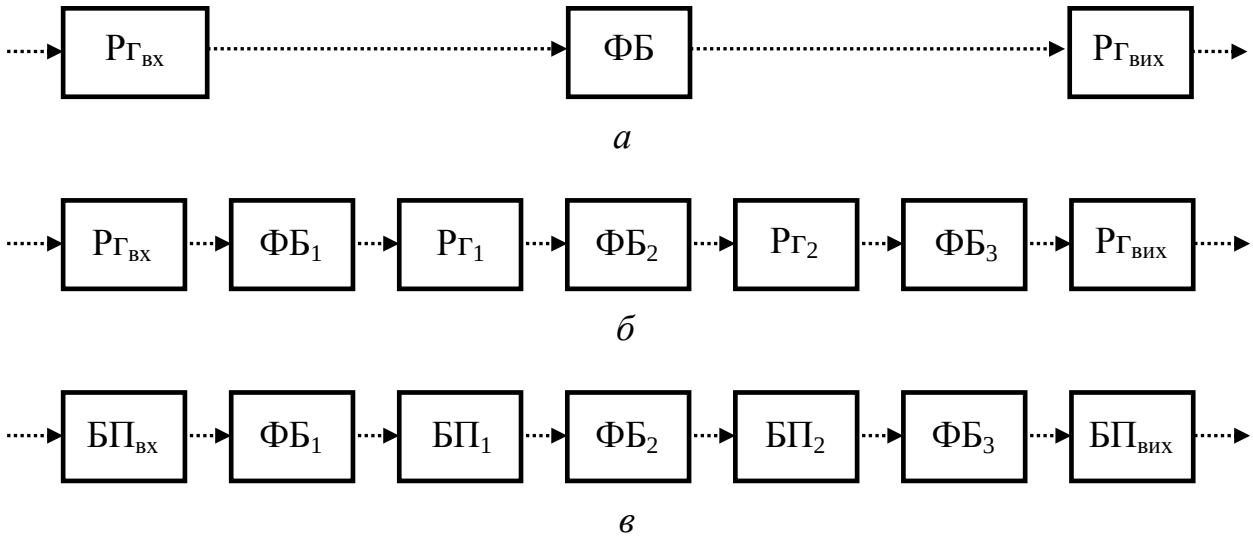


Рис. 4.32. Обробка інформації: *а* – в поодинокому блоці; *б* –в конвеєрі з регістрами; *в* – в конвеєрі з буферною пам'яттю

Тепер розподілимо функції, що виконуються у функціональному блоці *ФБ* (рис. 4.32, *а*), між трьома послідовними незалежними блоками: $ФБ_1$, $ФБ_2$ і $ФБ_3$, причому так, щоб максимальний час обробки в кожному $ФБ_i$ був однаковим і дорівнював $T_{max}/3$. Між блоками розмістимо буферні регістри $Рг_i$, (рис. 4.32, *б*), призначені для зберігання результату обробки у $ФБ_i$, на випадок, якщо наступний за ним функціональний блок ще не готовий використовувати цей результат. В розглянутій схемі дані на вхід конвеєра можуть подаватися з інтервалом $T_{max}/3$ (втричі частіше), і хоча затримка від моменту надходження першої одиниці даних в $Рг_{вх}$ до моменту появи результату її обробки на виході $Рг_{вих}$ як і раніше складає T_{max}, подальші результати з'являються на виході $Рг_{вих}$ вже з інтервалом $T_{max}/3$.

На практиці рідко вдається добитися того, щоб затримки в кожному $ФБ_i$, були однаковими. Як наслідок, продуктивність конвеєра знижується, оскільки період надходження вхідних даних визначається максимальним часом їх обробки в кожному функціональному блоці. Для усунення цього недоліку або принаймні часткової його компенсації кожен буферний регістр $Рг_i$, слід замінити буферною пам'яттю $БП_i$, здатною зберігати багато даних і організованою за принципом FIFO, – «першим увійшов – першим вийшов» (рис. 4.32, *в*). Обробивши елемент даних, $ФБ_i$ заносить результат в $БП_i$, витягає з $БП_{i-1}$ новий елемент даних і приступає до чергового циклу обробки, причому ця послідовність здійснюється кожним функціональним блоком незалежно від інших блоків. Обробка в кожному блоці може продовжуватися до тих пір, поки не ліквідовується попередня черга або поки не переповниться наступна черга. Якщо ємкість буферної пам'яті достатньо велика, відмінності в часі обробки не

позначаються на продуктивності, проте бажано, щоб середня тривалість обробки у всіх $ФБ_i$, була однаковою.

В архітектурі обчислювальних машин можна знайти множину об'єктів, де конвеєризація забезпечує відчутний приріст продуктивності ОМ, але найбільш відчутний ефект досягається під час конвеєризації етапів машинного циклу.

За способом синхронізації роботи ступенів конвеєри можуть бути синхронними і асинхронними. Для традиційних ОМ характерні *синхронні конвеєри*. Зв'язано це, перш за все, з синхронним характером роботи процесорів. *Асинхронні конвеєри* виявляються корисними, якщо зв'язок між ступенями не великий, а довжина сигнальних трактів між різними ступенями сильно різниться. Прикладом використання асинхронних конвеєрів можуть служити систоличні комп'ютери.

4.5.2. Синхронні лінійні конвеєри

Ефективність синхронного конвеєра багато в чому залежить від правильного вибору довжини тактового періоду $T_к$.

Мінімально допустиму $T_к$ можна визначити як суму найбільшого з часів обробки на окремому рівні конвеєра $T_{РМАХ}$ і часу запису результатів обробки в буферні регістри між рівнями конвеєра $T_{БР}$:

$$T_к = T_{РМАХ} + T_{БР}.$$

Через вірогідний «перекос» у надходженні тактуючого сигналу в різні рівні конвеєра попередній вираз слід доповнити ще одним елементом – максимальною величиною «перекосу» $T_{ПК}$:

$$T_к = T_{РМАХ} + T_{БР} + T_{ПК}.$$

Кожний рівень конвеєра може містити багато логічних трактів обробки. $T_к$ визначається найбільш довгими трактами в усіх рівнях конвеєра. Під час розробки конвеєра необхідно враховувати, що для всіх послідовних елементів, які обробляються одним і тим же рівнем, обробка першого елемента може проходити по тракту максимальної довжини, а другого елемента – по мінімальному тракту. Таким чином результат обробки другого елемента може з'явитись на виході рівня до того, як у вихідному регістрі рівня буде запам'ятований попередній результат. Щоб уникнути подібної ситуації, сума $T_{БР} + T_{ПК}$ повинна бути менша мінімального часу обробки в рівні $T_{РМІН}$, звідки

$$T_{БР} = T_{РМІН} - T_{ПК}.$$

Вибір довжини тактового періоду для конвеєра повинен відбуватися з урахуванням співвідношення

$$T_{РМАХ} + T_{БР} + T_{ПК} = T_к = T_{РМАХ} + T_{РМІН} - T_{ПК}.$$

Незважаючи на очевидні переваги вибору $T_к$, рівним нижній межі, звичайно орієнтуються на середнє значення між нижньою і верхньою межами.

Щоб охарактеризувати ефект, який досягається за рахунок конвеєризації обчислень, звичайно використовують три показники: *прискорення*, *ефективність* і *продуктивність*.

Під *прискоренням* розуміється відношення часу обробки без конвеєра і за його наявності. Теоретично найкращий час обробки вхідного потоку з N значень T_{NK} на конвеєрі з K рівнями і тактовим періодом $T_к$ можна визначити виразом

$$T_{NK} = (K + (N-1))T_к .$$

Вираз відображає той факт, що до появи на виході конвеєра результату обробки першого елемента повинно пройти K тактів, а наступні результати будуть проходити в кожному такті.

У процесорі без конвеєра загальний час виконання складає $NKT_к$. Таким чином, *прискорення обчислень* S за рахунок конвеєризації обчислень може описуватись виразом

$$S = \frac{NKT_K}{(K+(N-1))T_K} = \frac{NK}{K+(N-1)}.$$

При $N \to \infty$ прискорення прагне до величини, яка дорівнює кількості рівнів у конвеєрі.

Другою характеристикою конвеєрного процесора є *ефективність* E – доля прискорення, яка приходиться на один рівень конвеєра:

$$E = \frac{S}{K} = \frac{N}{K+(N-1)}.$$

Третя характеристика конвеєрного процесора – *пропускна здатність* або *продуктивність* P – ефективність, яка є діленою на довжину тактового періоду:

$$P = \frac{N}{T_K(K+(N-1))}.$$

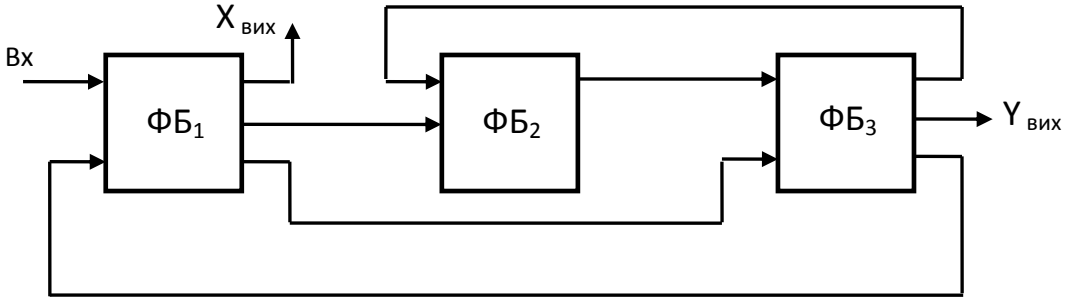
При $N \to \infty$ ефективність прагне до одиниці, а продуктивність – до частоти тактування конвеєра:

$$P = \frac{1}{T_K} = F_K.$$

4.5.3. Нелінійні конвеєри

Конвеєр не завжди є лінійним ланцюжком етапів. У ряді ситуацій виявляється вигідним, коли функціональні блоки з'єднані між собою не послідовно, а відповідно до логіки обробки, при цьому одні блоки в ланцюжку можуть пропускатися, а інші − створювати циклічні структури. Це дозволяє за допомогою одного і того ж конвеєра одночасно обчислювати більше однієї функції, проте якщо ці функції конфліктують між собою, то такий конвеєр важко завантажити повністю. Структура нелінійного конвеєра, що одночасно обчислює дві функції *X* і *Y*, приведена на рис. 4.33. Там же показана послідовність, в якій функції *X* і *Y* застосовують ті або інші функціональні блоки.

Щоб визначити, коли пора приступати до повторного обчислення тієї або іншої функції, необхідно побудувати діаграму одноразової реалізації цієї функції і відстежити по ній моменти, коли такий запуск не приведе до конфлікту, пов'язаного з одночасним зверненням до одного і того ж функціонального блока.



ФБ1	X					X		X
ФБ2		X		X				
ФБ3			X		X		X	

Y				Y	
		Y			
	Y		Y		Y

Рис. 4.33. Нелінійний конвеєр

Так, у ході реалізації функції *X* запуск чергового її обчислення можливий після 1, 3 і 6 тактів. Запуск паралельного обчислення функції *Y* можливий після 2 і 4 тактів. У разі запуску функції *Y* черговий її запуск дозволений після тактів 1, 3 і 5, а паралельний запуск функції *X* дозволений після 2 і 4 тактів.

4.5.4. Конвеєр команд. Конфлікти в конвеєрі команд

Ідея конвеєра команд була запропонована в 1956 році академіком С. А. Лебедєвим. Як відомо, циклом команди є послідовність етапів. Поклавши реалізацію кожного з них на самостійний пристрій і послідовно з'єднавши такі пристрої, ми отримаємо класичну схему конвеєра команд. У циклі команди можна виділити шість етапів:

1. **Вибірка команди** (ВК). Читання чергової команди з пам'яті і занесення її в регістр команди.

2. **Декодування команди** (ДК). Визначення коду операції та способів адресації операндів.

3. **Обчислення адрес операндів** (ОА). Обчислення виконавських адрес кожного з операндів відповідно до вказаного в команді способу їх адресації.

4. **Вибірка операндів** (ВО). Витягання операндів з пам'яті. Ця операція не потрібна для операндів, що знаходяться в регістрах.

5. **Виконання команди** ($В_к$К). Виконання вказаної операції.

6. **Запис результату** (ЗР). Занесення результату в пам'ять.

Використання конвеєра дозволяє значно скоротити час обробки команд. Проте на практиці через конфліктні ситуації, що виникають у конвеєрі, досягти потенційної продуктивності не вдається.

Конфліктні ситуації в конвеєрі прийнято позначати терміном *ризик,* а обумовлені вони можуть бути трьома причинами:

- спробою декількох команд одночасно звернутись до одного й того ж ресурсу комп'ютера (*структурний ризик*);
- взаємозв'язком команд по даних (*ризик по даних*);
- неоднозначністю під час вибірки наступної команди у випадку команд переходу (*ризик по управлінню).*

Структурний ризик має місце, коли декілька команд, які знаходяться на різних рівнях конвеєра, спробують одночасно використати один і той же ресурс, частіше за все – пам'ять. Подібних конфліктів частково удається уникнути за рахунок модульної побудови пам'яті і кеш-пам'яті. В цьому випадку є вірогідність того, що команди будуть звертатись або до різних модулів оперативної пам'яті (ОП), або одна з них стане звертатись до ОП, а інша – до кеш-пам'яті.

Ризик по даних – типова та регулярно виникаюча ситуація. Для пояснення сутності взаємозв'язку команд по даних призначимо, що дві команди в конвеєрі (*i* та *j*) передбачають звертання до однієї й тієї ж змінної *x*, причому команда *i* попереджає команду *j* (рис. 4.34).

В загальному випадку між *i* та *j* можуть бути три типи конфліктів по даних:

а) «*Читання після запису*» (RAW): команда j читає x до того, як команда i встигла записати нове значення x, тобто команда j помилково отримає старе значення x замість нового.

б) «*Запис після читання*» (WAR): команда j записує нове значення x до того, як команда i встигла прочитати x, тобто команда i помилково отримає нове значення x замість старого.

в) «*Запис після запису*» (WAW): команда j записує нове значення x до того, як команда i встигла записати як x власне значення, тобто x помилково містить i-те значення x замість j-го.

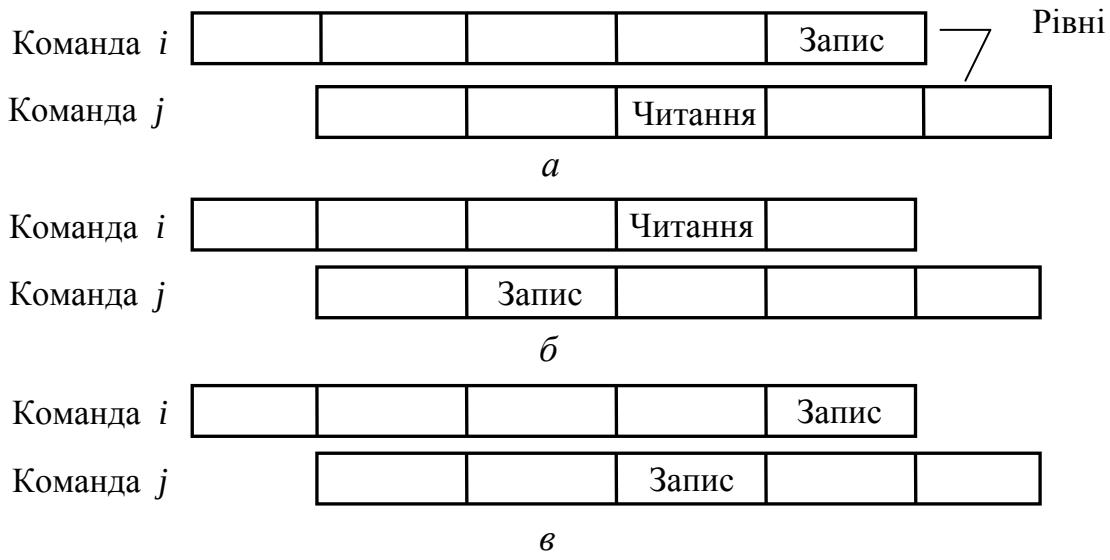


Рис. 4.34. Конфлікти по даних: *а* – «читання після запису»; *б* – «запис після читання»; *в* – «запис після запису»

Для боротьби з конфліктами по даних використовують як програмні, так і апаратні методи.

Програмні методи орієнтовані на усунення самої можливості конфліктів ще на стадії компіляції програми. Оптимізуючий компілятор намагається створити такий об'єктний код, щоб між командами, які схильні до конфліктів, знаходилась достатня кількість нейтральних у цьому плані команд. Якщо таке не вдається, то між командами, які конфліктують, компілятор вставляє необхідну кількість команд типу «Немає операції».

Фактичне розв'язання конфліктів покладається на апаратні методи. Найбільш ефективним рішенням є зупинка команди j на декілька тактів з тим, щоб команда i встигла завершитися або принаймні минути рівень конвеєра, який викликав конфлікт. Таку ситуацію називають «бульбашкою» в конвеєрі.

Зрозуміло, що зупинки конвеєра знижують його ефективність і розробники ОМ всіляко прагнуть скоротити загальне число зупинок або хоч би їх тривалість. Оскільки найбільш часті конфлікти по даних RAW, основні зусилля витрачаються на протидію саме цьому типу конфліктів. Серед відомих методів боротьби з RAW найбільшого поширення набув прийом *прискореного просування інформації* (forwarding). Зазвичай між двома сусідніми рівнями конвеєра розташовується буферний регістр, через який попередній рівень передає результат своєї роботи на подальший рівень, тобто передача інформації можлива лише між сусідніми рівнями конвеєра. У разі прискореного просування, коли для виконання команди потрібний операнд, вже обчислений попередньою командою, цей операнд може бути отриманий

безпосередньо з відповідного буферного регістра, минувши всі проміжні рівні конвеєра. З даною метою в конвеєрі передбачаються додаткові тракти пересилки інформації (тракти випередження, тракти обходу), які забезпечені засобами мультиплексування.

Найбільші проблеми під час створення ефективного конвеєра обумовлені командами, що змінюють природний порядок обчислень. Простий конвеєр орієнтований на лінійні програми. В ньому ступінь вибірки витягує команди з послідовних елементів пам'яті, використовуючи для цього лічильник команд (ЛчК). Адреса чергової команди в лінійній програмі формується автоматично, за рахунок додавання до вмісту ЛчК числа, рівного довжині поточної команди в байтах. Реальні програми практично ніколи не бувають лінійними. В них обов'язково присутні команди управління, які змінюють послідовність обчислень: безумовний та умовний перехід, виклик процедури, повернення з процедури і тому подібне Частка подібних команд у програмі оцінюється як 10-20% (за деякими джерелами вона істотно більша) [25]. Виконання команд, які змінюють послідовність обчислень (надалі їх називатимемо командами переходу), може приводити до припинення конвеєра на декілька тактів, через що продуктивність процесора знижується. Припинення конвеєра під час виконання команд переходу обумовлено двома чинниками.

Перший чинник характерний для будь-якої команди переходу і пов'язаний з вибіркою команди з точки переходу (за адресою, вказаною в команді переходу). Те, що поточна команда відноситься до команд переходу, стає ясним тільки після декодування (після проходження рівня декодування), тобто після двох тактів з моменту надходження команди на конвеєр. За цей час на перші рівні конвеєра вже поступлять нові команди, які вже витягнуті в припущенні, що природний порядок обчислень не буде порушений. У разі переходу ці рівні потрібно очистити і завантажити в конвеєр команду, розташовану за адресою переходу, для чого потрібна виконавська адреса останньої. Оскільки в командах переходу зазвичай вказані лише спосіб адресації і адресний код, виконавська адреса заздалегідь повинна бути обчисленою, що і робиться на третьому рівні конвеєра. Таким чином, реалізація переходу в конвеєрі вимагає певних додаткових операцій, виконання яких рівносильно зупинці конвеєра як мінімум на два такти.

Друга причина порушення ритмічності роботи конвеєра має відношення тільки до команд умовного переходу. В разі появи команди умовного переходу, конвеєр повинен бути очищений від непотрібних команд, що виконувалися до даного моменту і ще протягом декількох тактів (ДК, ОА, ВО, $В_кК$) не буде завершеною жодна інша команда. Це і є витрати через неможливість передбачення результату команди умовного переходу. Як видно, вони або істотно більші, ніж для інших команд переходу (якщо перехід відбувається), або відсутні зовсім (якщо перехід не відбувається).

Для скорочень затримок, обумовлених вибіркою команди з точки переходу, застосовуються декілька підходів:

- обчислення виконавчої адреси переходу на рівні декодування команди;
- використання буфера адрес переходу;
- використання кеш-пам'яті для зберігання команд, розташованих у точці переходу;
- використання буфера циклу.

У результаті декодування команди з'ясовується не тільки її приналежність до команд переходу, але також спосіб адресації і адресний код точки переходу. Це дозволяє відразу ж приступити до обчислення виконавської адреси переходу, не чекаючи передачі команди на третій рівень конвеєра, і тим самим скоротити час зупинки конвеєра з двох тактів до одного. Для реалізації цієї ідеї до складу рівня декодування вводяться додаткові суматори, за допомогою яких і обчислюється виконавська адреса точки переходу.

Буфером адрес переходу (BTB, Branch Target Buffer) є кеш-пам'ять невеликої ємності, в якій зберігаються виконавські адреси точок переходу декількох останніх команд, для яких перехід мав місце. Перед вибіркою чергової команди її адреса (вміст лічильника команд) порівнюється з адресами команд, поданих у BTB. Для команди, знайденої в буфері адрес переходу, виконавська адреса точки переходу не обчислюється, а береться з BTB, завдяки чому вибірка команди з точки переходу може бути почата на один такт раніше. Команда, посилання на яку у BTB відсутнє, обробляється стандартним чином. Якщо це команда переходу, то отримана під час її виконання виконавська адреса точки переходу заноситься у BTB, за умови, що команда завершилася переходом.

Застосування BTB дає найбільший ефект, коли окремі команди переходу в програмі виконуються багато разів, що типово для циклів.

Кеш-пам'ять команд, розташованих у точці переходу (BTIC, Branch Target Instruction Cache), – це вдосконалений варіант BTB, де в кеш-пам'ять крім виконавської адреси команди в точці переходу записується також і код цієї команди. За рахунок збільшення ємності кеш-пам'яті BTIC дозволяє у разі повторного виконання команди переходу виключити не тільки етап обчислення виконавської адреси точки переходу, але і етап вибірки розташованої там команди. Переваги даного підходу найбільшою мірою виявляються у разі багатократного виконання одних і тих же команд переходу, головним чином під час реалізації програмних циклів.

Буфер циклу є маленькою швидкодіючою пам'яттю, що входить до складу першого рівня конвеєра, де проводиться вибірка команд. У буфері зберігаються коди n останніх команд у тій послідовності, в якій вони вибиралися. Коли має місце перехід, апаратура спочатку перевіряє, чи немає потрібної команди в

буфері, і якщо це так, то команда витягується з буфера. Стратегія найбільш ефективна під час реалізації циклів та ітерацій, чим і пояснюється назва буфера Якщо буфер достатньо великий, щоб охопити все тіло циклу, вибірку команд з пам'яті досить виконати тільки один раз у першій ітерації, оскільки необхідні для подальших ітерацій команди вже знаходяться в буфері.

За принципом використання буфер циклу схожий на BTIC, з тією різницею, що в ньому зберігається послідовність виконання команд, а сам буфер менший по ємності і дешевший.

Серед ОМ, де реалізований буфер циклу, можна згадати деякі обчислювальні машини фірми CDC (Star 100, 6600, 7600) і супер-ЕОМ CRAY-1.

4.5.5. Методи вирішення проблеми умовного переходу

Проблеми умовних переходів приводять до найбільших витрат у роботі конвеєра. Для усунення або часткового скорочення цих витрат існують різні способи, які умовно можна розділити на чотири групи:

- буфери передвибірки;
- множинні потоки;
- затриманий перехід;
- передбачення переходу.

Рівномірність надходження команд на вхід конвеєра часто порушується, наприклад через зайнятість пам'яті або під час вибірки команд, які складаються з декількох слів. Щоб добитися ритмічної подачі команд на вхід конвеєра, між рівнем вибірки команди та іншою частиною конвеєра часто розміщують буферну пам'ять, яка організована за принципом черги (FIFO) і має назву *буфер передвибірки.*

Буфер передвибірки можна розглядати як декілька додаткових рівнів конвеєра. Подібне подовження конвеєра не позначається на його продуктивності (при заданій тактовій частоті); воно лише приводить до збільшення часу проходження команди через конвеєр. Типові буфери передвибірки у відомих процесорах розраховані на 2-8 команд.

Щоб одночасно з забезпеченням ритмічної роботи конвеєра розв'язати і проблему умовного переходу, в конвеєр включають два таких буфери (рис 4.35).

Кожна витягнута з пам'яті і поміщена в основний буфер команда аналізується блоком переходу. У разі виявлення команди умовного переходу (УП) блок переходу обчислює виконавчу адресу точки переходу і паралельно з продовженням послідовної вибірки в основний буфер організовує вибірку в додатковий буфер команд, починаючи з точки УП. Далі блок переходу визначає результат команди УП, залежно від якого підключає до залишку конвеєра потрібний буфер, при цьому вміст іншого буфера скидається.

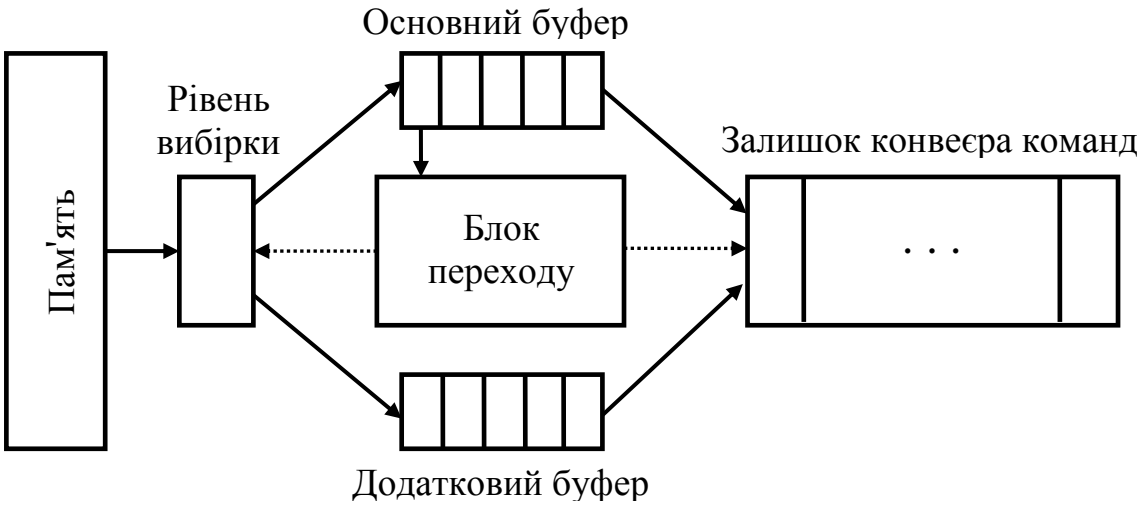


Рис. 4.35. Конвеєр з буферами передвибірки команд

Крім необхідності дублювання частини обладнання, у методу є ще один недолік. Так, якщо в потоці команд декілька команд УП проходять одна за одною або знаходяться достатньо близько, кількість можливих галужень збільшується і, відповідно, повинно бути збільшено і число буферів передвибірки.

Іншим рішенням проблеми переходів служить дублювання початкових рівнів конвеєра і створення тим самим двох ***паралельних потоків команд*** (рис 4.36).

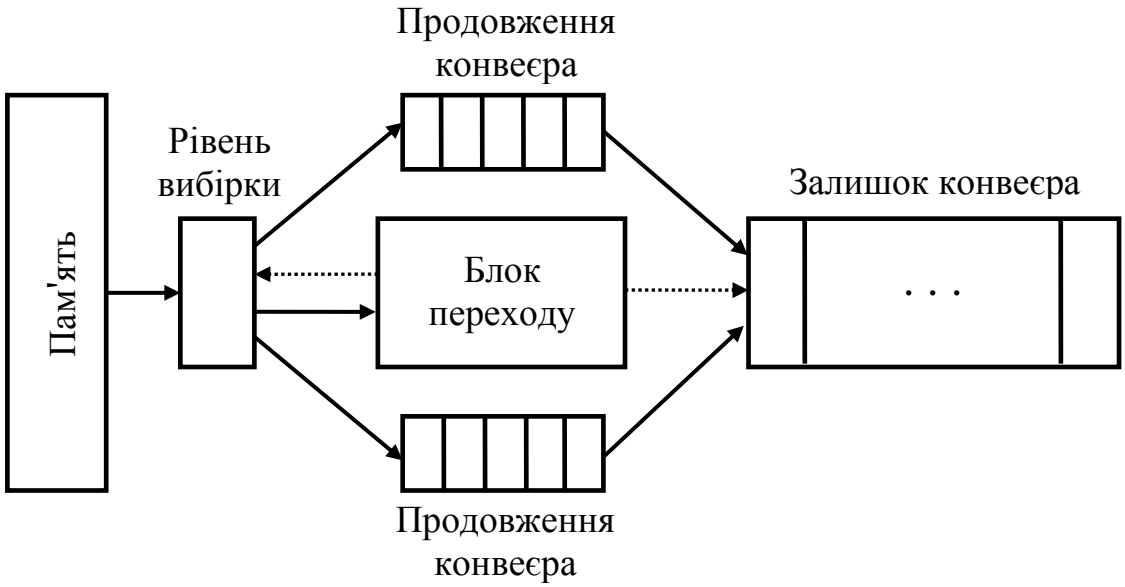


Рис. 4.36. Конвеєр з множинними потоками

В одній гілці такого конвеєра послідовність вибірки і виконання команд відповідає випадку, коли умова переходу не виконалась, у другій гілці – випадку виконання цієї умови. Обидва потоки сходяться в точці, де результат перевірки умови переходу стає очевидним. Подальше просування по конвеєру продовжує тільки «правильний» потік. Основний недолік методу полягає в

тому, що на конвеєр або в потік може поступити нова команда УП до того, як буде ухвалено остаточне рішення по поточній команді переходу. Кожна така команда вимагає додаткового потоку. Незважаючи на це, стратегія дозволяє поліпшити продуктивність конвеєра.

Стратегія ***затриманого переходу*** передбачає продовження виконання команд, які йдуть за командою УП, незалежно від її результату. Природно, що це має сенс, коли наступні команди є «корисними», тобто такими, які все одно повинні бути колись виконані, незалежно від того, відбувається перехід або ні, і якщо команда переходу ніяк не впливає на результат їх виконання.

Для реалізації цієї ідеї на етапі компіляції програми після кожної команди переходу вставляється команда «Немає операції». Потім на стадії оптимізації програми проводяться спроби «перемішати» команди так, щоб по можливості більшу кількість команд «Немає операції» замінити «корисними» командами програми. Зрозуміло, заміщати команду «Немає операції» можна лише на таку, яка не впливає на умову виконуваного переходу, інакше отримана послідовність команд не буде функціонально еквівалентною початковій. В оптимізованій програмі вдається замінити більше 20% команд «Немає операції» [25].

Передбачення переходів на сьогоднішній день розглядається як один з найбільш ефективних способів боротьби з конфліктами по управлінню. Ідея полягає в тому, що ще до моменту виконання команди умовного переходу або відразу ж після її надходження на конвеєр робиться припущення про найбільш вірогідний результат такої команди (перехід станеться або ні). Наступні команди надходять на конвеєр відповідно до передбачення. У разі помилкового передбачення конвеєр необхідно повернути до стану, з якого почалася вибірка «непотрібних» команд (очистити початкові рівні конвеєра), і перейти до завантаження, починаючи з «правильної» точки, що по ефекту еквівалентно призупиненню конвеєра. Ціна помилки може стати достатньо високою, але за правильних передбачень виграш буде більшим.

Серед механізмів передбачення переходів розрізняють *статичне* та *динамічне* передбачення переходів.

Статичне передбачення переходів здійснюється на основі апріорної інформації про програму, яка буде виконуватись. Передбачення робиться на етапі компіляції програми і в процесі обчислень вже не змінюється. Головна різниця між відомими механізмами статичного прогнозування полягає у виді інформації, яка використовується для передбачення, та її трактовці. Вихідна інформація може бути отримана двома шляхами: на основі аналізу коду програми або в результаті її «профілювання». Під «профілюванням» розуміють виконання програми при деякому еталонному наборі вихідних даних, яке супроводжується збором інформації про результати кожної команди умовного переходу. Вибір команд, які завершились переходом, фіксується в спеціальному

біті коду операції. У ході виконання програми поведінка конвеєра команд визначається після вибірки команди по стану згаданого біта в коді операції.

У *динамічних стратегіях* рішення про найбільш вірогідний результат команди умовного переходу приймається в ході обчислень, на основі інформації про попередні переходи (історії переходів), яка збирається в процесі виконання програми. В цілому, динамічні стратегії у порівнянні зі статичними забезпечують більш високу точність передбачення.

Ідея динамічного передбачення переходів передбачає накопичення інформації про результат виконання попередніх команд УП. Історія переходів фіксується у формі таблиці, кожний елемент якої складається з m бітів. Потрібний елемент таблиці вказується за допомогою k- розрядної двійкової комбінації – шаблона. Цим пояснюється загальноприйнята назва таблиці передісторії переходів – *таблиця історії для шаблонів* (PHT, Pattern History Table). Як шаблони для доступу до PHT можуть бути взяті:

- адреса команди умовного переходу;
- регістр глобальної історії;
- регістр локальної історії;
- комбінація попередніх варіантів.

Схема, де для доступу до PHT вибрана *адреса команди УП* (вміст лічильника команд), дозволяє враховувати поведінку кожної конкретної команди УП. Кожній команді умовного переходу в PHT відповідає свій лічильник. Коли команда закінчується переходом, вміст лічильника збільшується на одиницю, а інакше – зменшується на одиницю. Як шаблон для пошуку в PHT служать молодші k розрядів вмісту лічильника команд.

Регістр глобальної історії (GHR, Global History Register) являє собою l - розрядний регістр зсуву. Після виконання чергової команди УП вміст регістру зсовується на один розряд, а у звільнену позицію заноситься одиниця (якщо був перехід) або нуль (якщо переходу не було). Отже, кодова комбінація в GHR відображає історію виконання останніх l команд умовного переходу. Як шаблон для пошуку в PHT служать молодші k розрядів GHR (частіше $l = k$).

Регістр локальної історії (LHR, Local History Register) по логіці роботи аналогічний регістру глобальної історії, але призначений для фіксації результатів однієї й тієї ж команди УП. У схемах передбачення з LHR присутня так звана *таблиця локальної історії*, яка являє собою масив регістрів локальної історії. Як і в схемі з адресою команди переходу, кожний лічильник в PHT фіксує історію результату тільки однієї команди УП, але базується на більш детальних знаннях, які відображають ще і послідовність результатів.

Розмір PHT впливає на точність передбачень – зі збільшенням розміру PHT точність передбачень зростає. Типова кількість елементів PHT (елементарних лічильників) у різних процесорах варіюється від 256 до 4096.

4.5.6. Суперконвеєрні процесори

Ефективність конвеєра знаходиться в прямій залежності від того, з якою частотою на його вхід подаються об'єкти обробки. Добитись n - кратного збільшення темпу роботи конвеєра можна двома шляхами:

- розбиттям кожного рівня конвеєра на n «підрівнів» у разі одночасного збільшення тактової частоти усередині конвеєра також в n разів;
- включенням у склад процесора n конвеєрів, які працюють з перекриттям.

Перший з цих підходів має назву *суперконвеєризація*. По суті суперконвеєризація зводиться до збільшення кількості рівнів конвеєра як за рахунок додавання нових рівнів, так і шляхом ділення рівнів, що є, на декілька простих підрівнів. Головна вимога – можливість реалізації операції в кожному підрівні найбільш простими технічними засобами, а значить, з мінімальними витратами часу. Другою не менш важливою умовою є однаковість затримки в усіх підрівнях.

У табл. 4.2 показана довжина конвеєра команд у популярних мікропроцесорах.

Таблиця 4.2

Довжина конвеєра команд в популярних мікропроцесорах

Тип мікропроцесора	Кількість рівнів в конвеєрі команд
MIPS R4400	8
UltraSPARC I	9
Pentium III	10
Itanium	10
UltraSPARC III	14
Pentium 4	20

Критерієм для зачислення процесора до суперконвеєрних служить число рівнів у конвеєрі. До суперконвеєрних відносять процесори, де таких рівнів більше шести. Першим серійним суперконвеєрним процесором вважається MIPS R4400, конвеєр якого включає вісім рівнів.

Разом з виграшем суперконвеєризація має і недоліки. У довгому конвеєрі зростає вірогідність конфліктів. Дорожче стає помилка передбачення переходу – доводиться очищати більше число рівнів конвеєра, а це потребує більше часу. Ускладнюється логіка взаємодії рівнів конвеєра. Однак виробникам ОМ вдається вирішувати більшість з перерахованих проблем, про що свідчить зростання кількості рівнів у конвеєрах команд сучасних процесорів.

4.5.7. Архітектури з повним і скороченим набором команд

Сучасна технологія програмування орієнтована на мови високого рівня (МВР), головне завдання яких – полегшити процес написання програм. Більше 90% всього процесу програмування здійснюють на МВР. На жаль, операції, характерні для МВР, відрізняються від операцій, що реалізуються машинними командами. Ця проблема отримала назву *семантичного розриву* і веде вона до недостатньо ефективного виконання програм. Намагаючись подолати семантичний розрив, розробники ОМ розширюють систему команд, доповнюючи її командами, що реалізують складні оператори МВР на апаратному рівні, вводять додаткові види адресації і т. п. Обчислювальні машини, де реалізовані ці засоби, прийнято називати ОМ з *повним набором команд* (CISC − Complex Instruction Set Computer). До типу CISC можна віднести практично всі ОМ, що випускалися до середини 80-х років минулого століття, і значну частину з тих, що випускаються в даний час.

Характерні для CISC способи вирішення проблеми семантичного розриву разом з тим ведуть до ускладнення архітектури ОМ, головним чином, пристроїв управління, що, в свою чергу, негативно позначається на продуктивності в цілому. Крім того, в CISC дуже складно організувати ефективний конвеєр команд, який, як уже наголошувалося, є одним з найбільш перспективних шляхів підвищення продуктивності ОМ. Аналіз програм, що отримуються після компіляції з МВР, дозволив виявити цікаві закономірності [25]:

- Реалізація складних команд, еквівалентних операторам МВР, вимагає збільшення ємності управляючої пам'яті в мікропрограмному ПУ. Мікропрограми складних команд можуть займати до 60% управляючої пам'яті, тоді як їх частка в загальному об'ємі програми часто не перевищує 0,2%.
- У програмі, що відкомпілювалася, оператори МВР реалізуються у вигляді процедур (підпрограм), тому на операції виклику процедури і повернення з неї доводиться від 15 до 45% обчислювального навантаження.
- Під час виклику процедури викликаюча програма передає цій процедурі деяку кількість аргументів. Згідно з [25] в 98% випадків кількість аргументів, що передаються, не перевищує шести. Приблизно таке ж положення склалося і з параметрами, які процедура повертає викликаючій програмі. Більше 80% змінних, використовуваних програмою, є локальними, тобто створюються під час входу в процедуру і знищуються під час виходу з неї. Кількість локальних змінних, що створюються окремою процедурою, в 92% випадків не перевищує шести.
- Майже половину операцій у ході обчислень складає операція привласнення, що зводиться до пересилки даних між регістрами, елементами пам'яті або регістрами і пам'яттю.

Детальний аналіз результатів досліджень привів до серйозного перегляду традиційних архітектурних рішень, наслідком чого стала поява *архітектури зі скороченим набором команд* (RISC - Reduced Instruction Set Computer).

Основні риси RISC-архітектури

Головні зусилля в архітектурі RISC направлені на побудову максимально ефективного конвеєра команд, тобто такого, де всі команди витягуються з пам'яті і поступають в ЦП на обробку у вигляді рівномірного потоку, причому жодна команда не повинна знаходитися в стані очікування, а ЦП повинен залишатися завантаженим протягом всього часу. Крім того, ідеальним буде варіант, коли будь-який етап циклу команди виконується протягом одного тактового періоду.

Останню умову відносно просто можна реалізувати для етапу вибірки. Необхідно лише, щоб усі команди мали стандартну довжину, яка дорівнює ширині шини даних, що зє'днує ЦП і пам'ять. Уніфікація часу виконання для різних команд – значно складніше завдання, оскільки разом з регістровими існують також команди із зверненням до пам'яті.

Крім однакової довжини команд, важливо мати відносно просту підсистему декодування і управління: складний пристрій управління вноситиме додаткові затримки до формування сигналів управління. Очевидний шлях істотного спрощення ПУ є скорочення числа виконуваних команд, форматів команд і даних, а також видів адресації.

Необхідно підкреслити, що в скороченому списку команд повинні залишатися ті, які використовуються найчастіше. Дослідження показали, що 80-90% часу виконання типових програм приходиться на відносно малу частину команд (10-20%). До найчастіше використовуваних дій відносяться пересилка даних, арифметичні та логічні операції. Основна причина, що перешкоджає зведенню всіх етапів циклу команди до одного тактового періоду, – потенційна необхідність доступу до пам'яті для вибірки операндів і/або запису результатів. Слід максимально скоротити число команд, що мають доступ до пам'яті. Це міркування додає до раніше згаданих принципів RISC ще два:

- доступ до пам'яті під час виконання здійснюється тільки командами «Читання» і «Запис»;
- всі операції, окрім «Читання» і «Запис», мають тип «регістр– регістр».

Для спрощення виконання більшості команд і приведення їх до типу «регістр–регістр» потрібно забезпечити ЦП значним числом регістрів загального призначення. Велике число регістрів у регістровому файлі ЦП дозволяє забезпечити тимчасове зберігання проміжних результатів, використовуваних як операнди в подальших операціях, і веде до зменшення числа звернень до пам'яті, прискорюючи виконання операцій. Мінімальне

число регістрів, рівне 32, прийняте як стандарт дефакто більшістю виробників RISC-комп'ютерів.

Підсумовуючи сказане, концепцію RISC-комп'ютера можна звести до таких положень:

- виконання всіх (або принаймні 75% команд) за один цикл;
- стандартна однослівна довжина всіх команд, яка дорівнює природній довжині слова і ширині шини даних та допускає уніфіковану потокову обробку всіх команд;
- мале число команд (не більше 128);
- мала кількість форматів команд (не більше 4);
- мале число способів адресації (не більше 4);
- доступ до пам'яті тільки за допомогою команд «Читання» і – «Запис»;
- всі команди, за винятком «Читання» і «Запис», використовують внутрішньопроцесорні межрегістрові пересилки;
- пристрій управління з «жорсткою» логікою;
- відносно великий (не менше 32) процесорний файл регістрів загального призначення (згідно з [25] число РЗП в сучасних RISC-мікропроцесорах може перевищувати 500).

Регістри в RISC-процесорах. Відмітна риса RISC-архітектури – велике число регістрів загального призначення, що пояснюється прагненням звести всі пересилки до типу «регістр–регістр». Але збільшення числа РЗП здатне дати ефект лише у разі розумного їх використання. Оптимізація використання регістрів в RISC-процесорах забезпечується як програмними, так і апаратними засобами.

Програмна оптимізація виконується на етапі компіляції програми, написаної на МВР. Компілятор прагне так розподілити регістри процесора, щоб розмістити в них ті змінні, які протягом заданого періоду часу використовуватимуться найінтенсивніше.

На початковому етапі компілятор виділяє кожній змінній віртуальний регістр. Число віртуальних регістрів у принципі не обмежене. Потім компілятор відображає віртуальні регістри на обмежену кількість фізичних регістрів. Віртуальні регістри, використання яких не перекривається, відображаються на один і той же фізичний регістр. Якщо в певному фрагменті програми фізичних регістрів не хватає, то їх роль для віртуальних регістрів, що залишилися, виконують елементи пам'яті. В ході обчислень вміст кожної такої комірки за допомогою команди «Читання» тимчасово засилається в регістр, після чого командою «Запис» знов повертається в комірку пам'яті.

Завдання оптимізації полягає у визначенні того, яким змінним у даній точці програми найвигідніше виділити фізичні регістри. Найбільш поширений метод, вживаний для цієї мети, відомий як *розфарбовування графа*.

У загальному випадку метод формулюється таким чином. Є граф, що складається з вузлів і ребер. Необхідно розфарбувати вузли так, щоб сусідні вузли мали різний колір і щоб при цьому загальна кількість залучених кольорів була мінімальною. В нашому випадку роль вузлів виконують віртуальні регістри. Якщо два віртуальні регістри одночасно присутні в одному і тому ж фрагменті програми, вони з'єднуються ребром. Робиться спроба розфарбувати граф в n кольорів, де n – кількість фізичних регістрів. Якщо така спроба не увінчалася успіхом, то вузлам, які не вдалося розфарбувати, замість фізичних регістрів виділяються комірки в пам'яті.

Апаратна оптимізація використання регістрів в RISC-процесорах орієнтована на скорочення витрат часу під час роботи з процедурами. Найбільший час у програмах, написаних на МВР, витрачається на виклики процедур і повернення з них. Зв'язано це із створенням і обробкою великого числа локальних змінних і констант. Одним з механізмів для боротьби з цим ефектом є так звані *регістрові вікна*. Головне їх завдання – спростити і прискорити передачу параметрів від процедури, яка викликає, до тієї, яку викликає, і назад.

Регістровий файл розбивається на групи регістрів, які називають вікнами. Окреме вікно призначається глобальним змінним. Глобальні регістри доступні всім процедурам, що виконуються в системі у будь-який час. З іншого боку, кожній процедурі виділяється окреме вікно в регістровому файлі. Всі вікна мають однаковий розмір (зазвичай по 32 регістри) і складаються з трьох полів (рис. 4.37).

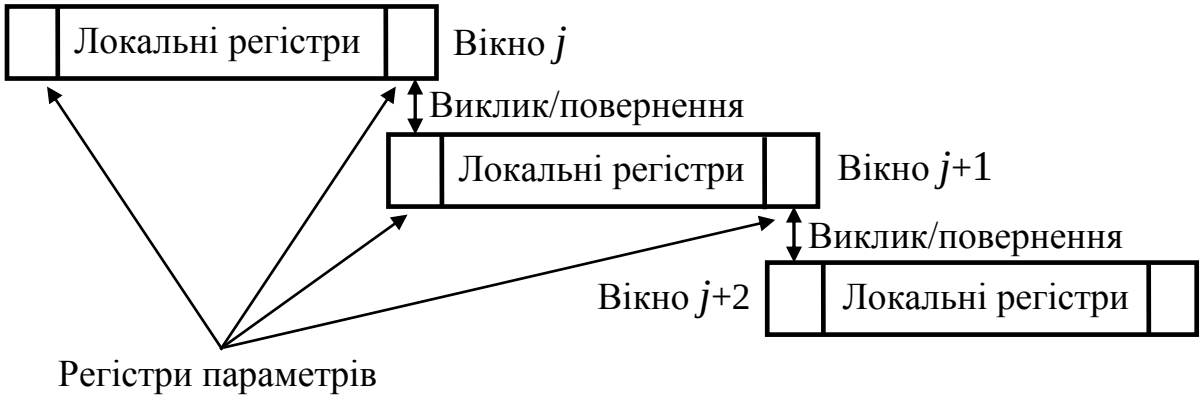


Рис. 4.37. Перекриття регістрових вікон

Ліве поле кожного регістрового вікна одночасно є і правим полем попереднього вікна. Середнє поле служить для зберігання локальних змінних і констант процедури.

Інша техніка апаратної оптимізації використання регістрів, що часто зустрічається, – додання деяким з них спеціальної якості: такі регістри в змозі приймати на себе ім'я будь-якого РЗП. Набір регістрів, що володіють подібною властивістю, називають *буфером перейменування*. Прийом виявляється дуже

зручним під час конвеєризації команд і дозволяє запобігти конфліктам, коли одна команда хоче скористатися регістром, у даний момент зайнятим іншою командою.

Переваги і недоліки RISC-архітектури. Порівнюючи переваги і недоліки CISC і RISC, неможливо зробити однозначний висновок про незаперечну перевагу однієї архітектури над іншою. Для окремих сфер використання ОМ кращою виявляється та або інша. Проте нижче приводиться основна аргументація «за» і «проти» RISC-архітектури [25].

Для технології RISC характерна порівняно проста структура пристрою управління. Площа, що виділяється на кристалі мікросхеми для реалізації ПУ, істотно менше. Так, у RISC I вона складає 6%, а в RISC II - 10%. Як наслідок, з'являється можливість розмістити на кристалі велике число регістрів ЦП (138 в RISC II). Крім того, залишається більше місця для інших вузлів ЦП і для додаткових пристроїв: кеш-пам'яті, блока арифметики з плаваючою комою, частини основної пам'яті, блока управління пам'яттю, портів вводу/виводу.

Уніфікація набору команд, орієнтація на потокову конвеєрну обробку, уніфікація розміру команд і тривалості їх виконання, усунення періодів очікування в конвеєрі – всі ці чинники позитивно позначаються на загальній швидкодії. Простий пристрій управління має небагато вентилів і, отже, короткі лінії зв'язку для проходження сигналів управління. Невелике число команд, форматів і режимів приводить до спрощення схеми декодування, і воно відбувається швидше. Використовуваний в RISC ПУ з «жорсткою» логікою швидше мікропрограмного. Високій продуктивності сприяє і спрощення передачі параметрів між процедурами. Таким чином, застосування RISC веде до скорочення часу виконання програми або збільшення швидкості за рахунок скорочення числа циклів на команду.

Простота ПУ, що супроводжується зниженням вартості і підвищенням надійності, також говорить на користь RISC. Розробка ПУ займає менше часу. Простий ПУ міститиме менше конструктивних помилок і тому надійніше.

Багато сучасних CISC-комп'ютерів, таких як VAX 11/780, VAX-8600, мають багато засобів для прямої підтримки функцій МВР, найбільш частих у цих мовах (управління процедурами, операції з масивами, перевірка індексів масивів, захист інформації, управління пам'яттю і т. д.). RISC також володіє низкою засобів для безпосередньої підтримки МВР і спрощення розробки компіляторів МВР, завдяки чому ця архітектура в плані підтримки МВР ні в чому не поступається CISC.

Недоліки RISC прямо пов'язані з деякими перевагами цієї архітектури. Принциповий недолік – скорочене число команд: на виконання ряду функцій доводиться витрачати декілька команд замість однієї в CISC. Це подовжує код програми, збільшує завантаження пам'яті і трафік команд між пам'яттю і ЦП.

Дослідження показали, що RISC-програма в середньому на 30% довша за CISC-програму, що реалізує ті ж функції.

Хоч велике число регістрів дає істотні переваги, само по собі воно ускладнює схему декодування номера регістра, тим самим збільшується час доступу до регістрів

ПУ з «жорсткою» логікою, реалізований у більшості RISC-систем, менш гнучкий, більш схильний до помилок, ускладнює пошук і виправлення помилок, поступається під час виконання складних команд.

Однослівна команда виключає пряму адресацію для повної 32-бітової адреси. Тому ряд виробників допускає невелику частину команд подвійної довжини, наприклад в Intel 80960.

4.5.8. Суперскалярні процесори

Оскільки можливості по вдосконаленню елементної бази вже практично вичерпані, подальше підвищення продуктивності ОМ лежить у площині архітектурних рішень. Як уже наголошувалося, один з найбільш ефективних підходів у цьому плані – введення в обчислювальний процес різних рівнів паралелізму. Раніше розглянутий конвеєр команд – типовий приклад такого підходу Тим же цілям служать і арифметичні конвеєри, де конвеєризації піддається процес виконання арифметичних операцій. Додатковий рівень паралелізму реалізується у векторних і матричних процесорах, але тільки під час обробки багатокомпонентних операндів типу векторів і масивів. Тут висока швидкодія досягається за рахунок одночасної обробки всіх компонентів вектора або масиву, проте подібні операнди характерні лише для достатньо вузького круга вирішуваних завдань. Основний об'єм обчислювального навантаження зазвичай приходиться на скалярні обчислення, тобто на обробку поодиноких операндів, таких, наприклад, як цілі числа. Для подібних обчислень додатковий паралелізм реалізується значно складніше, проте він можливий, і прикладом можуть служити суперскалярні процесори.

Суперскалярним (цей термін вперше був використаний у 1987 році) називається центральний процесор (ЦП), який одночасно виконує більш ніж одну скалярну команду. Це досягається за рахунок включення до складу ЦП декількох самостійних функціональних (виконавчих) блоків, кожен з яких відповідає за свій клас операцій і може бути присутнім у процесорі в декількох екземплярах. Структура типового суперскалярного процесора показана на рис. 4.38. Процесор включає шість блоків: *вибірки команд, декодування команд, диспетчеризація команд, розподілу команд по функціональних блоках, блок виконання* і *блок оновлення стану.*

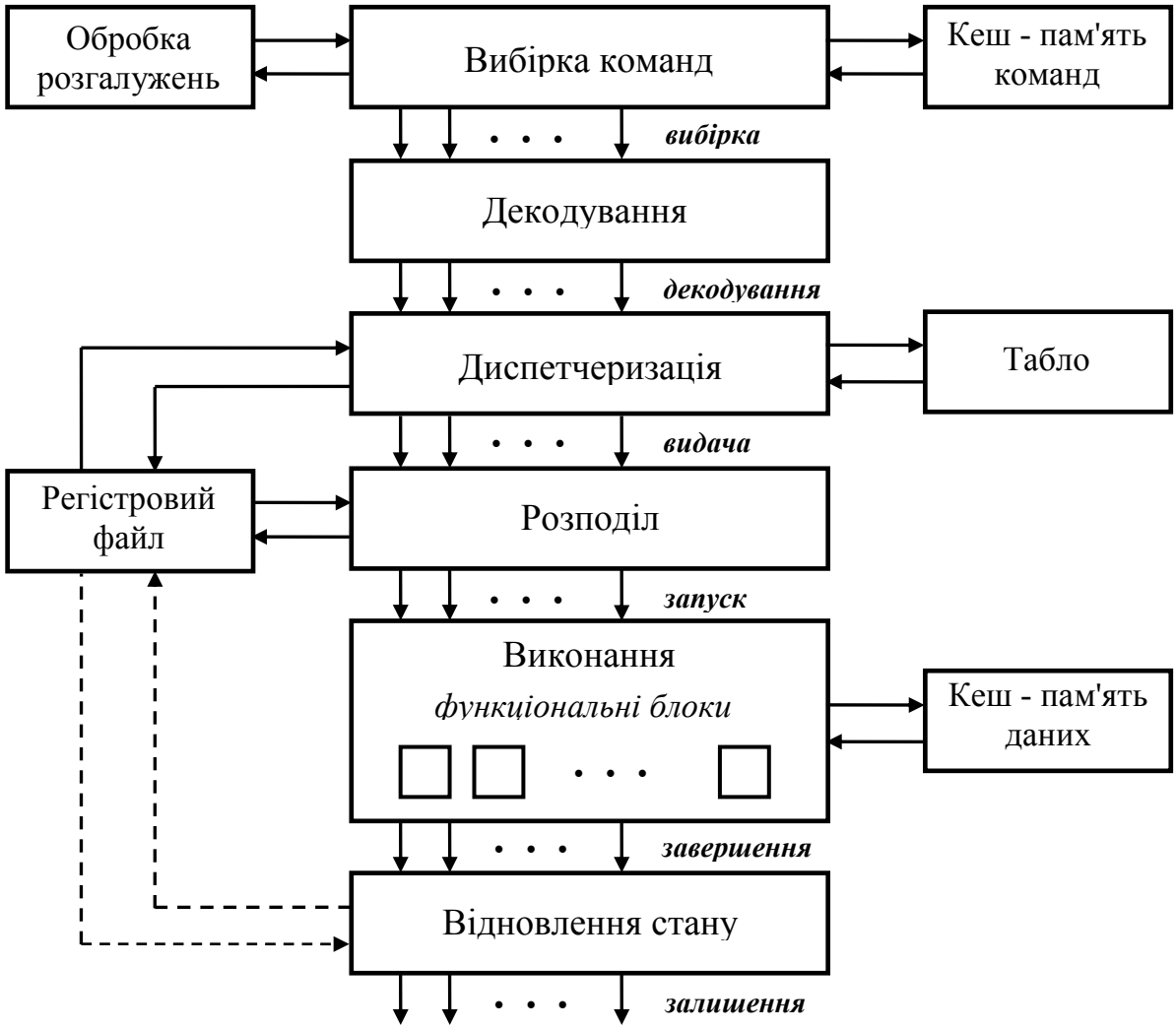


Рис. 4.38. Архітектура суперскалярного процесора

Блок вибірки команд витягує команди з основної пам'яті через кеш-пам'ять команд. Цей блок зберігає декілька значень лічильника команд і обробляє команди умовного переходу.

Блок декодування розшифровує код операції, що міститься у витягнутих з кеш-пам'яті командах. У деяких суперскалярних процесорах, наприклад у мікропроцесорах фірми Intel, блоки вибірки і декодування суміщені.

Блоки диспетчеризації і розподілу взаємодіють між собою і в сукупності відіграють у суперскалярному процесорі роль контролера трафіка. Обидва блоки зберігають черги декодованих команд. Черга блока розподілу часто розосереджується по декількох самостійних буферах – накопичувачах команд або схемах резервування (reservation station), – призначених для зберігання команд, які вже декодовані, але ще не виконані. Кожен накопичувач команд пов'язаний зі своїм функціональним блоком (ФБ), тому число накопичувачів зазвичай дорівнює числу ФБ, але якщо в процесорі використовується декілька однотипних ФБ, то їм додається загальний накопичувач. По відношенню до блока диспетчеризації накопичувачі команд виступають у ролі віртуальних функціональних пристроїв.

На додаток до черги, блок диспетчеризації зберігає також список вільних функціональних блоків, що називаються *табло* (Score-board). Табло використовується для відстеження стану черги *розподілу*. Один раз за цикл блок диспетчеризації витягує команди зі своєї черги, зчитує з пам'яті або регістрів операнди цих команд, після чого, залежно від стану табло, поміщає команди і значення операндів у чергу розподілу. Ця операція називається *видачею команд*. Блок розподілу в кожному циклі перевіряє кожну команду в своїх чергах на наявність усіх необхідних для її виконання операндів і у разі позитивної відповіді починає виконання таких команд у відповідному функціональному блоці.

Блок виконання складається з набору функціональних блоків. Прикладами ФБ можуть служити цілочисельні операційні блоки, блоки множення і додавання з плаваючою комою, блок доступу до пам'яті. Коли виконання команди завершується, її результат записується і аналізується *блоком оновлення стану*, який забезпечує облік отриманого результату тими командами в чергах розподілу, де цей результат виступає як один з операндів.

Як було відмічено раніше, суперскалярність передбачає паралельну роботу максимального числа виконавчих блоків, що можливо лише у разі одночасного виконання декількох скалярних команд. Остання умова добре поєднується з конвеєрною обробкою, при цьому бажано, щоб у суперскалярному процесорі було декілька конвеєрів, наприклад два або три.

Подібний підхід реалізований у мікропроцесорі Intel Pentium, де є два конвеєри, кожен зі своїм АЛП (рис. 4.39).

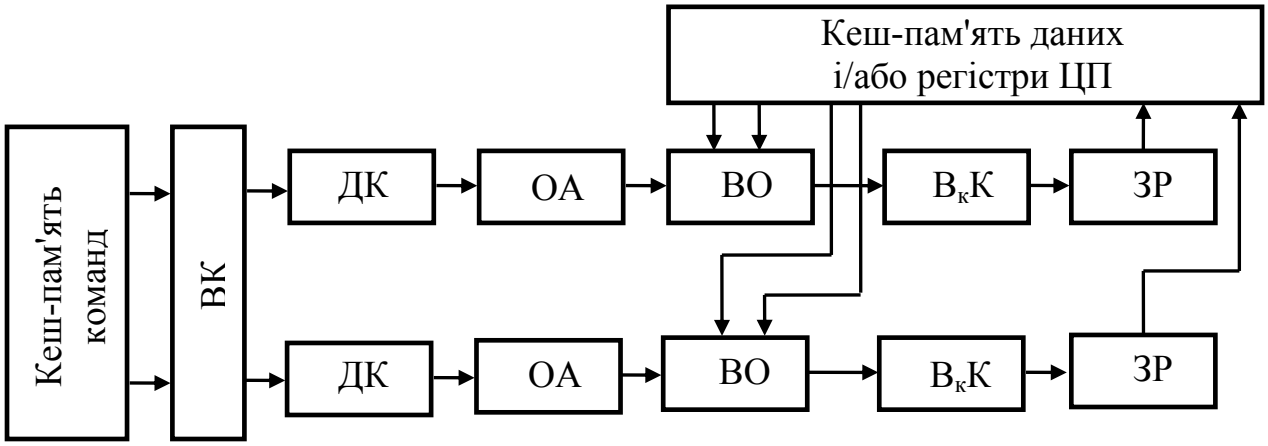


Рис. 4.39. Суперскалярний процесор з двома конвеєрами

Відзначимо, що тут, на відміну від стандартного конвеєра, в кожному циклі необхідно проводити вибірку більш ніж однієї команди. Відповідно, пам'ять ОМ повинна допускати одночасне зчитування декількох команд і операндів, що найчастіше забезпечується за рахунок її модульної побудови.

Більш інтегрований підхід до побудови суперскалярного конвеєра показаний на рис 4.40.

Тут блок вибірки (ВК) витягує з пам'яті більше однієї команди і передає їх через ступені декодування команди і обчислення адрес операндів у блок вибірки операндів (ВО). Коли операнди стають доступними, команди розподіляються по відповідних виконавчих блоках. Звернемо увагу, що операції «Читання», «Запис» і «Перехід» реалізуються самостійними виконавчими блоками.

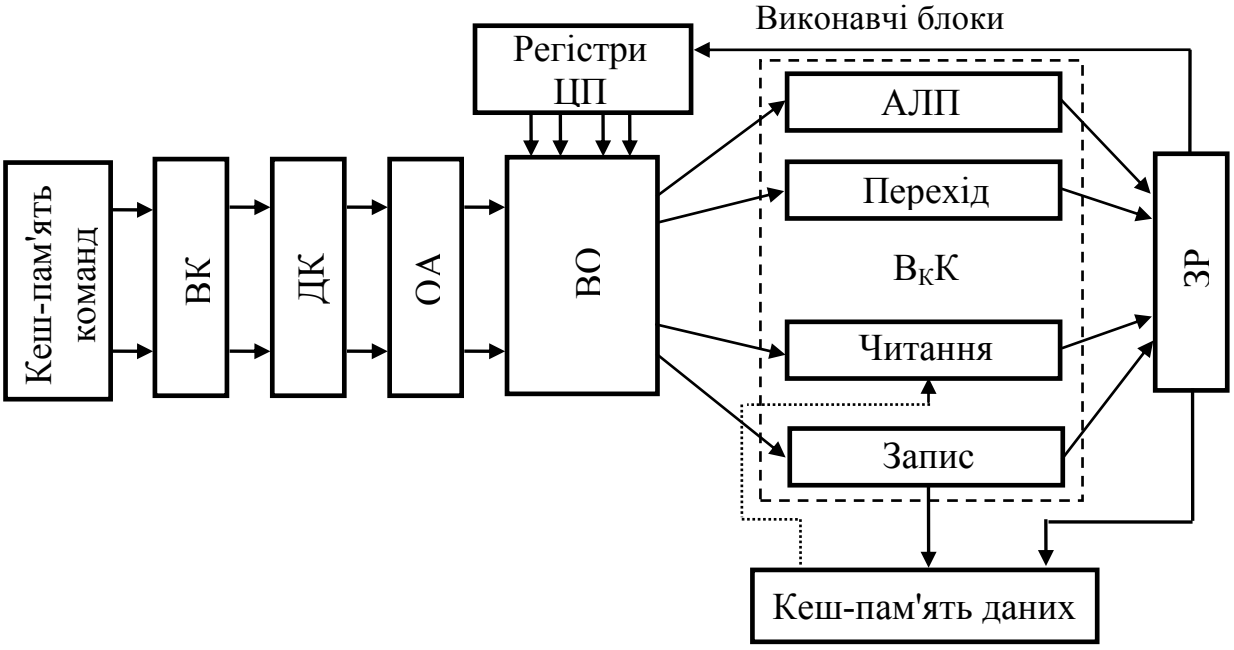


Рис 4.40. Суперскалярний конвеєр із спеціалізованими виконавчими блоками

Подібна форма суперскалярного процесора використовується в мікропроцесорах Pentium II і Pentium III фірми Intel, а форма з трьома конвеєрами – в мікропроцесорі Athlon фірми AMD. За різними оцінками, застосування суперскалярного підходу приводить до підвищення продуктивності ОМ у межах від 1,8 до 8 разів.

У процесорах деяких ОМ реалізовані як суперскалярність, так і суперконвеєризація. Таке поєднання має місце в мікропроцесорах Athlon і Duron фірми AMD, причому охоплює воно не тільки конвеєр команд, але і блок обробки чисел у формі з плаваючою комою.

Таким чином, суперскалярні мікропроцесори є лідируючим продуктом мікроелектроніки, і їх продуктивність постійно зростає.

4.5.9. Логіка роботи суперскалярного мікропроцесора

Попередня вибірка команд і передбачення переходів. Оскільки під час суперскалярної обробки необхідно витягати з пам'яті кілька команд за один такт для завантаження паралельно працюючих функціональних модулів, підвищені вимоги ставляться до пропускної здатності інтерфейсу мікропроцесор – пам'ять. В

сучасних мікропроцесорах застосовуються багаторівневі роздільні кеш-пам'яті даних і команд.

Для зменшення втрат процесорних циклів, зв'язаних із промахами у разі звертання до кеш-пам'яті у випадку виконання команд розгалуження, до складу системи кешування введені засоби передбачення переходів, основне призначення яких – підвищити імовірність наявності в кеш-пам'яті необхідної команди.

Виконання умовних розгалужень складається з таких етапів:

- розпізнавання команди умовного розгалуження;
- перевірка виконання умови переходу;
- обчислення адреси переходу;
- передача керування у випадку переходу.

На кожному етапі використовуються спеціальні прийоми підвищення продуктивності:

1. Для швидкого декодування використовуються або додаткові біти в полі команди, або переддекодування команд під час вибору з кеш-пам'яті команд.

2. Часто, коли команда вже обрана в кеш, умова переходу ще не вирахувана. Щоб не затримувати потік команд, у даному випадку використовується передбачення переходу по одній із декількох можливих схем. Деякі предбачувальники використовують статичну інформацію із двійкового коду програми або таку, що спеціально вироблена компілятором. Наприклад, певні коди операцій частіше виробляють розгалуження, ніж інші коди, або розгалуження більш імовірне (під час організації циклів), або компілятор може встановлювати прапор, який вказує напрямки переходу. Може також використовуватися статистична інформація, що отримана під час трасування програми.

Інші передбачувальники використовують динамічно формовану інформацію в процесі виконання програми. Звичайно це інформація, що стосується історії виконання даного розгалуження, що зберігається в таблиці розгалужень або в таблиці передбачень розгалужень. Таблиця передбачення розгалужень організується за асоціативним принципом, подібно кеш-пам'яті, її елементи доступні за адресою команди, розгалуження якої передбачається. В деяких реалізаціях елемент таблиці передбачення розгалуження є лічильником, значення якого збільшується у разі правильного передбачення і зменшується у разі неправильного. При цьому значення лічильника визначає переважний напрямок розгалужень.

У момент визначення дійсного значення умови розгалуження вноситься зміна в історію розгалуження. Якщо передбачення було невірним, то повинна ініціюватися вибірка правильних команд. Результати команд, що були умовно виконані, повинні бути анульовані.

3. Для визначення адреси розгалуження звичайно потрібно виконати

цілочислове додавання, що додає до поточного значення лічильника команд зсув, заданий у полі команди розгалуження. І хоча це не вимагає додаткових циклів для звертання до регістрів, прискорення обчислення адреси може бути досягнуте завдяки використанню буфера, який містить раніше використані адреси переходів.

Декодування команд і перейменування регістрів. У суперскалярних процесорах множина функціональних блоків поєднується з множиною конвеєрів команд. Таким процесорам властиві всі види залежностей, які характерні для поодиноких конвеєрів, причому положення справ посилюється тим, що конвеєрів декілька. В суперскалярних процесорах одночасна робота декількох конвеєрів стає джерелом додаткових неув'язок, зокрема *проблеми* послідовності надходження команд на виконання і *проблеми* послідовності завершення команд.

Перша проблема виникає, коли послідовність видачі декодованих команд на функціональні блоки відрізняється від послідовності, яка передбачена програмою. Подібна ситуація відома як *неупорядкована видача команд*. Термін *упорядкована видача команд* використовують, коли команди залишають рівні, які стоять до рівня виконання в упорядкованій програмою послідовності. В обох випадках завершення команд звичайно неупорядковане (*неупорядковане завершення команд)*, і це є *друга проблема*. Упорядковане завершення відбувається рідше.

Крім того, для суперскалярних процесорів є характерним ще один фактор – конфлікт по функціональному блоку, коли на нього претендують декілька команд, які надходять з різних конвеєрів.

Щоб забезпечити неупорядковану видачу команд, у конвеєрі необхідно максимально розв'язати рівні декодування і виконання. Це досягається за допомогою буферної пам'яті, яка має назву *вікно команд*. Кожна декодована команда спочатку поміщається у вікно команд. Процесор може продовжувати вибірку і декодування нових команд до повного заповнення буфера. Видача команд з буфера на виконання визначається не послідовністю їх надходження, а мірою готовності. Іншими словами, кожна команда, для якої вже відомі значення усіх операндів і вільний потрібний для її використання функціональний блок, негайно видається з буфера на виконання.

Неупорядковані видача і завершення команд – це додатковий потенціал підвищення продуктивності суперскалярного процесора, але для цього необхідно розв'язати дві проблеми:

- *усунути залежність команд по даних* (мова йде про залежності типу RAW – читання після запису, та WAW – запис після запису), тобто виключити використання як операнда «застарілого» значення регістра і не дозволяти, щоб чергова команда програми через порушення послідовності виконання команд занесла свій результат у регістр ще до того, як це зробила попередня команда;

- *зберегти такий порядок виконання команд*, щоб загальний результат обчислень залишався ідентичним результату, отриманому у разі строгого дотримання програмної послідовності.

Для усунення *залежності команд по даних* використовується прийом, відомий як *перейменування регістрів.* Спосіб розв'язання другої проблеми називають *переупорядковуванням команд*.

Перейменування регістрів. Коли команди видаються та завершуються упорядковано, кожний регістр у будь-якій точці програми містить саме те значення, яке диктується програмою. Вживання стратегій з неупорядкованою видачею та завершенням команд приводить до того, що запис у регістри може відбуватись також неупорядковано і окремі команди, коли вони звертаються до деякого регістра, замість потрібного одержують «застаріле» або «випереджаюче» значення.

Основна ідея перейменування регістрів у тому, що кожний новий результат записується в один з вільних у даний момент додаткових регістрів. При цьому посилання на регістр, що замінюється в усіх наступних командах, відповідним чином корегуються. Програміст складає програму, використовуючи імена логічних регістрів. Число фізичних регістрів апаратного регістрового файла (АРФ) звичайно більше числа логічних. «Зайві» регістри АРФ використовуються в процедурі перейменування для тимчасового зберігання результатів до моменту вирішення конфліктів по даних, після чого значення з регістра тимчасового зберігання переписується на своє «штатне» місце.

Для подолання зайвих RAW і WAW залежностей, що виникають у результаті обмеженості логічних ресурсів (комірок пам'яті, регістрів), використовується механізм динамічного відображення обумовлених текстом програми логічних ресурсів на фізичні регістри мікропроцесора. У разі такого підходу з одним логічним ресурсом може бути зв'язано кілька значень у різних фізичних регістрах, кожне з яких відповідає значенню логічної величини в один з моментів часу послідовного виконання програми.

Коли команда створює нове значення для логічного регістра, фізичний регістр, у якому міститься це значення, одержує ім'я. Наступні команди, що використовують це значення, забезпечуються ім'ям фізичного регістра. Дана процедура називається перейменуванням регістрів.

Номери логічних регістрів динамічно відображаються на номери фізичних регістрів за допомогою *таблиць підстановки*, які оновлюються після декодування кожної команди. Черговий результат записується в новий фізичний регістр, але значення кожного логічного регістра запам'ятовується, завдяки чому легко відновлюється у випадку, якщо виконання команди повинне бути перерване через виникнення виняткової ситуації або неправильного прогнозу напряму умовного переходу.

Перейменування регістрів може бути реалізоване іншим методом – за допомогою *буфера перейменування*. Це буфер з одним входженням для кожної ініційованої на виконання команди і являє собою асоціативний запам'ятовуючий пристрій або набір регістрів з асоціативним доступом. Цей буфер називається таким, що переупорядковує, тому що він використовується також для встановлення порядку команд у разі переривань. Даний буфер можна розглядати як FIFO чергу, що виконана у вигляді кільцевого буфера з покажчиками "початок" і "кінець".

Команди поміщаються в кінець буфера. Після завершення команди її результат заноситься в заздалегідь запропонований їй елемент черги, незалежно від місця в черзі, займаного цим елементом. До моменту досягнення командою початку буфера, якщо вона була виконана, її результат поміщається в регістровий файл, а сама команда віддаляється. Команда, що знаходиться в буфері і не виконана через те, що відсутні значення операнда, залишається в ньому аж до одержання цього значення. Одночасно може вибиратися з черги або поміщатися в неї кілька команд, однак завжди дотримується дисципліна FIFO.

Незалежно від способу перейменування в суперскалярному процесорі усуваються зайві залежності за даними.

Переупорядкування і виконання команд. Після декодування команд і перейменування регістрів команди передаються на виконання. Як уже відмічалося раніше, видача команд у функціональні блоки може виконуватись неупорядковано, по мірі готовності. Після формування для кожної команди упорядкованих трійок, що складаються з коду операції, фізичних операндів – джерела і результату, і розміщення їх у буферах, настає фаза динамічної перевірки готовності значень операндів для виконання команди.

В ідеалі команда готова до виконання, як тільки готові її вхідні операнди. Однак є ряд обмежень, зв'язаних із доступністю фізичних ресурсів, таких як виконавчі пристрої, комутатори і порти регістрових файлів (або буфера, що переупорядковує). Оскільки порядок виконання команд може відрізнятись від порядку, який передбачений програмою, необхідно забезпечити коректність їх операндів (це частково вирішується шляхом перейменування регістрів) і правильну послідовність занесення результатів у регістри регістрового файла. Одним з найбільш розповсюджених прийомів розв'язання цієї проблеми є *переупорядкування команд*. В його основі лежить використання *вікна команд* – буферної пам'яті, куди поміщаються усі команди, які пройшли декодування та перейменування регістрів. Вікно команди забезпечує відстрочення передачі команд на виконання до моменту готовності операндів, а також необхідну черговість завершення команд та загрузку їх результатів у регістри регістрового файла.

Відмітимо, що технологія, яка передбачає схему розподілу готових команд по потрібних для їх виконання функціональних блоках з одночасною перевіркою

їх доступності, має назву *диспетчеризація.*

Ця техніка відома також під назвою *шелвінг* (shelving). Нижче розглядаються два варіанти вікна команд – централізоване і розподілене.

Централізоване вікно команд. Дане вікно реалізується у вигляді так званого *табло* (Scoreboard). Табло – це буферний запам'ятовуючий пристрій, в якому зберігається деяка кількість останніх витягнутих з пам'яті і декодованих команд, а також поточна інформація про доступність ресурсів, що привертаються для їх виконання. Функціями табло є оперативне виявлення команд, для виконання яких уже доступні всі необхідні операнди і ресурси, і видача таких команд на виконання у відповідні функціональні блоки. Табло можна розглядати як систему попередньої диспетчеризації команд, проте воно здійснює контроль виконання команд і після їх видачі.

Всі витягнуті з пам'яті команди відразу ж після їх декодування і, якщо це необхідно, перейменування регістрів заносяться в табло, причому з дотриманням порядку їх проходження в програмі. Фізично табло реалізується на основі асоціативної пам'яті. Кожній команді виділяється одна комірка, яка складається з декількох полів:

- поля операції, де зберігається дешифрований код операції;
- двох полів операндів, що розміщують значення операндів, якщо вони відомі, або інформацію про те, звідки ці операнди повинні бути отримані;
- поля результату, вказуючого регістру, куди повинен бути поміщений результат виконання даної команди;
- поля бітів достовірності.

В табло також зберігається поточна інформація про доступність пристроїв обробки (функціональних блоків).

Функціонування табло тісно пов'язане з роботою буфера перейменування і може бути описане таким чином. Кожна команда після декодування і перейменування регістрів заноситься в чергову вільну комірку табло. Декодований код операції поміщається в поле операції. Якщо команда припускає завантаження результату в регістр, то на цей регістр є посилання в буфері перейменування (БП) і в поле результату заноситься номер входу БП, в якому зберігається останнє посилання на даний регістр. Далі робиться спроба заповнити поля операндів значеннями операндів. Спочатку проводиться пошук потрібного значення в апаратному регістровому файлі (АРФ). Якщо операнд не знайдений, виконується асоціативний пошук посилання на регістр у буфері перейменування. Коли результат вдалий, необхідне значення операнда береться з буфера перейменування. У будь-якому варіанті у разі виявлення достовірного значення операнда поле операнда комірки табло заповнюється знайденим значенням, а відповідний цьому полю біт достовірності встановлюється в одиницю. Якщо ж значення операнда ще не обчислене, то в поле операнда

комірки табло заноситься ідентифікатор входу буфера перейменування, де знаходиться останнє посилання на шуканий регістр, при цьому біт достовірності такого поля скидається в 0.

Оновлення інформації про готовність операндів і доступність функціональ-них пристроїв виконується в кожному циклі процесора.

Команда може бути лічена з табло і видана на виконання лише після того, як будуть занесені значення всіх операндів, і лише за умови, що потрібний для виконання цієї команди функціональний блок (ФБ) вільний. Після завершення команди у ФБ проводиться запис отриманого результату в ту комірку буфера перейменування, на яку вказує поле результату. Одночасно проводиться асоціативний доступ до всіх команд, що зберігаються в табло, і в тих з них, де в полях операндів вказаний ідентифікатор оновленого входу БП, цей ідентифікатор замінюється занесеним у регістр новим значенням, з відповідною корекцією бітів достовірності. Далі завершена команда покидає табло. Видалення команди з табло є підставою для перезапису значення результату даної команди в регістр АРФ і видалення відповідного запису з буфера перейменування.

Розподілене вікно команд. У варіанті розподіленого вікна команд на вході кожного функціонального блока розміщується буфер декодованих команд, що називається *накопичувачем команд* або *схемою резервування* (reservation station). Після вибірки і декодування команди розподіляються по схемах резервування тих ФБ, де команда виконуватиметься. В буфері команда запам'ятовується і по готовності видається в зв'язаний з даним пунктом функціональний блок. Логіка роботи кожного накопичувача аналогічна централізованому вікну команд. Видача відбувається тільки після того, як команда отримає всі необхідні операнди, і за умови, що ФБ вільний. Під час оновлення вмісту буфера перейменування файла проводиться доступ до всіх накопичувачів команд, і в них ідентифікатори оновлених входів замінюються значеннями операндів, що зберігаються в цих входах.

Відзначимо одну особливість даної схеми: не потрібно, щоб операнд був обов'язково занесений у відведений для нього регістр – він може бути прискорено переданий прямо в накопичувач команд для негайного використання або буферизований там для подальшого використання.

Число незалежних команд, які можуть виконуватися одночасно, варіюється від програми до програми, а також у межах кожної програми. В середньому число таких команд рівне 1-3, часом зростаючи до 5–6. Механізм резервування орієнтований на одночасну видачу декількох команд, що, як правило, легко реалізувати з розподіленим, а не централізованим вікном команд, оскільки темп завантаження розподілених буферів зазвичай менше, ніж потенційний темп видачі команд. Пропускна здатність лінії зв'язку між централізованим вікном команд і функціональними блоками повинна бути вище,

ніж у разі розподіленого вікна. Проте для централізованого вікна характерне ефективніше задіювання ємності буфера.

Ємність накопичувача команд у кожному функціональному блоці залежить від очікуваного числа команд для цього блока. Типовий накопичувач розрахований на 1–3 команди.

Для організації вікна виконання використовуються різні методи: однієї черги, багатьох черг або метод станції, що резервує.

Якщо є одна черга, то перейменування регістрів не потрібне, тому що доступність значень операндів може відзначатися бітом резервування, що відповідний кожному регістру. Регістр резервується, коли команда, що його модифікує, призначається на виконання. І регістр звільняється, коли закінчується виконання команди. Якщо для команди ресурси не були зарезервовані, то вона припиняє своє виконання.

У методі багатьох черг кожна черга організується для команд одного типу. Наприклад, черга команд із плаваючою крапкою або черга команд роботи з пам'яттю.

Третій метод допускає використання станції, що резервує, яка складається із сукупності елементів, кожний з яких містить позиції для розташування коду операції, найменування першого операнда, самого першого операнда, ознаки доступності першого операнда, найменування другого операнда, самого другого операнда, ознаки доступності другого операнда і найменування регістра результату. Коли команда завершує виконання і виробляє результат, то найменування результату порівнюється з найменуваннями операндів у станції, що резервує. Якщо в резервуючої станції виявляється команда, яка чекає цього результату, то дані записуються у відповідну позицію і встановлюється ознака їх доступності. Коли в команді доступні всі операнди, ініціюється її виконання. Станція, що резервує, стежить за доступністю операндів. Коли команда під час диспетчеризації потрапляє у резервуючу станцію, всі готові операнди із регістрового файла переписуються в поля цієї команди. Коли всі операнди готові, команда виконується. Інколи резервуюча станція містить не самі операнди, а покажчики на них в регістровому файлі або буфері, що переупорядковує.

Завершення виконання команд. Завершальною фазою виконання команди є фаза зміни стану процесора відповідно до виконаної команди. Призначення цієї фази – збереження послідовної моделі виконання програми, у разі реального рівнобіжного виконання окремих команд і умовного виконання команд розгалу-ження. Для зміни стану процесора застосовуються два основних способи, причому обидва основані на використанні двох станів: стану, зміненого в результаті операції, і стану, необхідного для відновлення.

При першому способі зберігається стан процесора в наборі контрольних точок або у буфері історії обчислень, які, за необхідності, використовуються

для відновлення стану.

Другий спосіб передбачає розгляд логічного (архітектурного) і фізичного стану процесора. Фізичний стан змінюється негайно після завершення чергової команди. Архітектурний стан змінюється тоді, коли відомий результат умовно виконаних команд. Для реалізації цього способу використовується буфер, що переупорядковує: результати з буфера відправляються у файл архітектурних регістрів і пам'ять.

У буфері, що переупорядковує, для кожної команди є відповідне їй значення лічильника команд і значення інших регістрів, які необхідні для коректного обслуговування переривань.

Ефективність використання суперскалярних архітектур обмежують принаймні дві обставини.

По-перше, є обмеження на ступінь паралелізму на рівні команд, навіть якщо застосовується найдосконаліша техніка суперскалярних обчислень. Перше обмеження виникає з умовних переходів. Інше виходить з того, що розмір вікна виконання (число активних команд, що можуть виконуватися паралельно) обмежує можливий властивий програмі паралелізм, тому що не розглядається рівнобіжне виконання команд, що знаходяться на відстані, яка перевищує розмір вікна.

По-друге, складність суперскалярного процесора зростає зі зростанням кількості паралельних команд, що виконуються, і навіть швидше.

Найімовірніше, що межею розпаралелювання у разі суперскалярної обробки є запуск на одночасне виконання в кожному такті 7–8 команд.

Таким чином, суперскалярні мікропроцесори є лідируючим продуктом мікроелектроніки, і їхня продуктивність постійно зростає, але у ході використання цих процесорів необхідно ретельно досліджувати архітектурні прийоми одержання високої продуктивності і перевіряти адекватність цих прийомів проблемній області, для розв'язання задач якої створюється обчислювальна система. Подальше підвищення продуктивності мікропроцесорів зв'язується в даний час зі статичним і динамічним аналізом коду з метою виявлення резервів паралелізму рівня окремих команд і програмних сегментів з використанням інформації, наданої компілятором мови високого рівня. Дослідження в даному напрямку привели до розробки мультискалярної архітектури процесорів, що є подальшим розвитком суперскалярної архітектури.

4.5.10. Процесори з рознесеною архітектурою

Підвищення продуктивності процесорів досягається за рахунок збільшення тактової частоти, удосконалювання рівнобіжної і конвеєрної обробки даних, а також зменшення часу доступу до пам'яті. Сучасні процесори

містять десять і більше обробляючих пристроїв, кожен з яких являє собою конвеєр. Якісне завантаження конвеєрів, що функціонують паралельно, забезпечується або апаратурою процесора, або компілятором, на вхід якого надходять програми на традиційній послідовній мові програмування, або спільно апаратурою і компілятором. У компіляторах використовується витончена техніка витягнення паралелізму з послідовних програм. Апаратура процесорів орієнтована на виділення більш простих форм паралелізму, у тому числі природного. Прагнення, властиве більшості програм використовувати природний паралелізм обчислення цілочисленних адресних виразів і власне обробки даних у форматі з плаваючою крапкою, призвело до появи рознесених архітектур.

У першому наближенні процесор з рознесеною архітектурою, як показано на рис. 4.41, складається з двох зв'язаних підпроцесорів, кожний з яких управляється власним потоком команд.

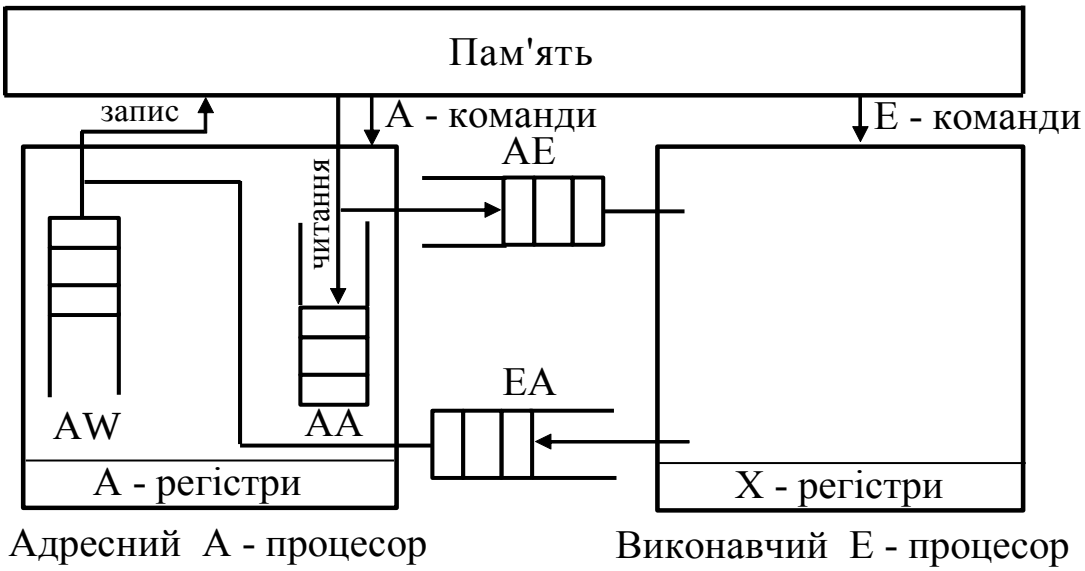


Рис. 4.41. Процесор з рознесеною архітектурою

Умовно ці підпроцесори називаються адресним А-процесором і виконавчим Е-процесором. А- і Е-процесори мають власні набори регістрів А0, А1,... і Х0, Х1,..., відповідно і набори команд. А-процесор виконує всі адресні обчислення і формує звернення до пам'яті з читання і запису. А-процесор є звичайним цілочисленним процесором, тому він здатний виконувати довільні цілочисленні перетворення, не зв'язані з обчисленням адрес. Е-процесор реалізує обчислення з плаваючою крапкою.

Дані, що витягаються з пам'яті, використовуються або в А-процесорі, будучи поміщеними в FIFO чергу АА, або містяться в FIFO черзі, яку назвали АЕ чергою, для відсилання в Е-процесор. Коли Е-процесору потрібні дані з пам'яті, він бере їх із черги АЕ. Якщо черга порожня, то Е-процесор затримується до надходження даних, які вирішують питання синхронізації

роботи А и Е-процесорів. Якщо Е-процесор виробив дані, які повинні бути відправлені в пам'ять, то він поміщає їх в FIFO чергу EA.

Під час запису даних у пам'ять після обчислення адреси А-процесор відразу відправляє адресу в FIFO чергу AW адреси запису в пам'ять, не чекаючи, поки дані надійдуть в чергу EA. А-процесор групує пари, вибираючи перші елементи черг EA і AW, і відправляє ці пари в пам'ять. Природно, якщо одна з черг або обидві порожні, то відсилання в пам'ять припиняються.

Під час читання даних А-процесор відправляє адреси в пам'ять із вказівкою черг AA або AE, в які повинні бути зчитані дані з пам'яті.

Рознесена архітектура дозволяє досягати у разі скалярної обробки продуктивності, що характерна для векторних процесорів, за рахунок передвибірки даних з пам'яті й автоматичного розгорнення декількох послідовних витків циклу в А-процесорі. Проблеми розщеплення програми на програми для А- і Е-процесорів вирішуються на рівні компілятора або спеціальним блоком-розщеплювачем.

Важливим системним аспектом рознесеної архітектури служить інтерфейс між процесором і пам'яттю за допомогою транзакцій читання і запису, що підводить логічний базис під концепцію побудови багатопроцесорних систем.

4.5.11. Мультискалярні процесори

Мультискалярні процесори використовують агресивну парадигму виконання коду з метою витягнення паралелізму рівня команд із послідовної програми, що подана мовою високого рівня. Відповідно до даної парадигми програма розбивається на сукупність задач за допомогою програмних і апаратних засобів. Задача – частина програми, виконанню якої відповідає безупинна область динамічної послідовності команд (наприклад, частина базисного блока, базисний блок, багато базисних блоків, одинична ітерація циклу, повний цикл, звертання до функції і т.д.). Задачі програми статично розмежовуються анотаціями. Залежності між операторами програми з управління подаються як граф управляючих залежностей (ГУЗ), в якому вершинами є задачі, а дугами задається порядок їх виконання. Динаміка виконання програми може розглядатися як обхід ГУЗ програми. На кожному кроці обходу мультискалярний процесор призначає одну задачу на один із процесорних елементів (ПЕ) для виконання, без обліку фактичного змісту задачі, і продовжує обхід ГУЗ від розглянутої вершини до наступної.

Задача призначається для виконання деякому процесорному елементу, передачею йому початкового значення програмного лічильника. Багато ініційованих у такий спосіб задач виконується паралельно на процесорних елементах, результатом чого є виконання багатьох команд за один процесорний цикл.

Кожний із процесорних елементів вибирає і виконує команди, що належать

виділеній йому задачі. Значення поділюваних процесорними елементами регістрів копіюються в кожен ПЕ. Результат модифікації вмісту регістрів динамічно направляється багатьом паралельним ПЕ відповідно з масками, що генеруються компілятором. Доступ до пам'яті здійснюється спекулятивно (умовно, за припущенням) без знання послідовності попередніх команд завантаження або збереження. Звертання до даних здійснюється паралельно багатьма ПЕ, обробка припиняється тільки у випадку істинної залежності даних.

Приклад архітектури мультискалярного процесора показаний на рис. 4.42.

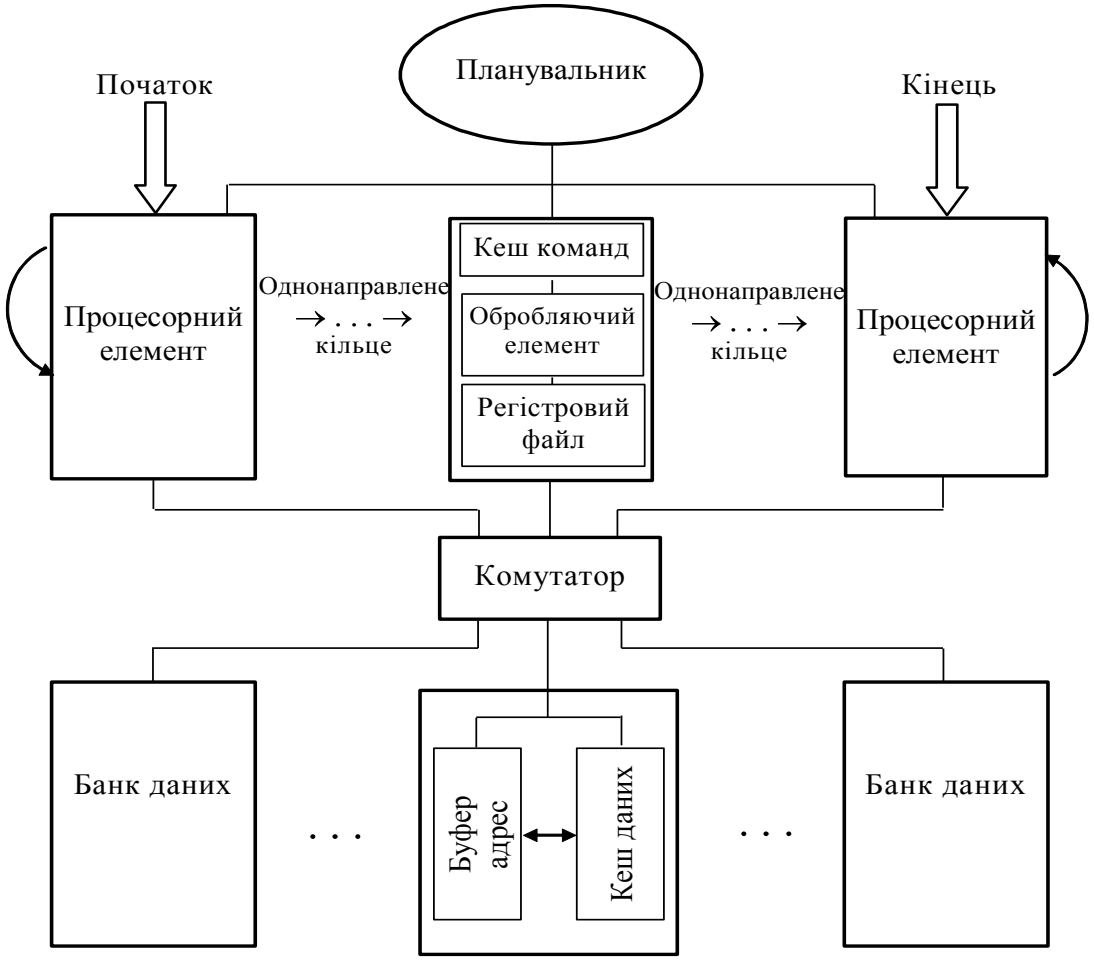


Рис. 4.42. Архітектура мультискалярного процесора

Загалом мультискалярний процесор можна розглядати як рівнобіжну обчислювальну систему, що складається із сукупності ПЕ з програмою-планувальником, що призначає задачі на ПЕ.

Як тільки задача призначена на ПЕ, він вибирає і виконує команди задачі, поки вона не завершиться. Багато ПЕ, кожний із власним внутрішнім механізмом послідовного виконання команд, підтримують виконання багатьох задач. Команди, що містяться всередині динамічного вікна виконання, обмежені першою командою в найпершій виконуваній задачі й останньою командою в останній виконуваній задачі. За умови, що кожна задача може містити цикли і

звертання до функцій, ефективний розмір вікна може бути надзвичайно великим. Істотним є те, що не всі команди всередині цього широкого діапазону одночасно розглядаються для виконання, а тільки обмежений набір усередині кожного з ПЕ.

Ключовим питанням у мультискалярній реалізації є забезпечення зв'язку за даними і керуванням між паралельними процесорами. Тобто, як забезпечити виконання послідовного обходу ГУЗ, якщо фактично виконується непослідовний обхід.

Послідовний обхід підтримується таким чином. По-перше, для кожного процесора забезпечується послідовна модель виконання призначеної йому задачі. По-друге, пропонується послідовний порядок виконання для сукупності процесорів, який підтримується за допомогою організації циклічної черги ПЕ. Покажчики початку і кінця черги ідентифікують ПЕ, які виконують найранішу і найпізнішу із призначених поточних задач відповідно.

У процесі виконання команд задачі виробляються і використовуються значення змінних програми. Ці значення зв'язані з місцем збереження, а саме з регістрами і з пам'яттю. Тому що під час послідовного виконання область збереження змінних розглядається як єдиний набір регістрів і пам'яті, мультискалярне виконання повинне підтримувати таку ж модель. Крім того, мультискалярне виконання повинне гарантувати, що значення використовуються і виробляються так само, як і у разі послідовного виконання. Щоб забезпечити це, необхідно синхронізувати обмін між задачами.

У випадку використання регістрової пам'яті логіка управління синхронізує створення значень регістрів у задачах-попередниках зі споживанням цих значень у задачах-спадкоємицях. Вироблені задачею регістрові значення можуть бути визначені статично і відзначені в масці створення задачі. В момент вироблення відповідного регістрового значення, якщо є необхідна відмітка в масці створення, це значення посилається через односпрямований кільцевий канал (див. рис. 4.42) наступним задачам, тобто в ПЕ, які є логічними спадкоємцями ПЕ, що виробив значення. Значення, які завантажуються із кільцевого каналу в регістри, що призначені для задач-спадкоємців, визначаються в масці накопичення, яка є об'єд-нанням масок створення активних у даний час задач-попередників. Як тільки значення отримані із модулів попередників, очищаються ознаки збереження в модулях-спадкоємцях. Якщо задача використовує одне із цих значень, команда, що споживає, може бути виконана тільки в тому випадку, якщо значення було отри-мане, інакше вона буде чекати отримання потрібного значення.

На відміну від значень регістрів, для значень, збережених у пам'яті, в силу динамічного обчислення адрес, не можна заздалегідь точно визначити, які з них використовуються чи виробляються задачею. Якщо відомо, що задача споживає значення з пам'яті (використовуючи команду завантаження), яке вироблене (за

допомогою команди збереження) у більш ранній задачі, можливо синхронізувати споживання і виробництво цього значення. Тобто завантаження в задачі-спадкоємиці можуть бути відкладені доти, поки в задачі-попередниці не буде виконана команда збереження. У більш загальному випадку, коли таке знання недоступне, може бути початий консервативний або агресивний підхід. Консерва-тивний підхід має на увазі необхідність чекання доти, поки не виникне впевненість, що команда завантаження прочитає правильне значення. Цей підхід звичайно має на увазі затримку виконання команд завантаження усередині задачі доти, поки не завершили операції запису в пам'ять усі задачі-попередниці, результат яких може бути використаний наступною командою. У разі агресивного підходу завантаження з пам'яті в регістри ПЕ повинні виконуватися спекулятивно, припускаючи, що задача-попередниця пізніше не буде зберігати значення в тій же самій комірці пам'яті. Щоб гарантувати, що ніяка задача-попередниця не записує значення в комірку пам'яті, попередньо читану задачею-спадкоємицею, повинна проводитися перевірка в процесі виконання обчислень. Якщо ця перевірка ідентифікує завантаження і збереження, які суперечать одне одному (не відбуваються у відповідному порядку), пізніша задача повинна бути перервана і повинна бути ініціалізована відповідна процедура відновлення. В мультискалярних процесорах використовується агресивний підхід. Через спекулятивний характер мультискалярне виконання повинне мати як засіб підтвердження правильності виконання, так і засіб виправлення у випадку неправильного виконання. Виконання команд усередині задач може розглядатися як спекулятивне з двох точок зору:

1) спекулятивне за управлінням;
2) спекулятивне за даними.

Якщо в результаті спекулятивного управлінням передбачення наступної задачі виявилося невірним, то наступна задача (задачі) повинна бути скасована і відновлена правильна послідовність задач. Аналогічно задача, що використовує неправильні дані, повинна бути скасована і повинне бути відновлене правильне значення даних. У будь-якому випадку скасування задачі приводить до скасування всіх задач, що виконуються після скасованої (інакше підтримка послідовної семантики виявляється складною).

Для спрощення збереження послідовної семантики виконання програми мультискалярний процесор видаляє задачі з циклічної черги в тому ж порядку, в якому поміщав їх у чергу. В процесі спекулятивного виконання задачі видаються значення, що можуть бути як правильними, так і неправильними. Тільки безумовно правильні результати задачі можуть бути безпечно використані іншими задачами. Проте в мультискалярному процесорі значення оптимістично посилаються для спекулятивного використання в процесі виконання інших задач. Оскільки задача посилає значення попередньо іншим

задачам, як тільки їх виробить, більша частина, якщо не всі значення, будуть послані до моменту, коли задача стає головною у черзі. Таким чином, скасування задачі для звільнення процесора і призначення нової задачі може бути виконане просто шляхом модифікування покажчика початку черги.

Переваги мультискалярної архітектури

Мультискалярний процесор має деякі властивості, що вигідно відрізняють його від традиційних суперскалярних мікропроцесорів.

У разі стандартного підходу точність передбачення розгалужень обмежує ступінь паралелізму. Якщо середня імовірність правильного передбачення переходу – 0,9, то імовірність правильного передбачення на п'ять розгалужень вперед тільки 0,6.

Мультискалярний процесор має велику глибину передбачення у разі забезпечення високої імовірності вибору правильного напрямку обчислень. Ця властивість обумовлена вибірковістю передбачення гілок. Мультискалярний процесор розбиває послідовний потік команд на задачі. Хоча задачі можуть містити внутрішні гілки, планувальник повинен передбачати тільки гілки, що відокремлюють задачі. Гілки, що містяться всередині задачі, не передбачаються (якщо вони не передбачені окремо всередині процесора).

Для суперскалярних процесорів наявність широкого вікна виконання приводить до збільшення числа відкладених команд і ускладнює контроль результатів виконання всіх команд у цьому вікні. В мультискалярній реалізації вікно може бути дуже широким, однак у будь-який момент часу тільки кілька команд повинні бути розглянуті на предмет видачі результатів (тільки одна для кожного процесора). Границі вікна відкладених команд можуть бути ідентифіковані першою й останньою командами в черзі на виконання.

Для одночасної видачі n результатів у процесорі повинна використовуватися логіка зі складністю n^2, щоб виконати перехресну перевірку залежностей серед команд. У суперскалярному процесорі це обмежує пропускну здатність логіки видачі. В мультискалярному процесорі кожен процесор видає команди незалежно, тобто використовується складність логіки порядку n. Перш ніж переупорядкувати доступ до пам'яті, необхідно ідентифікувати й обчислити всі адреси завантаження і запису значень.

У суперскалярній реалізації команди завантаження і запису упорядковуються (або зберігаються в первинній послідовності) і поміщаються в буфер разом з адресою доступу до пам'яті. Під час виконання команди завантаження перевіряється буфер, щоб гарантувати, що не відкладена ніяка більш рання команда запису по тій же самій або ще не визначеній адресі. У разі вибірки значення з пам'яті перевіряється буфер, щоб гарантувати, що не відкладена ніяка більш рання команда завантаження або запису по тій же самій або ще не визначеній адресі. В мультискалярній реалізації команди

завантаження і запису можуть бути виконані незалежно, без знання послідовності виконання команд завантаження і запису в задачах-спадкоємицях або попередницях.

В суперскалярному процесорі можлива генерація досить широкого вікна виконання з великою глибиною передбачення розгалужень. Крім того, можливо генерувати дуже гнучкий план виконання команд. Наприклад, завантаження у функції, що викликається, можуть виконуватися паралельно з запам'ятовуванням у функції, що викликає. Однак суперскалярний процесор не має уявлення про ГУЗ програми. Тому і виникає необхідність передбачення кожного переходу, що, у кінцевому рахунку, приводить до зниження точності передбачення і продуктивності.

Мультискалярний процесор багато в чому схожий на багатопроцесорну систему з загальною пам'яттю і дуже низьким рівнем непродуктивних витрат на планування. Головна їх відмінність полягає в тому, що багатопроцесорна система вимагає, щоб компілятор ділив програму на задачі, де всі співвідношення залежностей між задачами відомі, а мультискалярний процесор не вимагає ніякого апріорного знання щодо зв'язків команд з управління і даних.

Мультискалярна архітектура поєднує принципи низько- і високорівневого розпаралелювання, методи аналізу статичної і динамічної структур програми, завдяки чому дозволяє домогтися більш високих значень ефективності використання обчислювальних ресурсів процесора, ніж інші типи архітектур. Фактично в мультискалярних процесорах реалізований симбіоз автоматично розпаралелюючого компілятора, що дає вказівки апаратурі процесора у вигляді оцінок команд і спеціальних команд, і апаратних засобів, що сприймають ці вказівки.

Звичайно, викладений вище підхід не є єдино можливим у разі реалізації цієї плідної ідеї – залучення компілятора до розподілу завдань по процесорах і балансуванню завантаження процесорів у багатопроцесорних системах.

4.6. ІСТОРИЧНА ДОВІДКА ПРО МІКРОПРОЦЕСОРИ ВІДОМИХ КОМПАНІЙ−ВИРОБНИКІВ

4.6.1. Процесори компанії Intel

У цьому підрозділі проведемо історичний огляд мікропроцесорів компанії Intel і розглянемо їх порівняльні характеристики. Компанія Intel Corporation є безумовною рушійною силою в області персональних комп'ютерів. Кожне нове покоління її мікропроцесорів не просто має підвищену продуктивність, але і є технологічним стрибком в підвищенні ефективності виконання, тактової робочої частоти, пропускної здатності і конструктивних удосконалень (таких як динамічне виконання команд і SIMD-технологія). В табл. 4.3 приводяться

параметри мікропроцесорів компанії Intel, починаючи з першого по сьоме покоління.

Коротко розглянемо основні характеристики мікропроцесорів. В дужках поряд з назвою процесора приведений рік початку його виробництва.

Мікропроцесор 8086/8088 (1978/1979) містить у собі 29000 транзисторів і є першим 16-розрядним мікропроцесором, тобто розрядність його зовнішньої шини даних складає 16 біт. Це відразу ж дозволило збільшити пропускну здатність у два рази в порівнянні з 8-розрядними процесорами. Довжина кожного з 24-х регістрів процесорів 8086 /8088 була збільшеною до 16 розрядів замість колишніх 8 розрядів, а 20 ліній адреси забезпечують пряму адресацію 1048576 байт (1 Мбайт) зовнішньої системної пам'яті. Процесори 8086 і 8088 (як і всі подальші процесори компанії Intel) могли адресувати 64 Кбайт простору вводу/виводу.

Таблиця 4.3

Основні технічні и технологічні характеристики базовых процесорів компанії Intel з першого по сьоме покоління

Покоління	Сімейство процесорів	Рік випуску	Розрядність, *біт*	Тактова частота процесора,	Частота системної шини, *МГц*	Кеш-пам'ять L1, *Кбайт*	Кэш-пам'ять L2, *Кбайт*	Норма тех-процесу, *мкм*	Кількість транзисторів, *млн*
1	Intel 8086 Intel 8088	1978 1979	16 16	4,77 - 8 4,77 - 8	4,77-8 4,77-8	– –	– –	3 3	0,03 0,03
2	Intel 80286	1982	16	8 - 20	8 - 20	–	–	1,5	0,14
3	Intel 80386	1985	32	16 - 33	16 - 33	–	–	1	0,38
4	Intel 80486	1989	32	25 - 100	25 - 33	8	–	1	1,2
5	Intel Pentium Intel Pentium MMX	1993 1997	32 32	60 - 200 166-233	60 - 66 66	8+8 16+16	– –	0,5 0,35	3,3 5,0
6	Intel Pentium Pro Pentium II Pentium III	1995 1997 1999	32 32 32	150-200 233-600 450 – 1400	66 66-100 100 – 133	8+8 16+16 16+16	256 512 256/ 512	0,35 0,25 0,18 0,13	5,5 7,5 10 – 40
7	Intel Pentium 4	2000	32	1300 – 3400	400 –800	8+12	256/ 512	0,18 0,13	42 – 55

Шістнадцатирозрядний мікропроцесор 80186 (1980) з серії x86 був оснащений внутрішнім тактовим генератором, системним контролером, контролером переривань, контролером прямого доступу до пам'яті і програмованим таймером. Всі ці схеми були інтегровані в кристал самого мікропроцесора. Ніякий процесор компанії Intel ні до, ні після цього мікропроцесора не мав такого ступеня інтеграції. В цьому процесорі також вперше відмовилися від використання тактової частоти в 5 МГц, використовувалася частота в 8, 10 і 12,5 МГц. У всьому останньому процесор x186 був схожий на моделі 8086 і 8088, включаючи наявність 24-х регістрів і 20-ти адресних ліній доступу до 1 Мбайта оперативної пам'яті.

Мікропроцесор 80286 (1982) (вперше встановлювався в персональні комп'ютери IBM PC/AT і сумісні з ним) містить в собі 134000 транзисторів, має 24 регістри і володіє деякими іншими істотними перевагами в порівнянні з попередніми моделями. Цей процесор міг виконувати 1,2 MIPS, 1,5 MIPS або 2,66 MIPS під час роботи на тактовій частоті 8, 10 і 12,5 МГц відповідно. У цьому процесорі також був подоланий бар'єр в 1 Мбайт пам'яті, яка адресувалася, за рахунок використання 24 замість 20 адресних ліній, що дозволяє адресувати до 16 Мбайт оперативної пам'яті. Окрім цього, мікропроцесор 286 адресує до 1 Гбайта *віртуальної пам'яті*, що дозволяє організувати обмін блоками команд або даних між реальною оперативною пам'яттю процесора 286 і віртуальною пам'яттю, розміщеною, наприклад, на жорсткому диску. Мікропроцесор 286 може працювати спільно з математичним співпроцесором 80287.

Мікропроцесор 80386 (1985). Наступним мікропроцесором, випущеним компанією Intel в 1985 році, був центральний мікропроцесор 80386DX, що містив у собі 275000 транзисторів і мав 32 регістри. За рахунок використання повної 32-розрядної шини даних пропускна здатність цього мікропроцесора була подвоєна в порівнянні з мікропроцесором 80286. Випускалися версії мікропроцесора, які могли працювати на тактовій частоті в 16, 20, 25 і 33 МГц. Процесор, що працював на частоті 33 МГц, мав пропускну здатність до 50 Мбайт у секунду і міг виконувати до 11,4 MIPS. Повна 32-розрядна шина адреси дозволяла адресувати до 4 Гбайт оперативної пам'яті в доповненні до 64 Тбайт віртуальної пам'яті. У цьому процесорі для підвищення продуктивності вперше був використаний командний конвеєр, який, як було описано вище, дозволяє центральному процесору починати обробку наступної команди, не чекаючи кінця виконання попередньої команди.

Мікропроцесор 80486 (1989). Постійна гонитва за швидкістю і продуктивністю привела в 1989 році до появи 32-розрядного мікропроцесора 80486DX, який містив у собі 1,2 млн. транзисторів і мав 29 регістрів. Процесор 486DX містив 32 адресних лінії для адресації до 4 Гбайт фізичної оперативної

пам'яті і до 64 Тбайт віртуальної пам'яті. Продуктивність процесора 486DX в два рази перевершувала продуктивність процесора 386DX на частоті 33 МГц, і він виконував 26,9 мільйонів операцій у секунду. Спочатку випускалися дві версії цієї моделі – для роботи на тактовій частоті 25 МГц і 30 МГц.

У процесорах сімейства 486, також як і в 386-й серії, використовувався конвеєр для підвищення продуктивності обробки команд, і була додана кеш-пам'ять об'ємом 8 Кбайт. Використання кеш-пам'яті приводить до зменшення числа звернень до основної пам'яті (на цьому економиться час) за рахунок прогнозу наступної команди, яка буде потрібна центральному процесору, і її завантаження в кеш-пам'ять до того, як вона знадобитися ЦП. Якщо потрібна команда опиниться в кеш-пам'яті, то ЦП витягуватиме її звідти, не витрачаючи часу на доступ в основну пам'ять (повільнішу в порівнянні з кеш-пам'яттю). Іншим удосконаленням процесора 486DX є включення в його склад виконавчого блока, що виконує операції над числами з плаваючою комою, замість вико-ристання автономного математичного співпроцесора. Але це відноситься не до всіх процесорів 486-ї серії. Третьою відмінністю є випуск двох версій мікропроцесора 486DX – з напругою живлення в 5 і 3В. Версія з живлячою напругою 3В призначалася для використання в мобільних комп'ютерах.

Нарешті, мікропроцесор 486DX можна було замінити на продуктивніший без заміни системної плати. До 1989/1990 року термін служби комп'ютера визначався терміном служби його центрального процесора – якщо процесор застарівав, те ж саме відбувалося і з комп'ютером (точніше, з системною платою). Це вимушувало користувача купувати новий комп'ютер (або встановлювати нову системну плату) кожні декілька років для того, щоб використовувати сучасні комп'ютерні технології. Архітектура процесора 486 орієнтована на підтримку модернізації системи шляхом його заміни на інший з вищою внутрішньою тактовою частотою, який можна встановити в той же комп'ютер. Компанія Intel назвала таку технологію OverDrive.

Першими мікропроцесорами серії OverDrive стали процесори 80486DX2/50 і 80486DX2/66, випущені в 1992 році. Цифра «2» разом з буквами «DX» указує на те, що мікропроцесор працює на подвоєній частоті системної шини. Мікропроцесор 486DX2/50 працював у системі з частотою 25 МГц, але центральний процесор виконував 40,5 млн. операцій у секунду. Використання швидших процесорів на повільних системних платах дозволяло встановлювати ці мікропроцесори на вже існуючі системні плати. Обидва OverDrive-процесори мали вбудовані математичні співпроцесори і самі могли бути замінені ще швидшими версіями. Мікропроцесор 486DX2/50 випускався в двох версіях – з живленням 5 і 3В, а мікропроцесор 486DX2/66 випускався тільки у варіанті з напругою живлення 5В.

У 1992 році компанія Intel випустила високоінтегрований мікропроцесор сімейства 486 (споживаючий малу потужність живлення) під назвою 80486SL. Він мав 32-розрядні шини адреси і даних, 8 Кбайт вбудованої кеш-пам'яті, і вбудований математичний співпроцесор, що робило його практично ідентичним іншим мікропроцесорам 486 сімейств за винятком того, що він містив 1,4 млн. транзисторів. Вбудовані в нього додаткові схеми забезпечували функції управління живленням, що дозволяло мікропроцесори серії SL використовувати в переносних комп'ютерах. Ці мікропроцесори випускалися в різних версіях для роботи на тактовій частоті 25 і 33 МГц, і з живлячою напругою 3 і 5В. Мікропроцессор 486SL на тактовій частоті 33 МГц виконував 26,9 млн. операцій у секунду.

Останніми трьома моделями цього сімейства, випущеними в 1993 році, були процесори 80486DX2/40, 80486SX/SL і 80486DX/SL. Мікропроцесор 80486DX2/40 був третім OverDrive-процесором, призначеним для використання в комп'ютерах, що працюють на частоті 20 МГц, хоча сам мікропроцесор мав внутрішню робочу тактову частоту в 40 МГц і міг виконувати 21,1 млн. операцій у секунду. Мікропроцесори 486SX/SL (6,9 млн. операцій в секунду на частоті 33 МГц) і 486DX/SL (6,9 млн. операцій в секунду на частоті 33 МГц) ідентичні своїм оригінальним SX і DX версіям, а індекс SL означає, що вони мають вбудовані функції управління живленням, що важливе у разі використання в переносних комп'ютерах.

У 1994 році була випущена остання модель мікропроцесора сімейства 486 під назвою OverDrive DX4. Незважаючи на позначення DX4, ці OverDrive-процесори (напруга живлення 3,3В) працювали на потрійній тактовій частоті – наприклад, мікропроцесор 486DX4/100 міг працювати на системній платі з частотою 33 Мгц. Всі мікропроцесори сімейства 80486 володіли зворотною сумісністю з попередніми сімействами мікропроцесорів аж до процесорів 8086/8088.

Pentium (1993). До 1992 року центральні мікропроцесори 486-ї серії надійно влаштувалися в настільних комп'ютерах загального призначення, а компанія Intel вже провела основну роботу із створення центрального процесора наступного покоління. В 1993 році компанією Intel був представлений мікропроцесор Pentium, що містить 3,21 млн. транзисторів (інша назва серії – «P5» або «P54»). Процесор Pentium зберіг 32-розрядну адресну шину 486-го сімейства. З такою адресною шиною мікропроцесор Pentium міг безпосередньо адресувати 4 Гбайта оперативної пам'яті і забезпечити доступ до 64 Тбайт віртуальної пам'яті. 64-розрядна зовнішня шина даних подвоїла пропускну здатність у порівнянні з процесорами 486-ї серії. На частоті 60 МГц процесор Pentium виконує 10 млн. операцій у секунду, а на частоті 66 МГц – 11,6 млн. операцій у секунду (в два рази більше, ніж мікропроцесор 486DX2/66). Всі версії мікропроцесора Pentium включають у свій склад

математичний співпроцесор і можуть у майбутньому бути замінені мікропроцесорами OverDrive з більшою робочою тактовою частотою.

Початковий процесор Pentium використовує дві кеш-пам'яті по 8 Кбайт – одну для команд, іншу – для даних (всього 16 Кбайт). Використання подвійного конвеєра дає можливість дійсної обробки декількох команд за один такт. Іншим важливим удосконаленням в конструкції процесора Pentium є включення функцій управління живленням (по аналогії з процесорами серії 486SL), що дозволяє їх ефективно використовувати в переносних комп'ютерах.

Перші моделі процесорів Pentium живились напругою 5В, але, починаючи з моделей, що використовують тактову частоту 100 МГц (P54C), почала застосовуватися напруга 3,3В і нижче. Процесори Pentium володіють повною зворотною сумісністю зі всім програмним забезпеченням, написаним для процесорів 8086/8088 і пізніших моделей. Компанія Intel випустила багато моделей процесорів Pentium з тактовою робочою частотою до 200 МГц. Швидші моделі не випускалися, щоб не конкурувати з процесорами наступних поколінь: Pentium MMX, Pentium Pro, Pentium II/III.

Pentium Pro (1995). Незважаючи на те, що процесор Pentium добре працював з 16-ти і 32-розрядними операційними системами, розробники продовжували пошуки шляхів оптимізації продуктивності 32-розрядних операцій, особливо для операційної системи Windows NT і Windows 95, що з'явилася. Процесор Pentium Pro (інше позначення «P6» або «PPro») був удосконаленням процесора Pentium і призначався для використання в бізнес-додатках, таких як високопродуктивні робочі станції і мережні сервери. Різні моделі процесорів P6 працювали на тактовій частоті від 150 до 200 МГц і могли використовуватися в багатопроцесорних системах (до 4-х процесорів).

У процесорі Pentium Pro застосовується механізм динамічного виконання команд з метою підвищення продуктивності, а також використовуються дві окремі кеш-пам'яті (по 8 Кбайт) першого рівня: одна для даних, інша – для команд. Іншою важливою особливістю процесора Pentium Pro є використання вбудованої кеш-пам'яті другого рівня ємністю до 1 Мбайт. Це максимально збільшує продуктивність процесора P6 навіть у відсутності кеш-пам'яті другого рівня на системній платі.

Pentium MMX (1997). В 1997 році компанія Intel внесла значне удосконалення до процесорів Pentium під назвою мультимедійні розширення (MMX – multimedia extensions). Шляхом розвитку і поліпшення існуючої архітектури процесорів Pentium і додавання 57 нових команд, компанія Intel розробила і почала випускати в кінці 1990-х років мікропроцесор середнього рівня Pentium MMX. Працюючи на тактових частотах у межах від 133 до 233 МГц, процесор Pentium MMX показував на 10-20% більшу продуктивність у порівнянні зі «звичайними» процесорами Pentium, що працювали на тій же тактовій частоті, під час виконання одних і тих же програм. У ході виконання

програм, написаних з використанням команд MMX, персональний комп'ютер з процесором Pentium MMX міг забезпечити більшу глибину передачі кольорів і більшу роздільність у разі збереження високої швидкості зміни кадрів під час відтворення і створення відео.

У процесорі Pentium MMX був удвічі збільшений об'єм кеш-пам'яті (до 16 Кбайт) як для команд, так і для даних.

У процесори серії Pentium MMX увійшли і багато інших удосконалень. Суперскалярна архітектура процесора здатна виконувати за один машинний такт дві команди з цілочисельними аргументами, що збільшує продуктивність виконання обчислень над цілими числами. Конвеєрний пристрій виконання операцій з плаваючою комою, що підтримує операції з 32-х, 64-х і 80-розрядними даними, здатний виконувати дві команди за кожен машинний такт. Для збільшення продуктивності обробки команд був доданий командний конвеєр. Для прискорення операцій з пам'яттю обидва конвеєри використовують чотири буфери запису.

Pentium II (1997). Після того, як процесори Pentium MMX і Pentium Pro впевнено затвердилися на ринку персональних комп'ютерів, компанія Intel вирішила об'єднати кращі властивості цих процесорів – ефективність виконання програм процесора Pentium Pro з мультимедійними розширеннями процесора Pentium MMX. У результаті в 1997 році був випущений процесор Pentium II (або «PII», попередня назва – «Klamath»). Процесор Pentium II на частоті 266 МГц показував в 1,6 – 2 рази більшу продуктивність у порівнянні з процесором Pentium з тактовою частотою 200 МГц.

У процесорі Pentium II застосовувалося також динамічне виконання команд, вперше реалізоване в процесорі Pentium Pro. Алгоритм динамічного виконання команд використовує множинний прогноз галуження виконуваної програми для прогнозу ходу виконання програми після декількох команд умовних переходів, що приводить до прискорення потоку обробки команд і даних у процесорі. Алгоритм аналізу потоку даних потім створює оптимальну (переупорядковану) послідовність команд, досліджуючи взаємозв'язки команд. І, нарешті, проводиться «спекулятивне» виконання команд (у припущенні того, що здійсниться саме передбачений порядок виконуваних команд програми) на основі оптимізованої послідовності команд. Динамічне виконання постійно завантажує роботою суперскалярні операційні блоки процесора і збільшує загальну продуктивність.

У процесорі Pentium II використовується 32 Кбайт кеш-пам'яті: 16 Кбайт для даних і 16 Кбайт для команд. У корпусі процесора також знаходиться 512 Кбайт кеш-пам'яті другого рівня, що дозволяє отримати максимальну продуктивність процесора, не покладаючись на кеш-пам'ять системної плати. Процесор PII підтримує до 64 Гбайт фізичної оперативної пам'яті і може працювати в двопроцесорних системах – за наявності відповідної

системної плати, що підтримує багатопроцесорну технологію SMP. Пристрій конвеєрного виконання операцій з плаваючою комою підтримує 32-х, 64-х і 80-розрядні операції, обробляючи дві команди за один такт, і виконує понад 300 млн. операцій з плаваючою комою в секунду на частоті 300 МГц.

Головна відмінність від попередніх моделей мікропроцесорів полягає у використовуваному типі корпуса. Компанія Intel відмовилася від використання корпусів типу Socket 7 (Pentium) і Socket 8 (Pentium Pro) і прийняла «картріджний» тип корпусу, відомий як корпус з однорядним розташуванням виводів (SEC – Single Edge Contact). Тепер він відомий під назвою «Slot 1».

Pentium ll/lll Celeron (1998). Загальновідомі сьогодні мікропроцесори Celeron компанія Intel випустила в квітні 1998 року. Спочатку цей процесор був спрощеною версією процесора Pentium II. У ньому використовувалося ядро P6 і були всі функції процесорів Pentium Pro і Pentium II. Процесор Celeron мав кеш-пам'ять першого рівня ємністю 32 Кбайт (16 Кбайт для команд і 16 Кбайт для даних). Він включав MMX-функції, блок конвеєрного виконання операцій з плаваючою комою і архітектуру динамічної обробки команд. Процесор виготовлявся за тією ж 0,25-мкм технологією, що і Pentium II. Сучасні моделі процесорів Celeron мають у своїй основі ядро Coppermine процесора Pentium III і виготовляються за 0,18-мкм технологією.

Відсутність вбудованої кеш-пам'яті другого рівня серйозно обмежувала продуктивність перших моделей процесорів Celeron. Дешевші процесори конкурентів, які включали кеш-пам'ять другого рівня, перевершували за продуктивністю процесори Celeron з однаковою тактовою частотою. Починаючи з моделі Celeron 300A, компанія Intel додала в процесори Celeron вбудований кеш розміром 128 Кбайт. Для процесорів Celeron в корпусах PPGA і FC-PGA кеш-пам'ять другого рівня розташовувалася в кристалі самого процесора. Така інтеграція дозволила не тільки наздогнати за швидкістю інші аналогічні процесори, але і поліпшити його продуктивність ще більше. Фактично, кеш-пам'ять другого рівня ємністю 128 Кбайт, інтегрована в процесор Celeron і працюючи на частоті процесора, робить цей процесор порівнянним за продуктивністю з процесором Pentium II, у якого кеш-пам'ять другого рівня 512 Кбайт розташована поза кристалом і працює на половині частоти процесора.

Pentium III (1999). Процесор Intel Pentium III вперше з'явився в 1999 році, і в ньому було використано те ж саме базове ядро P6, що і в процесорах Pentium Pro і Pentium II (тому всі основні функції цієї лінії процесорів збереглися без змін). Пізніше процесори Pentium III почали виготовляти за 0,18-мкм технологією, що дозволило понизити робочу температуру кристала процесора (в порівнянні з технологією 0,25-мкм).

Загальна продуктивність процесора Pentium III підвищилася за рахунок збільшення робочої тактової частоти і використання шини FSB на частоті

133 МГц. У систему команд процесора були також додані нові інструкції, об'єднані в набір SSE (Streaming SIMD Extensions – Потокові розширення SIMD). Для їх підтримки були також додані нові регістри, що привело до збільшення числа транзисторів на кристалі до 9,5 млн. Як і у разі MMX, для використання розширень SSE необхідне відповідне програмне забезпечення. Команди множини SSE дозволяють збільшити продуктивність процесора під час роботи з тривимірною графікою. Процесор також має вбудовану кеш-пам'ять першого рівня ємністю 32 Кбайт, може адресувати 4 Гбайт ОЗП з використанням коду корекції помилок пам'яті (ECC), і може працювати в двопроцесорних системах. Перші процесори Pentium III (виготовлені за 0,25-мкм технологією) використовували ядро Katmai. Вони могли працювати на частотах від 450 до 600 МГц і мали 512 Кбайт кеш-пам'яті другого рівня. Кеш-пам'ять розташовувалася на окремому кристалі і працювала на половинній частоті ядра процесора. Пізні моделі процесорів Pentium III виготовлялися за 0,18-мкм технологією. Ядро цих процесорів отримало назву Coppermine і працювало на тактовій частоті 500 МГц і вище. Покращувана технологія виробництва дозволила використовувати меншу за об'ємом кеш-пам'ять другого рівня (256 Кбайт), розташовану на одному з процесором кристалі. Вона почала працювати на частоті ядра процесора, тому продуктивність процесора не знизилася.

Pentium ll/lll Xeon (1999). Процесор Xeon є високопродуктивною моделлю сімейства мікропроцесорів Pentium II/III. Він призначений для використання в робочих станціях і серверах. До характерних рис цього процесора можна віднести можливість його використання в багатопроцесорних системах (до восьми процесорів), а також те, що кеш-пам'ять другого рівня працює на частоті ядра і має збільшений розмір 512 Кбайт, 1 Мбайт або 2 Мбайт). Великий розмір кеш-пам'яті виключає можливість її розміщення на одному кристалі з процесором – вона знаходиться в окремому корпусі, розташованому поряд з ядром процесора. Компанія Intel вирішила проблему швидкості кеш-пам'яті, пов'язану з роздільним її розміщенням – в процесорі Xeon вона працює на частоті ядра процесора. Через роздільне розміщення кеш-пам'яті були збільшені розміри корпусу процесора Pentium Xeon, що привело до неможливості використання системних плат з процесорним роз'ємом Slot 1. З цієї причини для процесора Xeon був розроблений роз'єм Slot 2, що має збільшений розмір.

Pentium 4 (2000). Після того, як процесори сімейства Pentium II/III подолали бар'єр тактової частоти в 1 ГГц, внутрішні обмеження традиційної архітектури ядра P6 стали перешкодою на шляху подальшого нарощування продуктивності процесорів. Тому компанія Intel розробила новий тип мікропроцесора Pentium 4 з архітектурою «NetBurst». Ця архітектура має ряд нових і передових функціональних можливостей, включаючи гіперконвеєрну

технологію, 400 (в старших моделях 533) МГц системну шину, кеш-пам'ять для зберігання адрес виконаних програмних переходів виконуваної програми (Execution Trace Cache), і блок прискореної обробки (Rapid Execution Engine). До інших нововведень від-носяться кеш-пам'ять для зберігання адрес майбутніх програмних переходів (Advanced Transfer Cache), покращуване динамічне виконання команд, вдосконалений блок обробки чисел з плаваючою комою та мультимедійних даних і нові команди, що отримали назву SSE2 (Streamed SIMD Extensions – потокові розширення SIMD). У даний час процесори Pentium 4 працюють на тактовій частоті від 1,3 до 3 ГГц сумісно з комплектами мікросхем Intel 845, 850 та інше і призначені для використання в комп'ютерах, на яких виконується широкий круг мультимедійних і комунікаційних завдань. Сюди відносяться обробка потокових звукових і відеофайлів, що приймаються з Інтернет, обробка зображень, відеомонтаж, розпізнавання мови, тривимірна графіка, завдання автоматизованого проектування, ігри і робота в багатозадачному середовищі.

Процесор Pentium 4 містить 42 млн. транзисторів і випускається в корпусах з роз'ємом Socket 478 (у перших моделях використовувався роз'єм Socket 423). Зміна типу процесорного роз'єму обмежує можливість модернізації систем на базі ранніх процесорів Pentium 4. Сучасні процесори Pentium 4 випускаються на ядрах Prescott і Gallatin (перед цим використовувалися процесорні ядра Willamete і Northwood).

Pentium 4 Celeron (з 2000). Як і Pentium II/III Celeron, нове покоління бюджетних процесорів Intel поєднує помірну ціну з цілком гідною продуктивністю. Процесори випускаються по технологічному процесу 130 нм, а їх частоти досягають 2,8 ГГц. При цьому частота їх системної шини обмежена порогом 400 МГц, а розмір кеш-пам'яті другого рівня – скромним значенням 128 Кбайт.

Pentium 4 HT (2002) є оновленою версією процесора Pentium 4 з підтримкою технології Hyper-Threading. Одночасне виконання двох потоків дає для процесорів з технологією Hyper - Threading приріст продуктивності в 25-40% в порівнянні із звичайними процесорами Pentium 4. У системні чипсети для даного процесора введена підтримка пам'яті DDR SDRAM. Тактова частота системної шини підвищена і складає 800 МГц. Відбулася повна відмова від 180 нм технологічної норми на користь прогресивних 80 і 130 нм. Тактові частоти самого процесора починаються з 2,4 ГГц і досягають 3,06 ГГц. Об'єм кеш-пам'яті другого рівня досягає 1 Мбайт.

Pentium 4 Extreme Edition (2003) У цьому процесорі окрім кеш-пам'яті другого рівня об'ємом 512 Кбайт встановлена кеш-пам'ять третього рівня об'ємом 2 Мбайт а також істотно збільшена кількість транзисторів і як наслідок - продуктивність. Тактові частоти екстремальної редакції складають 3,2 і 3,4 ГГц, що не є максимальними значеннями, та і використовувана технологічна норма – 130 нм – не є рекордною.

Pentium 4 Xeon (з 2001). Зараз має офіціальну назву **Xeon**, а не Pentium 4 Xeon. Процесор Xeon призначений для серверів і робочих станцій і використовує системну шину з частотою 533 МГц. Від попереднього покоління він відрізняється новаторськими технологіями, включаючи мікроархітектуру NetBurst і технологію Hyper-Threading. Розмір кеш-пам'яті другого рівня досягає 2 Мбайт за частоти процесора 3,2 ГГц. Пропускна здатність підсистеми вводу/виводу в системах на базі Xeon досягає 4,3 Гбайт/с. Як оперативна пам'ять використовується DDR SDRAM (під час розробки чипсетів під ці процесори Intel відмовилася від підтримки Rabmus DRAM). Підтримуються конфігурації з двома процесорами, що (у поєднанні з технологією Hyper-Threading) дозволяє використовувати програмне забезпечення, розраховане на чотири процесори.

Розвитком цієї лінійки є процесори Xeon MP, які розраховані на дійсно багатопроцесорні конфігурації, – з 4, 8 і навіть більшої кількості процесорів. Дещо менша частота системної шини (480 МГц) компенсується кеш-пам'яттю другого рівня об'ємом 256 Кбайт (використовується архітектура Advanced Transfer Cache) і кеш-пам'яттю третього рівня об'ємом до 4 Мбайт. Підсистема вводу/виводу на базі шини PCI-X забезпечує пропускну здатність до 4,8 Гбайт/с. Системи на базі Xeon MP підтримують шину управління SMBus і гарячу заміну компонентів (у тому числі і процесорів).

Itanium (2001). Процесор Itanium, що випускається з середини 2001 року, є першим 64-розрядним процесором компанії Intel. Архітектура IA-64 компанії Intel поєднує в собі ряд нововведень, що підвищують продуктивність процесорів традиційних типів. Архітектура процесора Itanium заснована на наступному поколінні технологій, таких як чіткий паралелізм (EPIC, Explicit Parallelism), прогноз галужень (Prediction) і гадане попереджуюче виконання програми (Speculation), що забезпечують високу ефективність обробки і що збільшують число виконуваних команд за один цикл. Ця додаткова обробляювальна потужність призначена для задоволення вимог роботи в Інтернеті, у високопродуктивних серверах і робочих станціях. Крім того, архітектура IA-64 забезпечує можливість подальшого зростання продуктивності.

У даний час максимальна частота процесорів Itanium складає 800 МГц за частоти системної шини 266 МГц. Кеш-пам'ять першого рівня має об'єм 32 Кбайт, другого – 96 Кбайт, третього – аж до 4 Мбайт. Підтримуються багатопроцесорні конфігурації аж до 512 процесорів. Як оперативна пам'ять використовується дещо застаріла специфікація PC 100, а як шина розширення – PCI (версія з частотою 66 МГц).

Itanium 2 (2002). Розвитком лінійки 64-розрядних процесорів Intel є Itanium 2. В порівнянні з першим поколінням, у ньому збільшена частота системної шини (до 400 МГц), при цьому її розрядність складає 128 біт, що забезпечує пропускну здатність до 6,4 Гбіт/с. На зміну PCI прийшла PCI-X з

частотою 133 МГц, а підтримувана оперативна пам'ять тепер допускає використання технології DDR. Кеш другого рівня виріс до 256 Кбайт, третього – до 6 Мбайт. Робочі частоти процесорів лежать у діапазоні від 1 до 1,6 ГГц. Серед інших особливостей – вдосконалена архітектура автоматичної перевірки Machine Check Architecture (MCA) з розширеним використанням коду корекції помилок ECC.

Одночасне виконання двох потоків для процесорів з технологією Hyper-Threading надихнуло програмістів зайнятися розробкою багатопотокових програм, і підготувало ґрунт для появи в недалекому майбутньому багатоядерних процесорів. Intel представила свій власний набір 64-розрядних розширень, який назвала EM64T або IA-32e.

У 2006 році Intel змінила назву бренду Pentium на Core і випустила двоядерну мікросхему Core 2 Duo. На відміну від процесорів архітектури NetBurst (Pentium 4 і Pentium D), в архітектурі Core 2 ставка робилася не на підвищення тактової частоти, а на поліпшення інших параметрів процесорів, таких як кеш, ефективність і кількість ядер. Потужність цих процесорів, що розсіюється, була значно нижче, ніж у настільної лінійки Pentium. З параметром TDP, рівним 65 Вт, процесор Core 2 мав найменшу рассеиваемую потужність з настільних мікропроцесорів.

Характеристика багатоядерних процесорів приводиться в розділі 4.8.

4.6.2. Процесори компанії AMD

Компанія Advanced Micro Device (AMD), колишній союзник компанії Intel, стала її єдиним, сильним конкурентом на ринку мікропроцесорів. AMD відома як виробник добре сконструйованих і володіючих великою сумісністю «альтернативних» процесорів для персональних комп'ютерів. Вона почала займатися розробкою і виробництвом мікропроцесорів, починаючи з 386 серії (одним з яких був процесор AMD Am386). Хоч AMD раніше завжди відставала від компанії Intel у справі виробництва нових мікропроцесорів, зараз цей розрив зник. Після випуску новітніх моделей процесорів (таких як Athlon і Duron) компанія AMD навіть вирвалася трохи вперед за продуктивністю мікропроцесорів і швидкістю їх роботи (за оцінками деяких випробувальних програм). У табл. 4.4 приведені параметри процесорів AMD, починаючи з сімейства K5.

Серія процесорів Am486DX (1994). Процесори серії Am486 компанії AMD були відповіддю на процесори Intel 486 з подвоєною тактовою частотою і процесори OverDrive з потрійною тактовою частотою, які з'явилися на початку 1990 року. Компанія AMD вбудувала в процесори кеш-пам'ять із зворотним записом і покращені функції управління живленням, включаючи живлення від 3В, режим управління системою (SMM) і управління тактовою частотою (ці

режими призначені для використання в настільних і переносних комп'ютерах з енергозбережною технологією Energy Star).

Таблиця 4.4

Основні технічні та технологічні характеристики базових процесорів компанії AMD, починаючи з сімейства K5

Покоління	Особливість процесора	Рік випуску	Розрядність, *біт*	Тактова частота процесора, *МГц*	Частота системної шини, *МГц*	Кеш-пам'ять L1, *Кбайт*	Кэш-пам'ять L2, *Кбайт*	Норма тех-процесу, *мкм*	Кількість тран-зисторів, *млн*
K5	Без підтримки MMX	1996	32	75 - 116	50 - 66	16 + 8	–	0,35	4,3
K6	З підтримкою MMX	1997	32	166 – 300	66	32+32	–	0,3 - 0,25	8,8 – 9,3
K6	Ядро Chompers	1997	32	266 – 550	66 - 100	32+32	–	0,25	9,3
K6 - 2+		1998	32	450 – 550	100	32+32	128	0,18	–
K6 - 3/P	Ядро Sharptooth	1998	32	450	100	32+32	256	0,25	21,3
K7	Athlon	1999	32	500 – 1000	200 – 266	128	512	0,25-0,18	22
K7	Duron	2000	32	600 – 1800	200 – 266	128	64	0,18-0,13	22
K8	Athlon XP	2001	64	1800 – 2400	400 – 800	64+64	512-1024	0,13	37,2
K8	Athlon 64	2002	64	1333 – 2400	400 – 800	64+64	512-1024	0,13	37,2

Am5x86 (1995). Саме цей процесор вивів компанію AMD на ринок центральних мікропроцесорів. Після появи процесорів лінії Intel Pentium перед користувачами встала проблема заміни системної плати в своїх комп'ютерах з тим, щоб отримати «дійсний» Pentium-процесор або використовувати дорогий OverDrive-процесор для своєї 486-системної плати. Компанія AMD запропонувала процесор Am5x86 (або просто «5x86») як альтернативу Pentium OverDrive-процесорам. Процесор Am5x86 мав порівнянну з Pentium продуктивність, працюючи на 4-кратній тактовій частоті системної шини (133 МГц), використовуючи частоту 33 МГц шини 486-системної плати. Частота в 33 МГц також ідеально підходила для 33 МГц шини PCI, що

з'явилася у той час. Додаткові функції, такі як єдина 16 Кбайт кеш-пам'ять із зворотним записом, ще більш збільшували продуктивність процесора Am5x86. На практиці процесор Am5x86 показував більшу продуктивність, чим Pentium 75 МГц при набагато меншій вартості. Процесор Am5x86 часто використовувався для модернізації 486 комп'ютерів, оскільки така заміна дозволяла використовувати програмне забезпечення, створене для процесорів Pentium-класу без великих переробок апаратури.

Серія К5 (1996). Хоча процесор Am5x86 був дуже популярним, він не був «дійсною» альтернативою процесору Pentium. Таке положення справ продовжувалося до 1996 року, коли компанія AMD випустила процесор серії К5 для ринку персональних комп'ютерів, який був справжньою альтернативою процесору Pentium. Процесор К5 повністю сумісний з процесорним роз'ємом системних плат Socket 7 (Pentium). Для процесора К5 потрібна була нова системна BIOS для правильної ідентифікації і підтримки його роботи комплектом мікросхем. Процесор К5 повністю сумісний зі всіма x86-операційними системами і програмним забезпеченням.

Серія К6 (1997). Процесор К6 значно скоротив розрив у продуктивності між процесорами компаній AMD і Intel. Заснований на суперскалярній архітектурі RISC86, процесор К6 вважається конкурентоздатним за продуктивністю з процесором Pentium II. Він підтримує всі команди MMX, і він повністю сумісний зі всіма x86-операційними системами і програмним забезпеченням (включаючи програми, використовуючі MMX). Оскільки процесор К6 продовжував використовувати процесорний роз'єм Socket 7, що добре зарекомендував себе, він міг встановлюватися в комп'ютери замість процесорів К5 і Pentium, забезпечуючи можливість виконувати програми, які використовують команди MMX. Для правильної ідентифікації процесора К6 і його підтримки з боку системного комплекту мікросхем необхідно встановити в комп'ютер відповідну системну BIOS.

Процесор К6 містить сім паралельних виконавчих блоків, і в ньому реалізований дворівневий алгоритм прогнозу галужень програми. У разі додавання технології спекулятивного і позачергового виконання команд процесори сімейства К6 (з тактовою частотою 166-300МГц) стали серйозними конкурентами процесорам Intel Pentium MMX і Pentium II. Кеш-пам'ять першого рівня об'ємом 64 Кбайт розділена на два банки – 32 Кбайт для даних і 32 Кбайт для команд. Блок обробки чисел з плаваючою комою (відповідний стандарту IEEE 754) забезпечує продуктивність не гірше, ніж процесор Pentium MMX

К6-2 і К6-3 (1998). У 1998 році компанія AMD випустила вдосконалений процесор К6. Цифра 2 в назві К6-2 означає збільшену тактову частоту процесора і частоту шини з ядром процесора К6. Шина з робочою частотою 100 МГц підтримується системними платами Super 7 (що мають

процесорний роз'єм Socket 7 і підтримуються AGP). Значним удосконаленням процесора К6-2 є реалізація в них набору команд 3DNow!. Це набір з 21 команди, що підвищують продуктивність процесора під час обробки тривимірних зображень, виконання мультимедійних застосувань і додатків, що виконують великий об'єм операцій з плаваючою комою. Команди 3DNow! побудовані за принципом SIMD (Single Instruction Multiple Data) – одна команда, багато даних. Розширення 3DNow! також використовували в своїх процесорах інші компанії (ITD/Centaur і Cyrix). Реалізація набору мультимедійних команд 3DNow!, велика кеш-пам'ять першого рівня, що працює на частоті ядра, вбудована кеш-пам'ять другого рівня, сумісність з процесорним роз'ємом Socket 7 – ось деякі з властивостей процесора К6-2, які забезпечують його високу продуктивність і популярність. Процесор К6-3 – це процесор К6-2, в який на кристал процесора додали кеш-пам'ять другого рівня ємністю 256 Кбайт, що працює на повній частоті ядра процесора. Процесори К6-2 і К6-3 – це відповідь на процесори Pentium II/III (а конкуренція з ім'ям Pentium також важлива, як і конкуренція з продуктивністю процесора Pentium).

Athlon (1999). Після появи процесора Athlon компанії AMD змагання на ринку високопродуктивних процесорів вийшло на новий рівень. Почавши з тактової частоти 500 МГц, процесор Athlon досяг рівня більше 2 ГГц. Поява процесорів AMD Athlon і Intel Pentium 4 залишила далеко позаду всі інші компанії, що намагаються закріпитися на цьому сегменті ринку комп'ютерних технологій. Є припущення про те, що постійне удосконалення процесорів AMD, що використовують процесорний роз'єм Socket 7, стало однією з основних причин переходу компанії Intel на роботу з роз'ємами типу Slot 1. Але замість того, щоб теж розвивати лінію процесорів з роз'ємом типу Slot 1, компанія AMD вирішила, що настав час упроваджувати свої власні ідеї щодо способів інтеграції процесора в систему. Прийнявши роз'єм типу слот за основу (щоб виробникам системних плат не довелося перепроектувати розміщення компонент на платах для процесора Athlon) компанія розробила свій процесорний роз'єм і назвала його Slot A. Раз'еми Slot 1 і Slot A мають однакові розміри і число виводів, але не є сумісними.

Процесор Athlon оптимізований для роботи на високій тактовій частоті і використовує суперконвеєрну суперскалярну архітектуру. Процесор містить дев'ять виконавчих конвеєрів: три для адресних обчислень, три для операцій над цілими числами і три для виконання обчислень з плаваючою комою, набори команд 3DNow і MMX. Компанія AMD вперше використовувала конвеєрний суперскалярний виконавчий блок для обчислень з плаваючою комою, посилений технологією 3DNow, що збільшує продуктивність в іграх і додатках САПР. Згідно з деякими тестами продуктивність процесора AMD Athlon під час виконання операцій з плаваючою комою на 35% перевищує продуктивність процесора Pentium III Xeon з рівною тактовою частотою.

Процесор Athlon має кеш-пам'ять першого рівня об'ємом 128 Кбайт, а 64-розрядний контролер кеш-пам'яті другого рівня підтримує від 256 Кбайт до 8 Мбайт. Кеш-пам'ять використовує високопродуктивну системну шину, що усуває вузьке місце, пов'язане з обмеженою пропускною здатністю шини. Кеш-пам'ять другого рівня у процесорів Athlon з роз'ємом Slot A розташовується поза кристалом процесора, має об'єм 512 Кбайт і працює на частоті 2/5 або 1/3 частоти ядра. Кеш-пам'ять другого рівня у процесора Athlon з роз'ємом Socket A має розмір 512 Кбайт, розташована на одному з процесором кристалі і працює на повній частоті ядра процесора.

За заявою компанії AMD модель Athlon є процесором сьомого покоління 7x86), у його основі лежить абсолютно інша архітектура системної шини ніж та, що використовується в центральних мікропроцесорах сімейства Pentium компанії Intel.

Duron (2000). Процесор Duron (спрощена версія процесора Athlon) був випущений компанією AMD як конкурент процесору Celeron компанії Intel. У процесорі Duron використовується ядро процесора Athlon, в ньому також реалізована більшість характерних особливостей процесора Athlon: високошвидкісна системна шина, суперскалярний блок виконання операцій над числами з плаваючою комою, набір команд 3DNow, 0,18-мікронна технологія виготовлення. Єдиною відмінністю є менший об'єм кеш-пам'яті. Процесор Duron має ті ж 128 Кбайт кеш-пам'яті першого рівня, що і процесор Athlon, але тільки 64 Кбайт кеш-пам'яті другого рівня. Архітектура кеш-пам'яті в процесорах AMD спроектована таким чином, що не відбувається дублювання інформації в кеш-пам'яті першого і другого рівнів. У разі процесора Duron це означає, що його продуктивність заснована на використанні 192 Кбайт загальної кеш-пам'яті.

Athlon XP. На вихід операційної системи Microsoft Windows XP компанія AMD відповіла випуском «однойменного» процесора Athlon XP. Ця назва є чисто маркетинговою і не пов'язана з істотними змінами в архітектурі процесора. Вперше анонсовані на базі перевіреного ядра Palomino, процесори Athlon XP згодом перейшли на використання ядер Thoroughbred і Barton. Ядро Thoroughbred було першим, в якому був використаний новий для компанії технологічний процес 130 нм. У ньому також вперше була досягнута частота системної шини 333 МГц. Ядро Barton подвоїло об'єм внутрішньої кеш-пам'яті другого рівня, таким чином, її об'єм досяг 512 Кбайт. Крім того, в черговий раз була збільшена частота системної шини, яка досягла значення 400 МГц.

Athlon-64 (2003). Компанія AMD, не задовольняючись випуском 32-розрядних процесорів, розробила 64-розрядний процесор Athlon-64. Він виконаний по технології 130 нм, з використанням мідних міжз'єднань і застосуванням SOI (Silicon-on-Insulator – кремній на ізоляторі). Робоча частота системної шини – 800 МГц, при цьому сам процесор працює на частотах від 1,8

до 2,2 ГГц. Процесор підтримує всі 32-розрядні інструкції архітектури IA-32, зокрема розширення MMX, і 3DNow. Крім того, для реалізації обробки 64-бітових даних створений власний набір інструкцій AMD64. Процесор розсіює близько 90 Вт тепла (незважаючи на низьку напругу живлення – 1,5 В) і може нагріватися до температури аж до 70°С. Як оперативна пам'ять можуть використовуватися модулі DDR SDRAM об'ємом від 32 Мбайт до 4 Гбайт.

Opteron-64. Нове покоління 64-розрядних процесорів AMD відрізняється наявністю двоканального контролера пам'яті, що дозволяє використовувати до чотирьох модулів регістрової DDR SDRAM пам'яті (допустимий об'єм модуля складає від 64 Мбайт до 4 Гбайт). Opteron оптимізований для використання в двопроцесорних робочих станціях. Об'єм кеш-пам'яті другого рівня доведений до 1 Мбайт, при цьому весь кеш працює на частоті процесора. Використовується процесорний роз'єм Socket 940 і системна шина частотою 800 МГц.

AMD в 2004-му випускає перші в світі двоядерні x86-процесори Athlon 64 X2.

Аналіз багатоядерних процесорів приводиться в розділі 4.8.

4.6.3. Процесори компанії VIA Cyrix

Як альтернативний виробник мікропроцесорів компанія Cyrix заявила про себе в 1992 році, випустивши процесор Cyrix 486SLC, потім у 1993 році – процесор 486DX4. В 1995 році процесор Cyrix 5x86 (Mlsc) був єдиною серйозною альтернативою процесору AMD 5x86. Грунтуючись на зв'язках з компанією IBM, Cyrix зайняла третє місце на ринку процесорів для персональних комп'ютерів (після Intel і AMD), але не змогла подолати технологічне відставання і різницю в продуктивності процесорів, що стало перешкодою в розповсюдженні її пізніших моделей процесорів. У 1999 році торгову марку і розробки Cyrix купила компанія VIA Technologies – добре відомий розробник і виробник системних комплектів мікросхем. Через деякий час після цього компанія VIA придбала також компанію Centaur Technology. Остання була утворена в 1995 році і здобула популярність після випуску процесора WinChip. Ці два придбання дозволили компанії VIA дістати доступ на ринок процесорів.

Компанія VIA продовжила розробку високопродуктивного процесора, грунтуючись на нових технологіях компанії Cyrix, що має умовну назву «Joshua». Був створений процесор Cyrix III, заснований на ядрі Joshua CYRIX-VIA, але в серійне виробництво він не пішов. Компанія VIA також продовжила розробку ядра Centaur, і саме ця розробка була прийнята для реалізації комерційного процесора Cyrix III. Продуктивність і ціна мікропроцесора Joshua виявилася просто не конкурентоздатною на ринку недорогих процесорів.

Процесор Cyrix III, заснований на розробці компанії Centaur, отримав умовну назву «Samuel».

Серія 6x86 (1995). Компанія Cyrix випустила процесор 6x86 (названий «Ml» – остання версія була названа «M1r») в 1995 році у відповідь на процесор Intel Pentium. Він призначався для виконання як для 16-, так і для 32-розрядного програмного забезпечення. В процесорі використовувався роз'єм Socket 7, а його висока продуктивність забезпечувалася використанням двох оптимізованих суперконвеєрних виконавчих блоків і вбудованим блоком виконання операцій з плаваючою комою. Виконавчі блоки цілочисельних операцій і операцій з плаваючою комою мали високу пропускну здатність завдяки використанню таких методик, як перейменування регістрів, позачергове виконання команд, усунення залежності по даних, прогноз галуження і спекулятивного виконання. Процесор включав єдину кеш-пам'ять із зворотним записом. У багатьох відношеннях у процесорі 6x86 використовувалися ті ж технології, що і в процесорах Pentium-класу.

Існують такі версії процесорів 6x86: PR120+, PR133+, PR150+, PR166+H PR200+. (Знак «+» означає вищу продуктивність, ніж у відповідної моделі процесора Pentium).

Процесорам 6x86 властиві два недоліки. Блок обробки чисел з плаваючою комою не так добре працює, як в аналогічних процесорах компаній Intel і AMD. Хоч це і не робить істотного впливу на роботу більшості програм і операційних систем, але під час виконання програм з великим об'ємом математичних обчислень (особливо ігрових програм з тривимірною графікою) відчувається недолік продуктивності цих процесорів. У подальших версіях процесорів (наприклад, M2) блок обробки чисел з плаваючою комою працює набагато краще. Другим недоліком процесорів 6x86 є велика кількість тепла, що виділяється ними, – більш ніж у процесорів аналогічного класу AMD і Intel. З метою вирішення даної проблеми в 1996 році був випущений процесор серії 6x86L (або M1R). Буква «L» означає використання зниженого живлення. Конкретніше – процесори 6x86L використовували подвійне живлення: 3,3В для зовнішнього інтерфейсу процесора і 2,8В для роботи процесорного ядра. Для підтримки процесорів серії 6x86L системна плата повинна забезпечувати подвійну напругу живлення. **MEDIAGX (1996).** Традиційні персональні комп'ютери для обробки мультимедійної інформації використовують плати розширення, наприклад відеокарти і звукові карти. Це здорожує персональний комп'ютер і є джерелом потенційних конфліктів. У процесор Cyrix MEDIAGX були вбудовані відео- і звукові функції, разом з іншими численними функціями системної плати. Цей процесор, що має високий ступінь інтеграції і достатньо хорошу продуктивність, призначений для використання в недорогих комп'ютерах початкового рівня.

Процесор MEDIAGX є 64-розрядним пристроєм з випробуваним ядром x86-сумісного процесора. Центральний процесор безпосередньо взаємодіє з шиною PCI і динамічною пам'яттю (DRAM). Високоякісна SVGA-графіка забезпечується вдосконаленим графічним прискорювачем, вбудованим у процесор MEDIAGX. Процесор має декілька моделей з тактовою частотою від 120 до 300 МГц. Він включає єдину кеш-пам'ять першого рівня в 16 Кбайт, блок обробки чисел з плаваючою комою і функції системного управління (SMM).

Процесор MEDIAGX також працює з пам'яттю EDO RAM і підтримує до 128 Мбайт ОЗП в чотирьох банках.

Слід мати на увазі, що процесор MEDIAGX не сумісний з гніздом Socket 7. Корпус процесора призначений для поверхневого монтажу безпосередньо на системну плату, спеціально для нього призначену.

6X86MX (1997). Процесор 6x86MX (що позначається також як «M2») був відповіддю компанії Cyrix на процесори MMX-класу, такі як AMD K6 і Pentium MMX. У процесорі 6x86MX в 4 рази збільшена кеш-пам'ять (до 64 Кбайт), а тактова частота підвищена до 200 МГц і вище. Крім того, додана підтримка команд MMX, які прискорюють виконання цілочисельних операцій. Цей процесор показує оптимальну продуктивність на 16- і 32-розрядних додатках під управлінням операційних систем Windows 95/98, Windows NT, OS/2, DOS, UNIX та інших x86-операційних систем. Процесор 6x86MX має суперконвеєрну архітектуру і використовує такі передові технології, як перейменування регістрів, позачергове виконання команд програми, усунення залежності по даних, прогноз галужень програми і спекулятивне виконання програми.

Процесор 6x86MX випущений в таких варіантах: 150 МГц (PR166), 166 МГц (PR200), 188 МГц (PR233), 255 МГц (PR266), 250 МГц (PR300). Сучасні версії процесора Cyrix M2 працюють на тактовій частоті до 333МГц (PR466).

VIA/Cyrix III (1999). Процесор Cyrix III є останнім процесором компанії VIA Technologies, в якому використовується ядро Centaur XinChip (перейменоване компанією VIA в Samuel). Розроблений для сегменту ринку недорогих комп'ютерів, процесор Cyrix III використовує стандартний процесорний інтерфейс Socket 370. За ліцензійною угодою з Intel він використовує шину P6, що також сприяє зниженню вартості процесора і простоті його інтеграції в існуючі системні плати.

Моделі процесора Cyrix III, що випускаються, працюють на тактовій частоті від 500 до 700 МГц. Процесор має 128 Кбайт кеш-пам'яті першого рівня, що працює на повній частоті процесора. Кеш-пам'ять другого рівня відсутня, що знижує продуктивність процесора при виконанні додатків з тривимірною графікою. Продуктивність також обмежена тим, що виконавчий блок обробки чисел з плаваючою комою працює на половинній частоті ядра

процесора. Процесор підтримує набори мультимедійних команд 3DNow компанії AMD і MMX компанії Intel, він працює на сучасних частотах шини FSB в 66, 100 або 133 МГц.

Процесор Cyrix III виконаний за 0,18-мкм технологією і використовує тільки 10Вт потужності живлення на повній частоті роботи. Це сприяє малому виділенню тепла і робить його привабливим для установки в переносних комп'ютерах.

VIA Samuel II (2001). Процесор Samuel II, призначений стати наступником процесора Cyrix III, відрізняється від останнього наявністю доповнення у вигляді кеш-пам'яті другого рівня в 64 Кбайт і виконавчого блока обробки чисел з плаваючою комою, що працює на повній частоті процесора. Процесор Cyrix III з оптимізованою кеш-пам'яттю першого рівня (навіть з 128 Кбайт) не міг конкурувати з аналогічним процесором Intel Celeron. За допомогою процесорів серії Samuel II компанія VIA перевищила тактову частоту в 1 ГГц. Перше покоління процесорів Samuel II з ядром C5B виготовлялося за 0,15-мкм технологією. В даний час випускається вдосконалене ядро процесора, виготовлене за технологією 0,13 мкм.

У процесорі VIA Cyrix III використовується ядро з напругою живлення 1,5В, наслідком чого є практична відсутність нагріву процесорів під час їх роботи. Процесори VIA Cyrix III можуть працювати в комп'ютерах без використання вентилятора, що означає практично безшумну роботу комп'ютера. Компанією VIA також розроблено ядро процесора C5X (під назвою «Ezra»). Цей процесор включає ще більшу кеш-пам'ять другого рівня (256 Кбайт), а також здатний підтримувати набір команд SSE2 компанії Intel.

4.7. ПРОЦЕСОРИ ДЛЯ ВИСОКОПРОДУКТИВНИХ ОБЧИСЛЮВАЛЬНИХ МАШИН І СИСТЕМ

У сфері розробки і застосування процесорів для високопродуктивних обчислювальних машин і систем останніми роками відбулися значні стратегічні зміни. До середини 1980-х років високопродуктивні обчислювальні машини і системи (БПК, суперкомп'ютери) створювалися як одиничні екземпляри спеціалізованого призначення, для яких проектувалася унікальна архітектура дорогих багатокомпонентних векторно-конвеєрних процесорів, що збираються з великої кількості мікросхем. У теперішній час високопродуктивні обчислювальні машини і системи переходять у розряд таких, що серійно випускаються і відносно широко використовуються.

Основні технічні і технологічні характеристики процесорів для високопродуктивних обчислювальних машин і систем наведені в табл. 4.5.

На ринку процесорів для високопродуктивних обчислювальних машин і систем (особливо в сегменті наймогутніших з них) до останнього часу

переважали традиційні для цієї сфери застосування процесори, такі як Alpha корпорації DEC, ULTRASPARC корпорації Sun Microsystems, PA-RISC корпорації Hewlett-Packard, Power корпорації IBM, MIPS корпорації SGL. Проте лідируючі позиції починають займати виробники x86-процесоров, яким вдалося практично повністю зайняти середній і нижній цінові сегменти.

Таблиця 4.5

Основні технічні і технологічні характеристики процесорів для високопродуктивних обчислювальних машин і систем

Модель процесора	Кодове найменування процесорного ядра	Тактова частота процесора, *ГГц*	Частота системної шини, *МГц*	Кеш-пам'ять L2, *Кбайт*	Норма тех-процесу, *мкм*	Типове тепловиділення, *Вт*
Intel Xeon DP	Gallatin Nocona Irwindale	3,2 3,6 3,6	533 800 800	512 512 2048	0,13 0,09 0,09	70–75 70–75 70–75
Intel Xeon MP	Gallatin Potomac Cranford	3,0 3,33 3,66	400 400 400	512 1024 1024	0,13 0,09 0,09	67–74 67–74 67–74
Intel Itanium 2	McKinley Madison Deerfield Madison 9M	1,0 1,6 1,6 1,6	400 533 667 800	256 256 256 256	0,18 0,13 0,13 0,09	100 120 120 120
Intel Pentium D	Prescott	3,2	800	2048	0,09	69
AMD Opteron	SledgeHammer Venus/Troy/Athens Denmark/Italy	2,4 2,6 2,2	800 1000 1000	1024 1024 2048	0,13 0,09 0,09	55–95 55–95 55–95
IBM Power4+, Power5	–	1,9	800	2048	0,13	140

Компанії AMD і Intel користуються на цьому ринку приблизно тією ж стратегією, що і на мобільному. Вони пропонують адаптовані для даних умов процесори своїх модельних рядів, які, завдяки невисокій ціні і великій кількості програмного забезпечення для x86-архітектури, широко застосовуються у високо-продуктивних обчислювальних машинах і системах. Intel пропонує серію процесорів Xeon двох модифікацій. Перша з цих модифікацій – Xeon DP для робочих станцій і двопроцесорних серверів – є практично повним аналогом Pentium 4 з підтримкою багатопроцесорності. Інша модифікація – багато-процесорна версія Xeon Mp – оснащується кеш-пам'яттю третього рівня, об'єм

якої складає 1 або 2 Мбайт. Ці процесори найчастіше використовуються в чотирипроцесорних конфігураціях, вони мають 400-МГц системну шину і оснащені засобами Hyper-Threading. AMD для високопродуктивних обчислювальних машин і систем до недавнього часу пропонувала серію процесорів Athlon MP, що є повним аналогом Athlon Xp, але здатних працювати в двопроцесорних конфігураціях. Athlon MP зміг завоювати чималу популярність завдяки вигідному поєднанню ціна/продуктивність.

Подальше посилення позицій корпорацій Intel і AMD пов'язане з впровадженням 64-розрядних процесорів. Найпродуктивніші обчислювальні системи вже давно будуються на базі 64-розрядних RISC-процесорів, таких, наприклад, як вищезазначені процесори Alpha, Power, ULTRASPARC, PA-RICS. Перехід на 64 розряди дає те ж, що свого часу дав перехід від 8 до 16 і потім від 16 до 32 розрядів: значне збільшення об'єму прямоадресованої пам'яті і підвищення швидкості і/або точності деяких обчислень.

Intel і AMD під час проектування 64-розрядних процесорів використовують майже прямо протилежні підходи. Створювана архітектура Intel не є продовженням x86. Хоч можливість використання програмного забезпечення для x86 залишається. AMD ж, навпаки, пропонує 64-розрядне розширення архітектури x86, здатне ефективно працювати як на 32-, так і на 64-розрядних завданнях. 64-розрядна архітектура IA-64 розробляється Intel з 1994 року за активної участі корпорації Hewlett-Packard. Ця архітектура має ідеологію VLIW і реалізована в її вдосконаленому варіанті EPIC. Архітектура IA-64 вперше була застосована Intel в 2001 році в процесорі *Itanium* з ядром Merced. Це складний і дорогий процесор, що містить 325 млн. транзисторів: 25 млн. у процесорному ядрі, включаючи кеш-пам'ять першого і другого рівнів об'ємом 32 і 96 Кбайт відповідно, і по 75 млн. в кожній з чотирьох додаткових мікросхем, що складають у сумі кеш-пам'ять третього рівня об'ємом 4 Мбайт. Itanium підтримує до 4 включених паралельно процесорів (для їх більшої кількості необхідний відповідний міст). На цьому процесорі вдалося досягти тактової частоти 800 МГц. Наступним кроком став випуск у 2002 році процесора *Itanium 2* на базі ядра McKinley. В порівнянні з Merced він швидший, компактніший і набагато кращий за всіма основними параметрами. В Itanium 2 кеш-пам'ять другого рівня збільшена до 256 Кбайт, кеш-пам'ять третього рівня (до 3 Мбайт) інтегрована в процесорне ядро.

Пропоноване AMD 64-розрядне вирішення x86-64 реалізується в архітектурі *Hammer*, що відноситься до восьмого покоління процесорів x86. Архітектура Hammer достатньо близька до архітектури процесора Athlon, тобто є її 64-розрядним розширенням. Також є дев'ять виконавчих блоків і конвеєр завдовжки 12 рівнів. Довжина регістрів збільшена до 64 біт, додано декілька нових регістрів. Введена деяка кількість нових 64-розрядних інструкцій. Підтримується новий і дуже ефективний набір операцій з плаваючою крапкою,

що використовує 16 нових 128-розрядних регістрів. Об'єм кеш-пам'яті другого рівня – до 1 Мбайт.

64-розрядний процесор AMD *Opteron* представлений в 2003 році. Він стартував з частоти 1400 МГц. Opteron має багато переваг як перед 64-розрядними серверними процесорами Intel, так і перед традиційними 64-розрядними процесорами Alpha, Power PC, ULTRASPARC, MIPS. Основний плюс Opteron: при відносно невисоких значеннях ціни і тепловиділення ці процесори мають високую продуктивність. Важливим моментом слід визнати і те, що розробка корпорації AMD, незважаючи на 64-розрядну архітектуру, вже зараз забезпечена достатньою кількістю програмного забезпечення, яка завдяки своїй зворотній сумісності з 32-розрядною системою команд x86 дозволяє без втрат і навіть з виграшем у швидкодії виконувати звичайний 32-бітовий програмний код.

4.8. БАГАТОЯДЕРНІ ПРОЦЕСОРИ

Багатоядерні процесори містять декілька процесорних ядер в одному корпусі (на одному або декількох кристалах). Процесори, призначені для роботи однієї копії операційної системи на декількох ядрах, є високо-інтегрованою реалізацією мультипроцесора.

Двоядерність процесорів включає такі поняття, як наявність логічних і фізичних ядер: наприклад, двоядерний процесор Intel Core Duo складається з одного фізичного ядра, яке у свою чергу розділене на два логічних. Процесор Intel Core 2 Quad складається з чотирьох фізичних ядер, що істотно впливає на швидкість його роботи.

10 вересня 2007 року були випущені в продаж нативні (у вигляді одного кристала) чотирьох'ядерні процесори для серверів AMD Opteron, що мали в процесі розробки кодову назву AMD Opteron Barcelona. 19 листопада 2007 року вийшов у продаж чотирьох'ядерний процесор для домашніх комп'ютерів AMD Phenom. Ці процесори реалізують нову мікроархітектуру K8L (K10).

27 вересня 2006 року Intel продемонструвала прототип 80-ядерного процесора. Передбачається, що масове виробництво подібних процесорів стане можливе не раніше переходу на 32-нанометровий техпроцес.

26 жовтня 2009 року Tilera анонсувала 100-ядерний процесор широкого призначення серії TILE-Gx. Кожним процесорним ядром є окремий процесор з кешем 1, 2 і 3 рівнів. Ядра, пам'ять і системна шина зв'язані за допомогою технології Mesh Network. Процесори виготовляються по 40-нм нормах техпроцесу і працюють на тактовій частоті 1,5 ГГц. Випуск 100-ядерних процесорів призначався на 2011 рік.

На даний момент масово доступні двох-, чотирьох- і шестиядерні процесори, зокрема Intel Core 2 Duo на 65-нм ядрі *Conroe* (пізніше на 45-нм ядрі

Wolfdale) і Athlon 64 X2 на базі мікроархітектури K8. У листопаді 2006 року вийшов перший чотирьох'ядерний процесор Intel Core 2 Quad на ядрі *Kentsfield*, що є збіркою з двох кристалів Conroe в одному корпусі. Нащадком цього процесора став Intel Core 2 Quad на ядрі *Yorkfield* (45 нм), архітектурно схожому з *Kentsfield*, але він має більший об'єм кеша і робочі частоти.

Компанія AMD пішла по власному шляху, виготовляючи чотирьох'ядерні процесори єдиним кристалом (на відміну від Intel, перші чотирьох'ядерні процесори якої є фактично склеюванням двох двоядерних кристалів). Незважаючи на всю прогресивність подібного підходу, перший «чотирьох'ядерник» фірми, який отримав назву AMD Phenom X4, вийшов не дуже вдалим. Його відставання від сучасних йому процесорів конкурента складало від 5 до 30 і більше відсотків залежно від моделі і конкретних завдань.

В 2009 році обидві компанії відновили свої лінійки чотирьох'ядерних процесорів.

Intel представила сімейство Core i7, що складається з трьох моделей, які працюють на різних частотах.

Core i7 (*Bloomfield, Lynnfield* або *Gulftown*) – чотирьох- або шестиядерний процесор останнього покоління, призначений для настільних комп'ютерів вищого класу. Вперше представлений у листопаді 2008 року. Чотирьох'ядерні *Bloomfield* і *Lynnfield* виготовляються по 45-нм технології, шестиядерні *Lynnfield* – по 32-нм технології. Мікроархітектура Nehalem має три рівні кеш-пам'яті: L1 – 64 Кбайт (32 Кбайт для даних і 32 Кбайт для інструкцій) для кожного ядра; L2 – 256 Кбайт для кожного ядра; L3 – 8 або 12 Мбайт, є загальною для всіх ядер. Особливостями даного процесора є:

- Природжена чотирьох'ядерна будова. Єдиний процесорний кристал включає чотири ядра з 256-кілобайтним кешем L2 і загальний кеш L3, що розділяється.
- Заміна процесорної шини Quad Pumped Bus новим послідовним інтерфейсом QuickPath з топологією крапка-крапка, який може використовуватися не тільки для з'єднання процесора і чипсета, але і для зв'язку процесорів між собою.
- Вбудований у процесор контролер пам'яті, що підтримує триканальну DDR3 SDRAM - 1066/1333 Мгц із напругою до 1,6В. При цьому кожен канал здатний працювати з двома модулями DIMM. Підтримка технології SMT (Simultaneous multithreading), яка є аналогічною пам'ятній технології Hyper-Threading. Завдяки їй кожне ядро Core i7 може виконувати два обчислювальні потоки одночасно, внаслідок чого процесор представляється в операційній системі вісьма ядрами.
- Кеш третього рівня, що розділяється, загальним об'ємом 8 Мбайт.

- Вбудований мікроконтролер PCU, незалежно керуючий напругою та частотою кожного з ядер і володіючий можливостями автоматичного розгону окремих ядер при пониженому навантаженні на інші ядра.
- Підтримка нового набору інструкцій SSE4.2.

Core i7 проводиться за технологією з нормами виробництва 45 нм, складається з 731 млн. транзисторів і має площу ядра 263 кв.мм.

Базова тактова частота для всіх моделей Core i7 – 133 Мгц, номінальні частоти досягаються застосуванням множників. В модифікаціях Core i7 Extreme множник розблокований, що дозволяє безперешкодно підвищувати тактову частоту процесора.

Сумісні набори системної логіки: серія 8xx – Intel H55 Express, H57 Express, P55 Express, Q57 Express, серія 9xx – Intel X58 Express.

Слабкою стороною платформи, використовуючою Core i7, є її надмірна вартість, оскільки для установки даного процесора необхідна дорога материнська плата на чипсеті Intel X58 і триканальний набір пам'яті типу DDR3, що також має на даний момент високу вартість.

Техпроцес супершвидкісного процесора виробництва Intel – Core i7 з вісьма ядрами планується зменшити з 45 до 32 нм. Завдяки підтримці триканальної пам'яті DDR3 об'єм RAM в 6 Гбайт стане стандартом, як і необхідні для цього 64-бітові операційні системи.

Core i5 (*Clarkdale* або *Lynnfield*) – дво- або чотирьохядерний процесор останнього покоління, призначений для настільних комп'ютерів середнього рівня. Вперше представлений 8 вересня 2009 року. Встановлюється в розєми LGA1156. Двоядерні *Clarkdale* виготовляються по 32-нм технології, чотирьохядерні *Lynnfield* – по 45-нм технології. Мікроархітектура Nehalem. Має три рівні кеш-пам'яті: L1 – 64 Кбайт (32 Кбайт для даних і 32 Кбайт для інструкцій) для кожного ядра; L2 – 256 Кбайт для кожного ядра; L3 – 4 або 8 Мбайт, є загальною для всіх ядер. Процесор оснащений вбудованим двоканальним контролером оперативної пам'яті DDR3-1066/1333 Мгц з напругою до 1,6В. Модулі, що розраховані на більш високу напругу, не будуть працювати з цим чипом і навіть можуть його зашкодити.

Оснащений вбудованим контролером PCI Express 2.0 x16, завдяки якому графічний прискорювач може підключатися прямо до процесора. В моделях із вбудованим графічним ядром GMA HD до чипа може підключатися одна відеокарта в режимі x16, в моделях без вбудованої графіки – дві відеокарти в режимі x8 кожна.

Для з'єднання з набором системної логіки застосовується шина DMI (Digital Media Interface) с пропускною здатністю 2 Гбайт/с.

У двоядерних моделях (серія 6xx) вбудований графічний адаптер GMA HD і реалізована технологія Hyper-Threading, в чотирьох'ядерних (серія 7xx) графіки і Hyper-Threading немає. В моделях, номер яких закінчується на 1, тактова частота графіки становить 900 Мгц, в моделях, номер яких закінчується на 0, графічне ядро працює на частоті 733 Мгц.

У всіх Core i5 реалізована технологія автоматичного підвищення тактової частоти Turbo Boost у ресурсномістких завданнях.

Базова тактова частота для всіх моделей Core i5 – 133 Мгц, номінальні частоти досягаються застосуванням множників.

Сумісні набори системної логіки: Intel H55 Express, H57 Express, P55 Express, Q57 Express.

Core i3 (*Clarkdale*) – двоядерний процесор останнього покоління, призначений для настільних комп'ютерів початкового рівня. Вперше представлений 7 січня 2010 року. Мікроархітектура Nehalem. Встановлюється в роз'єми LGA1156. Виготовляється по 32-нм технології.

Процесор Core i3 використовує три рівні кеш-пам'яті: L1 – 64 Кбайт (32 Кбайт для даних і 32 Кбайт для інструкцій) для кожного ядра; L2 – 256 Кбайт для кожного ядра і L3 – 4 Мбайт, є загальною для всіх ядер.

Оснащений вбудованим двоканальним контролером оперативної пам'яті DDR3-1066/1333 МГц із напругою до 1,6В. Модулі, що розраховані на більш високу напругу, не будуть працювати з цим чипом і навіть можуть його зашкодити.

Оснащений вбудованим контролером PCI Express 2.0 x16, завдяки якому графічний прискорювач може підключатися прямо до процесора. Для з'єднання з набором системної логіки застосовується шина DMI (Digital Media Interface) з пропускною здатністю 2 Гбайт/с.

У процесори Core i3 вбудоване графічне ядро GMA HD з дванадцятьома конвеєрами і тактовою частотою 733 Мгц.

Базова тактова частота для всіх моделей Core i3 – 133 МГц, номінальні частоти досягаються застосуванням множників.

Сумісні набори системної логіки: Intel H55 Express, H57 Express, P55 Express, Q57 Express.

Компанія AMD у свою чергу перевела випуск своїх CPU на основі мікроархітектури K10 на 45-нанометровий технологічний процес – з'явилися процесори AMD Phenom II. Від AMD Phenom вони відрізняються не тільки досконалішою технологією виробництва, але і злегка переробленим ядром, окремі блоки якого були оптимізовані і допрацьовані. Процесор має три рівні кеш-памяті: L1 – об’ємом 128 Кбайт на кожне ядро, L2 – об’ємом 512 Кбайт на кожне ядро і L3 – об’ємом 6 Мбайт. Новий процесор увійшов до складу платформи, що отримала назву Dragon.

Процесори модельного ряду AMD Phenom II виготовляються методом іммерсійної літографії по 45-нанометровому техпроцесу. Це перші процесори AMD, у ході виробництва яких застосовується така досконала технологія. Нагадаємо, що Intel почала використовувати 45-нанометровую технологію для виробництва мікропроцесорів ще в кінці 2007 року. Модельний ряд AMD Phenom II складається з двох чотирьох'ядерних CPU для роз'єму Socket Ам2+ – топового AMD Phenom II X4 940 Black Edition з розблокованим множником і частотою 3 ГГц і AMD Phenom II X4 920, що працює на частоті 2,8 ГГц.

AMD Phenom II X4 містить 758 млн. транзисторів, площа його кристала складає 258 мм^2. Процесор попереднього покоління Phenom X4 складається з 450 млн. транзисторів, розміщених на площі 285 мм^2. Більш ніж півторакратне збільшення кількості транзисторів обумовлено тим, що:

- всі чотири ядра в обох моделях розміщені на одному кристалі;
- нові процесори сумісні зі всіма моделями материнських плат, що є на сьогоднішній день, з роз'ємом Socket Ам2+ (для установки Phenom II потрібно буде всього лише відновити BIOS). AMD продовжує наслідувати концепцію зворотної сумісності, і процесори Phenom II також можуть працювати в деяких платах з роз'ємом Socket Ам2.

У 2011 році Intel анонсує, а потім трохи пізніше випускає партію нових процесорів на архітектурі Sandy Bridge, для нового, що вийшов в тому ж році сокета LGA 1155. Це друге покоління сучасних процесорів Intel.

Цікаво, що в тому ж 2011, вийшов новий сокет LGA 2011, для якого вийшли два супер-процесора i7 4820K (4 ядра, 8 потоків, з L3 кешем - 10 Мб) і i7 4930K (6 ядер, 12 потоків, L3 кеш дорівнює цілих 12 Мб).

У 2012 Intel випускає 3 покоління процесорів, під назвою Ivy Bridge. Нове покоління процесорів, базується на сокеті - LGA 1155.

Провідні процесори четвертого покоління (Haswell-E): i7-5960X Extreme Edition, i7-5930K і 5820K, адаптовані під серверні рішення. Вони базуються на новому 2011 v3 сокеті і продуктивність у них виняткова, що не дивно, адже у старшого процесора в лінійці цілих 16 потоків і 20 Мб кеша.

У 2015 виходить Skylake, на сокеті 1151. Процесори цього покоління відрізняється від усіх попередніх: по-перше, зменшеними розмірами теплорозподільної кришки, для поліпшеного теплообміну з системою охолодження на процесорі, по-друге, підтримкою пам'яті DDR4.

У 2016 році Intel представила п'яте покоління процесорів − Broadwell-E. Core i7-6950X був перший в історії десктопний десятіядерний процесор в світі.

2 березня 2017 го року в продаж надійшли нові процесори старшої лінійки AMD Ryzen 7, що включали в себе 3 моделі: 1800x, 1700x і 1700. На сьогодні старший Ryzen - найшвидший восьміядерний процесор в світі.

Зараз компанії AMD і Intel представляють нові флагманські процесори. У AMD це Ryzen Threadripper, у Intel – Core i9. Процесор Intel Core i9-7980XE це вісімнадцяти ядерний тридцяти шести потоковий процесор. Intel Core i9-7960X це шістнадцяти ядерний тридцяти двох потоковий процесор. Процесор AMD Ryzen Threadripper 1950X – шістнадцяти ядерний тридцяти двох потоковий.

КОНТРОЛЬНІ ПИТАННЯ

1. Які функції виконує процесор в обчислювальній машині?
2. Опишіть узагальнену структуру процесора.
3. Що таке процесорні регістри і яке їхнє призначення?
4. Від яких параметрів залежить продуктивність процесора?
5. Які операції виконуються в АЛП? Як залежно від реалізації цих операцій підрозділяються АЛП?
6. Чим відрізняються АЛП блокового типу від багатофункціональних АЛП?
7. Опишіть структуру АЛП найпростішого мікропроцесора.
8. Охарактеризуйте склад операційних пристроїв, що входять в АЛП. З яких міркувань і яким чином він може змінюватися?
9. Поясніть поняття "операційні пристрої з жорсткою структурою". Які їхні переваги і недоліки?
10. Чим обумовлена назва операційних пристроїв з магістральною структурою? Порівняйте магістральні структури з жорсткими структурами.
11. Чим обумовлена специфіка цілочисельного додавання і віднімання? Яку роль відіграє в них додатковий код?
12. Сформулюйте сутність цілочисельного «традиційного» множення. Як враховуються знаки співмножників?
13. Охарактеризуйте сутність логічних методів прискорення множення.
14. Поясніть сутність апаратних методів прискорення множення.
15. Порівняйте організацію цілочисельного ділення з відновленням залишку і без відновлення залишку. Як враховуються у ході ділення знаки операндів?
16. В чому полягає особливість операції множення в АЛП з плаваючою комою?
17. Опишіть загальні принципи побудови пристрою управління (ПУ).
18. Вкажіть основні відмінності ПУ з жорсткою логікою від ПУ зі збереженою в пам'яті логікою.

19. Назвіть способи кодування мікрокоманд (МК). Наведіть для кожного способу схему кодування МК.

21. Опишіть механізм конвеєрної обробки команд у процесорі.

22. Охарактеризуйте можливі конфліктні ситуації в конвеєрі процесора і запропонуйте способи їх вирішення.

23. На основі яких механізмів вирішуються проблеми умовних переходів у конвеєрі команд процесора?

24. Чим відрізняється статичний метод прогнозування переходів у конвеєрі процесора від динамічного методу?

25. Дайте поняття суперскалярної архітектури процесора.

26. Опишіть векторні і матричні схеми роботи процесорів.

27. Дайте загальну характеристику процесорів сімейства Intel Pentium.

28. Назвіть основні відмінності в архітектурі і параметрах сучасних процесорів корпорацій Intel і AMD.

29. Які переваги дає перехід до 64-розрядної архітектури процесорів.

5. ОРГАНІЗАЦІЯ СИСТЕМНОГО ІНТЕРФЕЙСУ ТА АРХІТЕКТУРА СИСТЕМНОЇ ПЛАТИ

5.1. ПОНЯТТЯ ІНТЕРФЕЙСУ ТА ЙОГО ХАРАКТЕРИСТИКИ

Апаратні інтерфейси є одним з основних компонентів обчислювальної системи із змінним складом обладнання. Вони дозволяють робити обмін даними і управляючою інформацією між пристроями фізичної структури ОС за інформаційними правилами.

Уніфікація правил взаємодії забезпечує можливість підключення до ОС різноманітних периферійних пристроїв (ПП), які відрізняються призначенням, швидкодією та принципами дії.

Апаратним інтерфейсом прийнято називати сукупність правил уніфікованої взаємодії між окремими пристроями, а також сукупність апаратних, програмних і конструктивних засобів, необхідних для реалізації цих правил [9].

Взаємодія здійснюється з допомогою сигналів, що передаються під дією електричних (або оптичних) ланцюгів, які називаються лініями інтерфейсу.

Сукупність ліній, згрупованих за функціональним призначенням, прийнято називати шиною інтерфейсу.

Уніфікація правил взаємодії направлена на забезпечення інформаційної, електричної і конструктивної сумісності; а саме уніфікація і стандартизація лежать в основі побудови інтерфейсів.

Інформаційна сумісність досягається за рахунок єдиних вимог, що ставляться до структури і складу ліній інтерфейсу, алгоритмів взаємодії, засобів кодування і форматів даних, управляючої і адресної інформації, часовим співвідношенням між сигналами.

Електрична сумісність означає узгодженість параметрів електричних або оптичних сигналів, які передаються середовищем інтерфейсу, відповідність логічних сигналів, що передаються середовищем інтерфейсу, відповідність логічних станів рівням сигналів. Електрична сумісність визначає вимоги до навантажувальної можливості компонентів і характеристик ліній передачі, що використовуються (довжина, допустимі активне і реактивне навантаження, порядок підключення схем узгодження і т.д.).

Конструктивна сумісність означає можливість механічного з'єднання електричних ланцюгів, а іноді і механічної заміни деяких блоків, цей вид сумісності забезпечується стандартизацією з'єднувальних елементів (роз'ємів, штекерів і т.п.) кабелів, конструкцій плат і т.п.

Інтерфейси в системах вводу/виводу (СВВ) виникають між різними рівнями ієрархії фізичного складу ОС, тому вимоги, які ставляться до організації обміну, істотно відрізняються.

Єдиний стандартний інтерфейс не зміг би забезпечити ефективну роботу різноманітних пристроїв, що використовуються на різних рівнях ієрархії СВВ. Цим пояснюється наявність системи інтерфейсів різних рангів, які відрізняються своїми характеристиками і ступенем уніфікації.

Залежно від вимог уніфікації виділяють:

- фізичну реалізацію інтерфейсу, тобто склад і характеристики ліній передачі, конструкцію засобів їх підключення (наприклад, роз'єм), вид і характеристики сигналів;
- логічну реалізацію інтерфейсу, тобто протоколи взаємодії або алгоритми формування сигналів обміну.

Система апаратних інтерфейсів є однією з основних частин поняття архітектури ОС [9]. В структурі ОС з виділеними процесорами вводу/виводу (ПВВ) відмітимо інтерфейси чотирьох рангів (рис. 5.1).

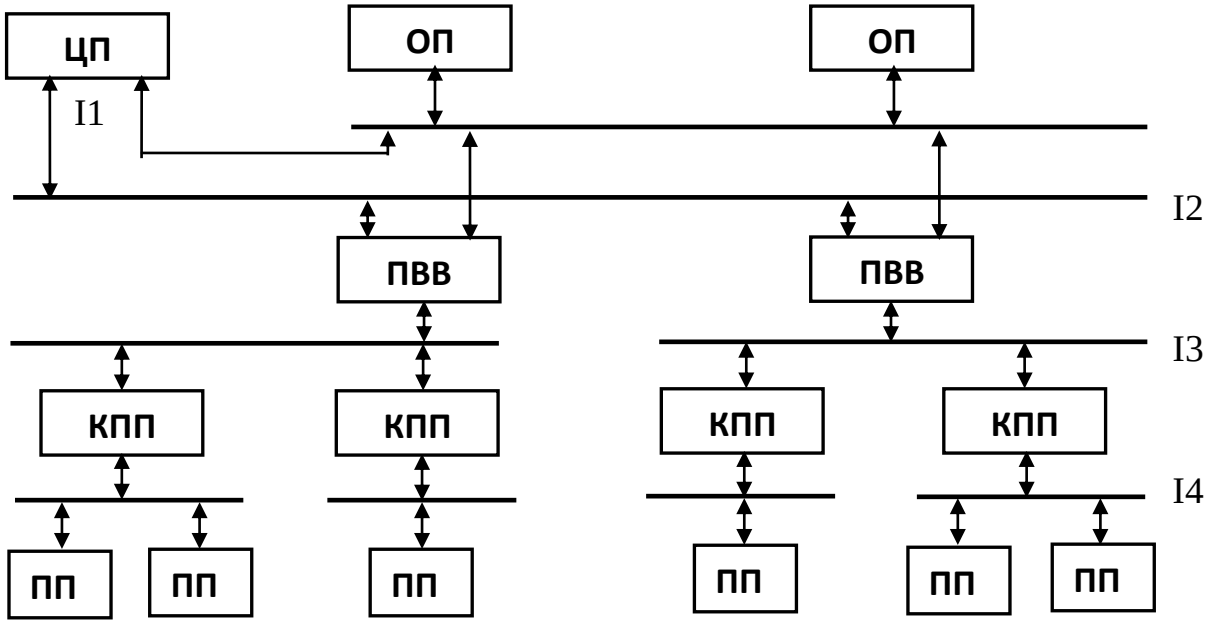


Рис. 5.1. Інтерфейси великих ЕОМ

Через інтерфейс I1 проводиться обмін інформацією між ОП і процесорами (центральним − ЦП або ПВВ). Через інтерфейс I2 – управляючою інформацією між ЦП і ПВВ. Інтерфейси I1 та I2 є внутрішніми, які відображають особливості певної моделі і не уніфікуються.

Інтерфейси вводу/виводу (I3) забезпечують обмін між ПВВ і контролерами ПП (КПП); вони стандартизуються, що дає можливість використовувати однакові контролери і ПП в різних моделях ЕОМ однієї системи.

Інтерфейси I4 створюють групу так званих «малих» інтерфейсів, під впливом яких власне ПП й поєднується з контролером. Ступінь уніфікації

малих інтерфейсів залежить від типу ПП і контролера. Так, якщо контролер призначений для управління тільки одним ПП і конструктивно об'єднаний разом з ними, то їх інтерфейс не уніфікується.

Якщо ж контролер призначений для одночасного обслуговування множини ПП, то відповідний малий інтерфейс повинен бути стандартизований.

У разі підключення апаратури систем передачі даних відповідні інтерфейси прийнято називати стиками.

Для міні- і мікро-ЕОМ характерна (рис. 5.2) наявність інтерфейсу І0, за допомогою якого зв'язані між собою ЦП, ОП і контролери. Цей інтерфейс прийнято називати системним (або об'єднаним), він уніфікований для всього сімейства ЕОМ.

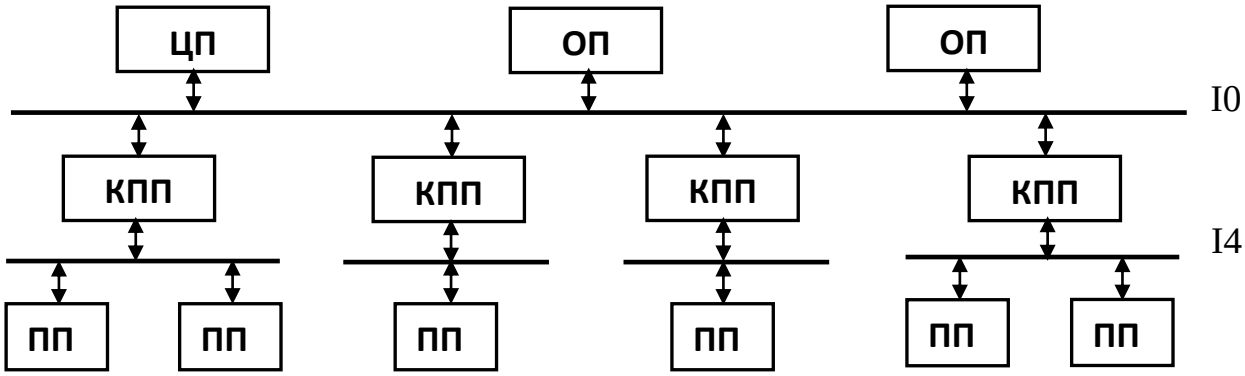


Рис. 5.2. Інтерфейси міні- і мікро-ЕОМ

Контролери в міні- і мікро-ЕОМ достатньо прості, оскільки управління обміном між ПП і ОП робиться в значній мірі програмним шляхом. Це дозволяє для сімейства ЕОМ з різними інтерфейсами І0 використовувати однакові ПП (але з різними контролерами).

Інтерфейси прийнято характеризувати такими параметрами:

- *видом зв'язку*, тобто можливістю вести дуплексну (повідомлення можуть одночасно передаватися у двох напрямках, що потребує двох каналів зв'язку), напівдуплексну (повідомлення можуть передаватись в двох напрямках, але одночасно можлива передача тільки в одному) або симплексну передачу (повідомлення можуть передаватись тільки в одному напрямку);
- *пропускною здатністю*, тобто кількістю інформації, що передається через інтерфейс за одиницю часу;
- *максимально допустимою відстанню між пристроями*, або сумарною довжиною ліній, що з'єднують всі пристрої інтерфейсу;
- *затримками під час організації передачі*, що викликані необхідністю виконання підготовчих та закінчуючих дій по встановленню зв'язку між пристроями.

Конкретні значення цих параметрів залежать від багатьох факторів, частково від інформаційної ширини інтерфейсу, способу синхронізації,

середовища інтерфейсу, топологічної структури з'єднань і організації ліній інтерфейсу, суміщення або функціонального розділення ліній. Всі ці фактори визначають організацію інтерфейсу.

5.2. ОРГАНІЗАЦІЯ ІНТЕРФЕЙСІВ

Організація інтерфейсів визначається способом передачі інформації (паралельної або послідовної, асинхронної або синхронної), з'єднанням пристроїв і використанням ліній [9].

5.2.1. Послідовна і паралельна передача інформації

Цифрові повідомлення можуть передаватись у послідовній і паралельно-послідовній формі; відповідно інтерфейси прийнято ділити на послідовні і паралельні.

У послідовному інтерфейсі передача даних здійснюється всього по одній лінії, хоч загальне число ліній може бути і більшим. Інтерфейси послідовного типу характеризуються відносно невеликими швидкостями передачі і низькою вартістю мережі зв'язку. Вони можуть використовуватись для підключення низькошвидкісних ПП, що розташовані на значних відстанях від центрального ядра ЕОМ (інтерфейси рангу І4).

У паралельному інтерфейсі передача повідомлення виконується послідовно квантами, що мають m байт. Кожний квант передається одночасно по m лініях. Величина m має назву *ширина інтерфейсу* і, як правило, дорівнює або кратна байту.

Найбільш розповсюджені паралельні інтерфейси, в яких $m = 8$ або $m = 16$. Для внутрішніх інтерфейсів рангу І1 і І2 високопродуктивних ЕОМ ширина інтерфейсу може бути значно більшою.

Розкид параметрів середовища інтерфейсу викликає неоднакові викривлення фронтів і затримки сигналів, що передаються по різних лініях $Л_1 \div Л_m$. Це означає, що одночасно видані передавачем ПРД сигнали на лінії $Л_1 \div Л_m$ сприймаються приймачем ПРМ неодночасно, а в інтервалі (t_1, t_2) (рис. 5.3, *а* і *б*).

Таке явище називається *перекосом інформації*. В інтервалі (t_1, t_2) приймач може сприйняти будь-яку кодову комбінацію (X_i), $i = (1, m)$, яка відрізняється від комбінації (B_i), що повинна бути прийнята.

Для виключення можливості прийому неправильної кодової комбінації в паралельних інтерфейсах вводять додаткову лінію стробування.

Сигнал стробу *STR*, що передається по ній, повинен надійти в приймач ПРМ в момент t_{str}, який відповідає закінченню встановлення на входах ПРМ стану (B_i), тобто в момент, коли виконується умова $t_{str} > t_2$. При цьому необхідно передати сигнал *STR* з затримкою відносно моменту видачі інформації сигналів на лінії $Л_1 \div Л_m$:

$$\tau_{str} > 2max\ (\Delta t_{ij}) = 2max\ |t_i - t_j|\ ,$$

де t_i, t_j – найраніший і найпізніший моменти надходження сигналів у приймач ПРМ по лініях i та j, відповідно у разі одночасної їх видачі передавачем;

$\Delta\ \tau_{ij}$ – можливий розкид моментів надходження сигналів по лініях $Л_1 \div Л_m$, а $\Delta\ \tau_{str}$ – по лінії стробу.

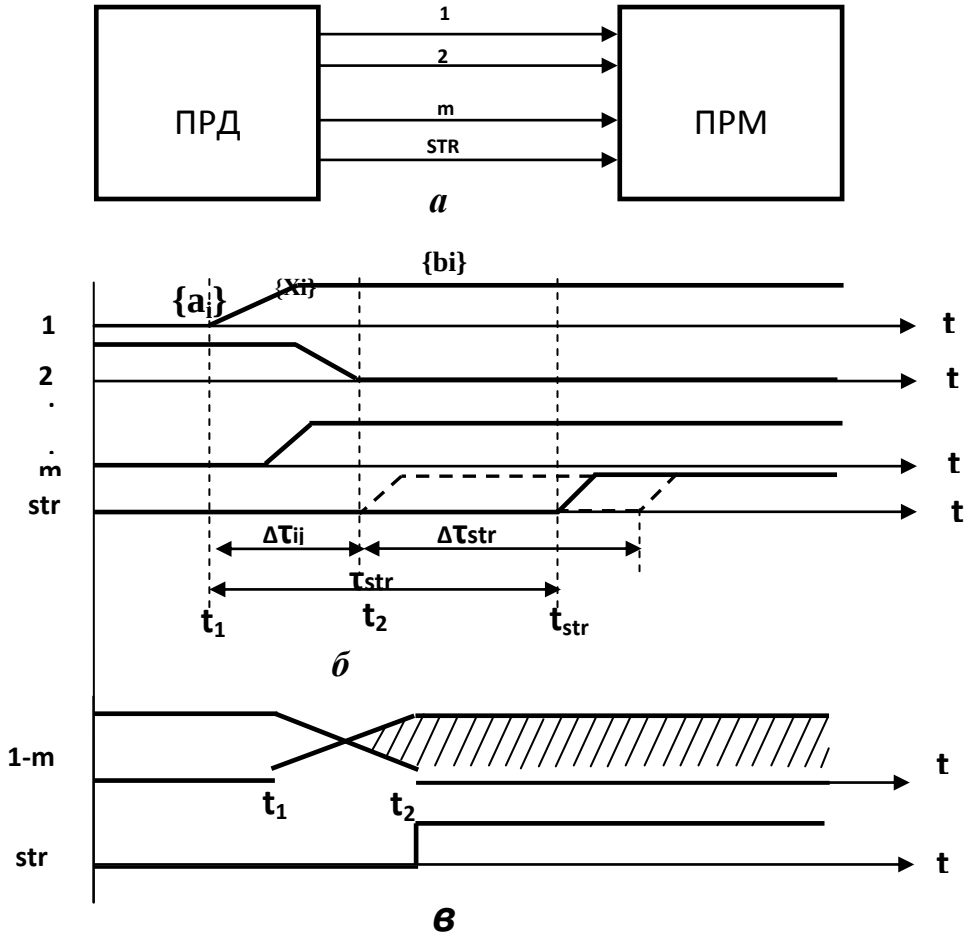


Рис. 5.3. Часові діаграми сигналів паралельного інтерфейсу

Далі будемо використовувати умовну форму часової діаграми (рис.5.3, *в*,) на якій паралельна передача сигналів по лініях $Л_1 \div Л_m$ обумовлена однією широкою смугою, звужена частина якої відповідає інтервалу перекосу (t_1, t_2), строб показаний у вигляді сигналу ідеальної форми в момент завершення інтервалу перекосу.

5.2.2. Синхронна і асинхронна передача інформації

Взаємодія передатчика ПРД і приймача ПРМ передбачає узгодження в часі моментів передачі й прийому квантів інформації.

Під час синхронної передачі передавач ПРД підтримує постійні інтервали між черговими квантами інформації в процесі передачі всього повідомлення або значної його частини.

Приймач ПРМ незалежно або з допомогою сигналів управління, що надходять від передавача, забезпечує прийом квантів у темпі їх видачі.

Для реалізації синхронного режиму передачі у разі послідовного інтерфейсу передавач ПДР на початку повідомлення передає завчасно обумовлену послідовність бітів, яка називається символом синхронізації *SYN*.

Перехід лінії інтерфейсу із стану «0» в стан «1» використовується приймачем для запуску внутрішнього генератора, частота якого збігається з частотою генератора в передавачеві; приймач ПРМ розпізнає символ *SYN*, що передається, після чого приймає наступний символ повідомлення, починаючи з його першого біта. Цей процес проілюстрований на рис. 5.4.

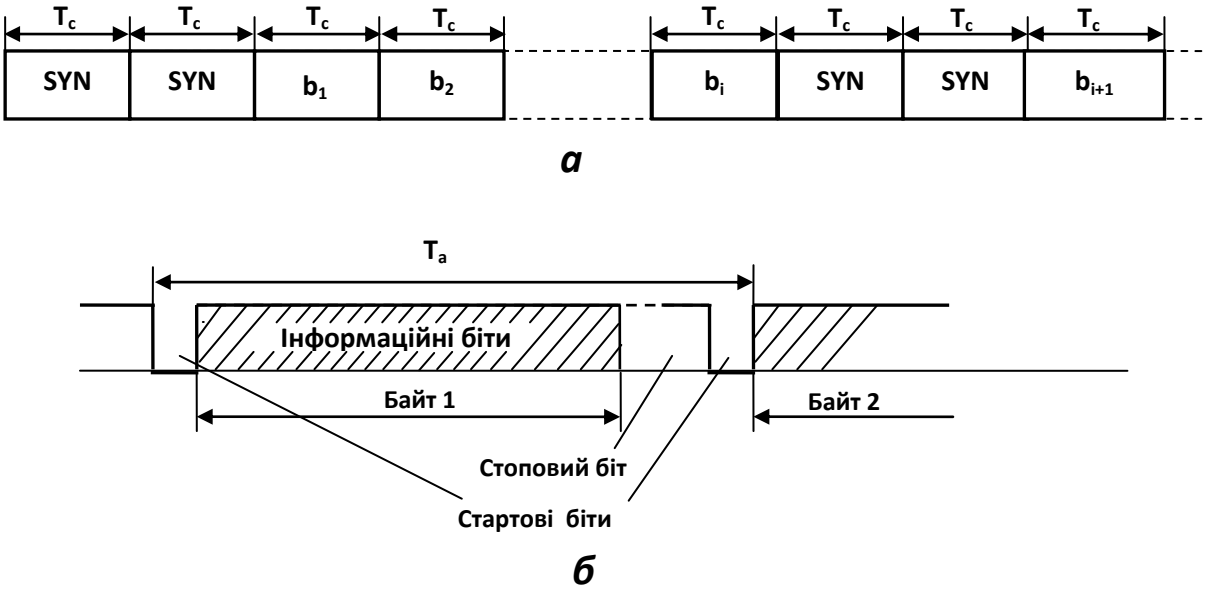


Рис. 5.4. Синхронна (*а*) і асинхронна (*б*) передача в послідовному інтерфейсі

Стабільність інтервалів передачі (прийому) символів забезпечується синхронно працюючими незалежними генераторами в передавачеві і приймачеві, які мають високу стабільність частоти.

У разі порушення синхронізації передавач повинен вставити в послідовність байт повідомлень, що передаються, додаткові символи *SYN*.

Якщо під час послідовної передачі використовуються додаткові лінії інтерфейсу, то синхронна передача підтримується сигналами синхронізації, які передаються по лініях управління.

Аналогічно з допомогою сигналу синхронізації реалізується синхронна передача в паралельному інтерфейсі. Як сигнал синхронізації використовується стробуючий сигнал.

Наступний квант інформації передається тільки після того, як попередній квант прийнятий, зафіксований і розпізнаний у приймачеві.

Якщо передача повідомлень через інтерфейс виконується між передавачем ПРД і одним з кількох приймачів ПРМ, то інтервал синхронізації *T* встановлюється з розрахунку на найбільш повільний приймач ПРМ, тобто

$$Tc \geq max\ Tc_i\ .$$

Передачу називають асинхронною, якщо синхронізація передавача й приймача здійснюється під час передачі кожного кванта інформації. Інтервал між передачею кожного кванта нестабільний. У разі послідовного інтерфейсу кожен байт, який передається, «обрамляється» стартовими і стоповими сигналами, як показано на рис. 5.4, *б*.

Стартовий сигнал змінює стан лінії інтерфейсу і служить для запуску генератора в приймачеві, стоповий сигнал переводить лінію в початковий стан і зупиняє роботу генератора.

Таким чином, синхронізація передавача й приймача підтримується тільки в інтервалі передачі одного байта.

У разі паралельного інтерфейсу режим асинхронної передачі звичайно реалізується за схемою «запит – відповідь».

Приймач ПРМ, отримавши сигнал по лінії стробу і зафіксувавши байт повідомлення по лініях Л1 ÷ Лm, формує відповідний сигнал: квитанцію *RCP*, який пересилається в передавач ПРД; таку передачу називають передачею з квитуванням.

Сигнал *RCP* є дозволом передавачеві перевести лінії Л1 ÷ Лm і лінію стробування в попередній стан, після чого приймач ПРМ також скидає сигнал *RCP*. Скид сигналу *RCP* служить для передавача дозволом на передачу наступного байта.

Квитування дозволяє як би підлатувати темп обміну під кожний конкретний пристрій і забезпечити в ряді випадків високий темп обміну, не дивлячись на необхідність передачі сигналів в двох напрямках. Крім того, квитування забезпечує високу надійність передачі і достовірність даних, що передаються.

Однак під час передачі з квитуванням може виникнути ситуація, за якої процес обміну припиняється через відмову, яка спричинить відсутність сигналу квитанції. Виявлення подібних ситуацій базується на вимірюванні інтервалу часу, протягом якого передавач гарантовано повинен отримати сигнал-квитанцію. Якщо за цей встановлений інтервал сигнал Тто передавачем не буде отриманий, то фіксується відмова. Такий контроль називається контролем за тайм–аутом, а інтервал Тто – інтервалом тайм–ауту, величина якого повинна відповідати умові

$$T_{TO} \geq \max\ \{\ Ta_i\ \},$$

де Ta_i – можливі інтервали між видачею квантів інформації пристроями за відсутності відмов.

5.2.3. З'єднання пристроїв і організація ліній інтерфейсу

З'єднання між собою декількох пристроїв виконується за допомогою індивідуальних ліній для кожної пари пристроїв або загального для всіх пристроїв середовища інтерфейсу на основі розділення часу.

У другому випадку для попередження конфліктних ситуацій, які виникають під час спроб кількох пристроїв одночасно використовувати загальне середовище, виділяють спеціальну схему управління інтерфейсом, яку називають арбітром.

У загальному випадку можуть бути реалізовані такі види обміну: передача від одного пристрою тільки одному іншому, від одного пристрою всім іншим (трансляційний обмін); від одного пристрою кільком довільно призначеним пристроям (груповий обмін).

Апаратні інтерфейси СВВ звичайно реалізують тільки перший вид обміну – між двома пристроями.

Організація інтерфейсу повинна надавати можливість пристрою:

- займати загальне середовище інтерфейсу на час передачі повідомлення; процес надання середовища інтерфейсу одному пристрою називається арбітражем і виконується схемами арбітра;
- звертатись до іншого пристрою за його адресою; цей процес називається адресацією;
- ідентифікувати пристрій, який ініціює обмін; цей процес нерозривно пов'язаний з процедурою арбітражу і його основою є послідовне опитування пристроїв.

Організація адресації і опитування, а також структура схеми управління інтерфейсом у значній мірі визначається способом з'єднання пристроїв. За цією ознакою розрізняють радіальний, магістральний, ланцюговий і комбінований інтерфейси.

Радіальний інтерфейс. Центральний пристрій (*Пц*) з'єднаний з підключеними пристроями $П_1$, …, $П_n$ за допомогою індивідуальних ліній, які монопольно належать кожному з них (рис. 5.5).

Управління інтерфейсом повністю зосереджене в пристрої *Пц*. За необхідності передати або одержати квант інформації від *Пі* за ініціативою центрального пристрою (*Пц*) на регістр *РгА* заноситься адреса пристрою *Пі* і відповідно з ним перемикач *К* з'єднує лінії *Лц* з лініями *Лі*. При цьому пристрої *Пц* і *Пі* з'єднуються між собою, а всі інші пристрої відключаються і в обміні участі не беруть.

Якщо ініціатива обміну виходить від периферійного пристрою *Пі*, то він передає сигнал по своїй лінії запиту (на рис. 5.5 показані штрихами), який надходить в *і*-й розряд регістра запиту *РгЗ*. Як тільки *Пц* звільняється від попереднього обміну, його пристрій управління інтерфейсом ПУ послідовно

опитує розряди регістра *РгЗ* і за допомогою перемикача *К* з'єднує лінії *Лц* з відповідними лініями *Лі* пристрою *Пі*.

Порядок опитування розрядів *РгЗ* визначає пріоритет обслуговування пристроїв *Пі*.

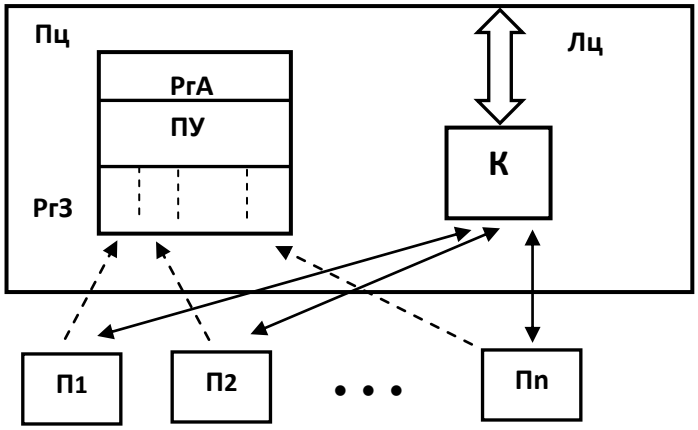


Рис. 5.5 Структура радіального інтерфейсу

Відмінними особливостями радіального способу підключення є:

- сконцентроване в центральному пристрої управління інтерфейсом, яке призначене для узгодження моментів прийому і передачі повідомлення;
- наявність індивідуальних інформаційних ліній, які потребують значних затрат на приймально-передавальну апаратуру і кабелів зв'язку;
- використання мінімального числа ліній управління;
- можливість порівняно просто пристосувати ПП до вимог інтерфейсу, а також проводити фізичне підключення і відключення пристроїв без порушення безперервної роботи інших.

Цей спосіб характерний для інтерфейсів нижніх рангів, особливо під час послідовного способу передачі інформації. Йому віддають перевагу через необхідність підключення до ЕОМ достатньо простих ПП, наприклад, пристроїв технологічної автоматики і контрольно-вимірювальної апаратури.

Магістральний інтерфейс. Центральний пристрій *Пц* з'єднаний з підключеними пристроями $П_1$, …, $П_n$ за допомогою єдиної магістралі, яка використовується ними на основі розподілу часу (рис. 5.6).

Сигнал на будь-якій лінії магістралі фізично доступний кожному пристрою, тому для організації обміну між пристроєм *Пц* і одним з підлеглих пристроїв необхідно логічно відключити всі інші.

Всім пристроям *Пі*, що підключені до магістралі, привласнені адреси (номери), які фіксуються у вигляді власної адреси пристрою на спеціальних регістрах, що розташовані у всіх *Пі*. Адреси пристроїв однієї магістралі не повторюються; запис адреси в регістр пристрою *Пі* виконується вручну під час підключення його до магістралі.

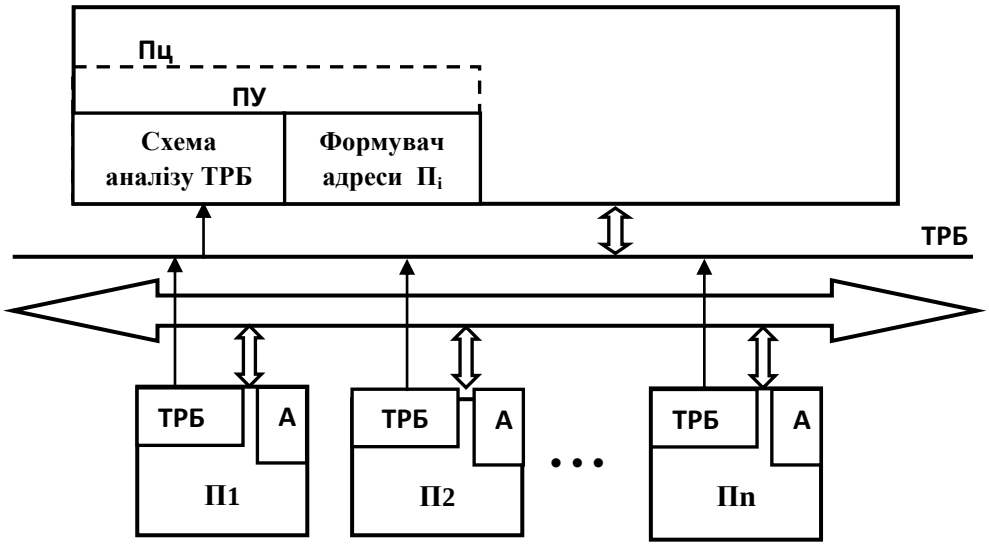


Рис. 5.6. Структура магістрального інтерфейсу

Припустимо, що обмін виконується з ініціативи пристрою *Пц*. Тоді він проводить цикл адресації, який полягає в передачі адреси пристрою, що запитується по магістралі. Адреса надходить до всіх пристроїв *Пі*, де проводиться порівняння переданої адреси з власною адресою. Якщо власна і адреса, що запитується, збігаються, то пристрій *Пі* виявляє сигнал готовності до прийому інформації від *Пц*.

Якщо обмін в інтерфейсі проводиться з ініціативи підлеглого пристрою *Пі*, то спочатку виключається можливість використання магістралі будь-яким іншим пристроєм. З цією метою в магістралі передбачають спеціальну лінію запитів (лінія *ТРБ*), на яку пристрій *Пі* незалежно від інших може виставляти сигнал запиту (або вимоги *ТРБ*). Сигнал запиту означає для *Пц*, що на магістралі є один або декілька пристроїв ***Пі***, що запитують обмін. Виявивши сигнал запиту (цю функцію виконує схема аналізу *ТРБ*), пристрій *Пц* повинен дати дозвіл на заняття магістралі тільки одному із пристроїв *Пі*, що запитують для виконання передачі даних. Для цього проводиться опитування пристроїв *Пі*, тобто пристрій *Пц* послідовно виконує адресацію всіх *Пі* до тих пір, доки не отримає підтвердження запиту. Пристрій *Пц*, отримавши підтвердження від *Пі*, припиняє дальше формування адрес, тобто призупиняє опитування, а пристрій *Пі*, який у процесі опитування розпізнав свою адресу і підтвердив збігання адрес, логічно підключається до магістралі для передачі даних.

У разі магістрального способу підключення управління інтерфейсом розподілене між центральним пристроєм *Пц*, який містить схему аналізу запитів і засобів формування послідовностей адрес, і підлеглими $П_1$ …, $П_n$ пристроями, які містять регістр власної адреси, схему збігання адрес і схему запиту обміну.

За такого способу організації інтерфейсу зменшується об'єм приймально-передавальної апаратури і кабельних з'єднань, але ускладнюється схема управління в *Пі*. Сигнали на лініях магістралі доступні одночасно всім пристроям, тому передача адрес і даних не потребує значних витрат часу, однак процедура опитування дуже довга через послідовне перебирання адрес *Пі*. Тому через це в реальні інтерфейси, що побудовані за магістральним способом з паралельними колективними лініями, додають елементи радіального або ланцюгового підключення.

Ланцюговий інтерфейс. У випадку ланцюгового інтерфейсу підлеглі пристрої $П_1$, …, $П_n$ підключаються до центрального послідовно, утворюючи ланцюг (рис. 5.7).

У ланцюговому інтерфейсі всім пристроям $П_1$, …, $П_n$ привласнюються адреси, що не повторюються. Тоді, якщо обмін ініціюється пристроєм *Пц*, адреса пристрою (*Пі*), що запитується, передається на лінії $Л_1$ і потрапляє в пристрій $П_1$. Адреса, що запитується в пристрої $П_1$, порівнюється з власною адресою $П_1$. Якщо адреси не збіглися, то комутатор *К* з'єднує лінії $Л_1$ з лініями $Л_2$. Таким чином, адреса пристрою, що запитується, потрапляє в $П_2$ і процедура повторюється.

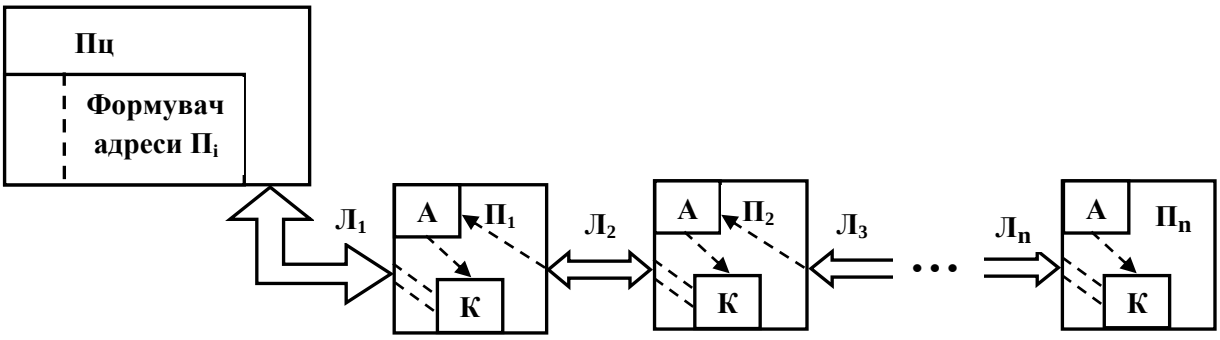


Рис. 5.7. Структура ланцюгового інтерфейсу

Якщо значення адрес збіглися, то пристрій, який упізнав свою адресу, логічно підключається до *Пц,* процедура адресації виконується послідовно.

Нехай обмін ініціюється одним з пристроїв $П_1$, …, $П_n$, наприклад $П_2$. При цьому пристрій відключає за допомогою комутатора *К* всі пристрої більш низького пріоритету ($П_3$, …, $П_n$), тобто розмикає лінії $Л_3$. Потім пристрій $П_2$ передає свою адресу по лінії $Л_2$. Ця адреса або передається пристроєм $П_1$ на лінії $Л_1$, якщо $П_1$ не веде обміну, для чого комутатор *К* в $П_1$ підключає лінії $Л_2$ до ліній $Л_1$, або блокується, якщо пристрій $П_1$ веде обмін з *Пц*. Процедура опитування не потребує послідовного перебору адрес $П_1$, …, $П_n$, що значно її прискорює.

Однак в описаному вигляді ланцюгове підключення пристроїв не використовується. Це пояснюється значними затратами часу на процедуру

адресації через її послідовний характер, значними витратами на комутуючу апаратуру і неможливістю фізичного відключення пристроїв без порушення роботи інших.

Комбіновані інтерфейси. В комбінованих інтерфейсах використовується магістральний принцип паралельної передачі інформацій, а для прискорення ідентифікації використовуються управляючі лінії, які з'єднують пристрої за радіальним (магістрально-радіальний інтерфейс) або ланцюговим (магістрально-ланцюговий інтерфейс) принципом.

На рис. 5.8 приведена *структура магістрально-радіального інтерфейсу*.

Всі види інформації передаються по паралельній магістралі *М*. Заи необхідності зв'язатися з будь-яким пристроєм *Пі* центральний пристрій *Пц* передає йому сигнал по індивідуальній лінії управління (дозвіл роботи). Цей сигнал служить для підключення пристроїв до магістралі *М* з допомогою комутатора *К*, всі інші пристрої від магістралі відключені, але мають можливість передачі сигналів запиту по своїх індивідуальних лініях управління в блок управління магістраллю (арбітр), що розташований в *Пц*.

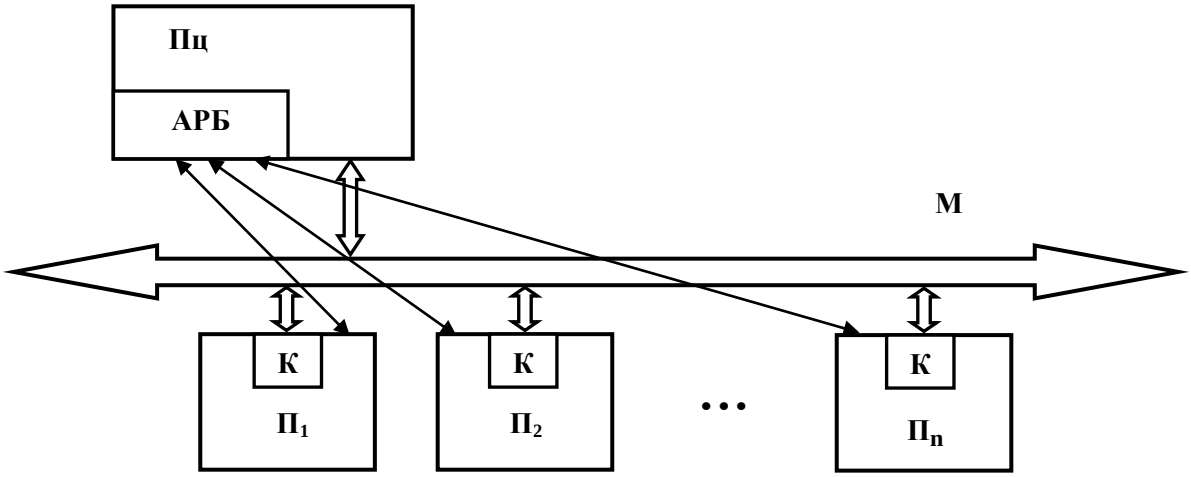


Рис. 5.8. Структура магістрально – радіального інтерфейсу

Таким чином, кожний з пристроїв $П_1$, …, $П_n$ з'єднаний з *Пц* двома індивідуальними лініями: лінією запиту і лінією дозволу. Пристрій *Пц* аналізує запити, що надходять по системі індивідуальних ліній в регістр запитів, і залежно від прийнятої системи пріоритетів видає сигнал на одну з ліній дозволу роботи, тим самим забезпечується зв'язок по магістралі *М* центрального пристрою *Пц* з одним із пристроїв $П_1$, …, $П_n$.

Магістрально – ланцюгова структура є найбільш розповсюдженою в апаратурних інтерфейсах СВВ. Всі види інформації передаються по загальній магістралі; адресація виконується так само, як і в магістральному інтерфейсі, але для прискорення передбачена лінія управління, яка з'єднує пристрої $П_1$, …, $П_n$ за ланцюговим принципом (рис. 5.9).

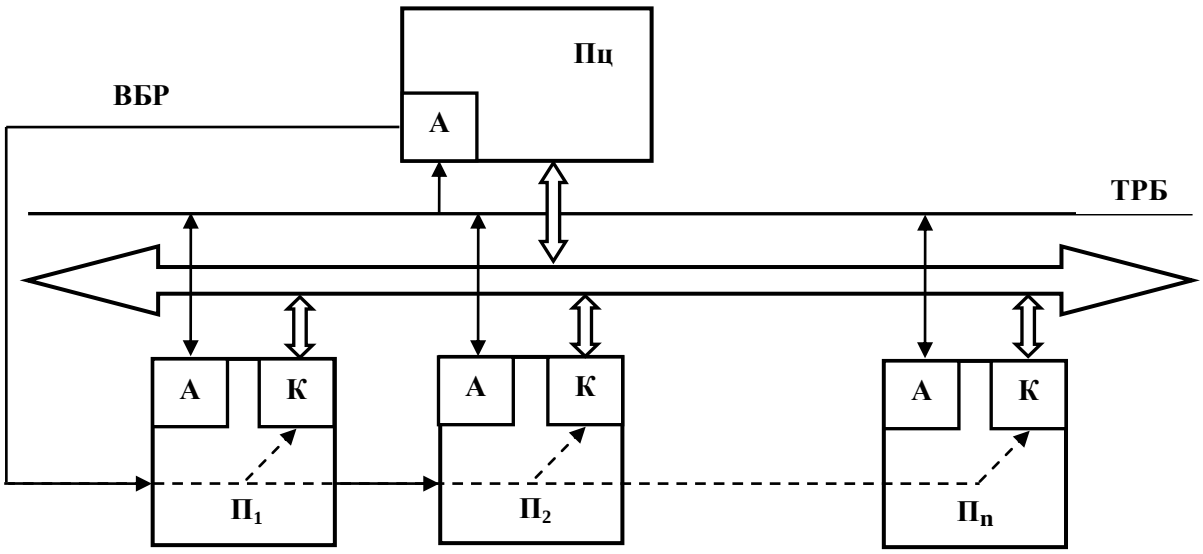


Рис. 5.9. Структура магістрально – ланцюгового інтерфейсу

Магістрально-ланцюгова структура дозволяє будувати інтерфейси, в яких можливий обмін між фіксованим і довільно вибраним пристроєм або між двома довільними пристроями.

Пристрій, що запитує обмін, називається ведучим (або задатчиком ЗДТ), а другий пристрій, який бере участь в обміні, – веденим (або виконавцем). Розв'язанням конфліктів керує арбітр (АРБ). Схема арбітра може бути зосередженою і розподіленою. В першому випадку ланцюгова лінія інтерфейсу служить для передачі сигналу дозволу (вибірки ВБР) від арбітра всім пристроям, які можуть ініціювати обмін. Для узгодження роботи арбітра і пристроїв передбачаються лінії запиту (ТРБ) і вказівки зайнятості магістралі (ЗАН).

Якщо ініціюється обмін з боку пристроїв $П_1$, …, $П_n$, то кожний з них може виставляти сигнал запиту на лінію ТРБ. Отримавши цей сигнал, пристрій $Пц$ з метою селекції пристрою, що запитує, починає процедуру опитування, тобто видає сигнал на лінію ВБР. Сигнал ВБР надходить на пристрій $П_1$. У випадку, якщо обмін ініційований пристроєм $П_1$, лінії магістралі з допомогою комутатора $К$ підключаються до $П_1$, пристрій формує сигнал ЗАН, а сигнал ВБР на наступний пристрій $П_2$ не передає.

Якщо сигнал ТРБ був сформований будь-яким іншим пристроєм, то пристрій $П_1$ передає сигнал ВБР по ланцюговій лінії на пристрій $П_2$, де проводиться такий же аналіз, і т.д.

Для своєї ідентифікації пристрій $П_i$ на початку повідомлення передає власну адресу.

5.3. ОРГАНІЗАЦІЯ ШИН КОМП'ЮТЕРА

Сукупність трактів, об'єднуючих між собою основні пристрої ОМ (центральний процесор, пам'ять і модулі вводу/виводу), утворює структуру взаємозв'язків обчислювальної машини. Структура взаємозв'язків повинна забезпечувати обмін інформацією між:

- центральним процесором і пам'яттю;
- центральним процесором і модулями вводу/виводу;
- пам'яттю і модулями вводу/виводу.

Інформаційні потоки, що характерні для основних пристроїв ОМ, показані на рис. 5.10.

Взаємозв'язок частин ОМ і її «спілкування» із зовнішнім середовищем забезпечуються системою шин. Більшість машин містять декілька різних шин, кожна з яких оптимізована під певний вид комунікацій. Частина шин прихована всередині інтегральних мікросхем або доступна тільки в межах друкарської плати.

Деякі шини мають доступні ззовні токи, з тим щоб до них легко можна було підключити додаткові пристрої, причому більшість таких шин не просто доступні, але і відповідають певним стандартам, що дозволяє під'єднувати до шини пристрої різних виробників.

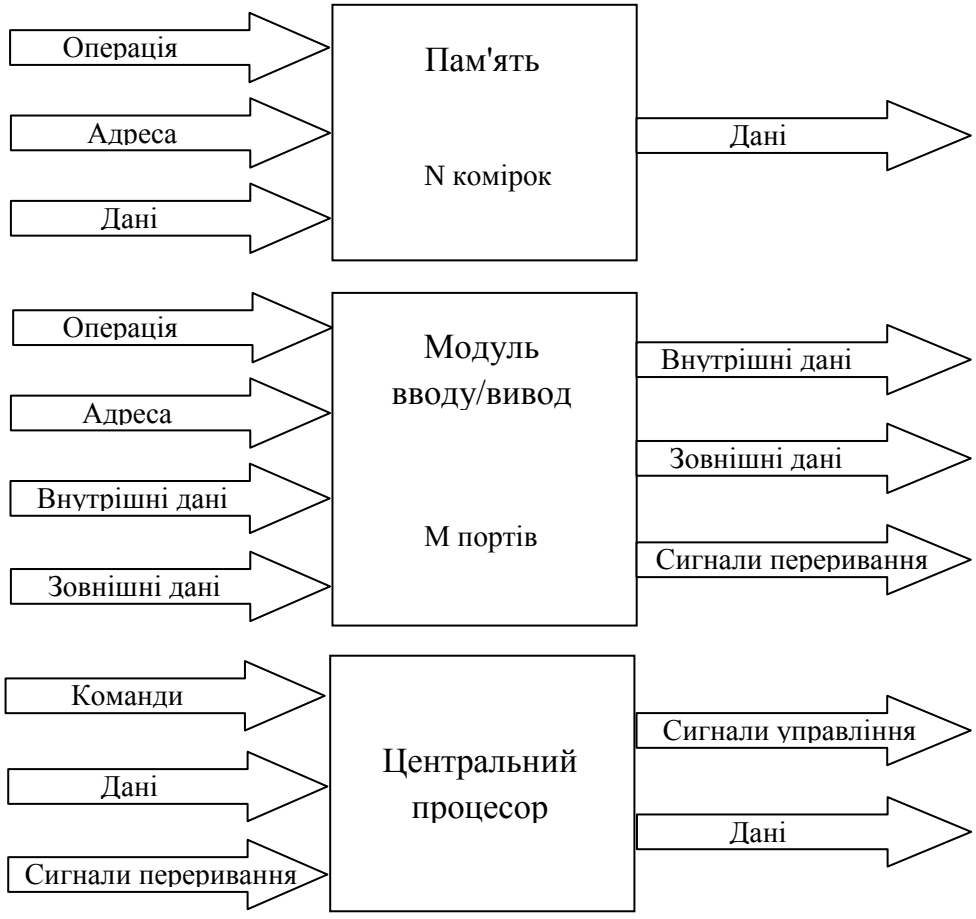


Рис. 5.10. Інформаційні потоки в обчислювальній машині

Щоб охарактеризувати конкретну шину, потрібно описати [24]:

- сукупність сигнальних ліній;
- фізичні, механічні та електричні характеристики шини;
- використовувані сигнали арбітражу, стану, управління і синхронізації;
- правила взаємодії підключених до шини пристроїв.

Шину утворює набір комунікаційних ліній, кожна з яких здатна передавати сигнали, що представляють двійкові цифри 1 і 0. По лінії може пересилатися розгорнена в часі послідовність таких сигналів. При сумісному використанні декілька ліній можуть забезпечити одночасну (паралельну) передачу двійкових чисел. Фізично лінії шини реалізуються у вигляді окремих провідників, як смужки провідного матеріалу на монтажній платі або як алюмінієві чи мідні провідні доріжки на кристалі мікросхеми.

Операції на шині називають транзакціями. Основні види транзакцій − транзакції читання і транзакції запису. Якщо в обмені бере участь пристрій вводу/виводу, можна говорити про транзакції вводу і виводу, по суті еквівалентних транзакціям читання і запису відповідно. Шинна транзакція включає дві частини: пересилання адреси і прийом (або пересилання) даних.

Коли два пристрої обмінюються інформацією по шині, один з них повинен ініціювати обмін і управляти ним. Такого роду пристрої називають ведучими (bus master). У комп'ютерній термінології «ведучий» – це будь-який пристрій, здатний узяти на себе володіння шиною і управляти пересилкою даних. Ведучий не обов'язково використовує дані сам. Він, наприклад, може захопити управління шиною на користь іншого пристрою. Пристрої, що не володіють можливостями ініціювання транзакції, носять назву ведених (bus slave). В принципі до шини може бути підключено декілька потенційних ведучих, але у будь-який момент часу активним може бути тільки один з них: якщо декілька пристроїв передають інформацію одночасно, їх сигнали перекриваються і спотворюються. Для запобігання одночасної активності декількох ведучих в будь-якій шині передбачається процедура допуску до управління шиною тільки одного з претендентів (арбітраж). У той же час деякі шини допускають широкомовний режим запису, коли інформація одного ведучого передається відразу декільком веденим (тут арбітраж не потрібний). Сигнал, направлений одним пристроєм, доступний решті всіх пристроїв, підключених до шини.

Англійський еквівалент терміну «шина» – «bus».

5.3.1. Типи і призначення шин комп'ютера

Важливим критерієм, що визначає характеристики шини, може служити її цільове призначення. По цьому критерію можна виділити:

- шини «процесор-пам'ять»;
- шини вводу/виводу;
- системні шини.

Взаємозв'язок шин і основних пристроїв у типовому комп'ютері на основі процесорів Pentium III показаний на рис. 5.11.

Шина «процесор-пам'ять» забезпечує безпосередній зв'язок між центральним процесором (ЦП) обчислювальної машини і основною пам'яттю

(ОП). У сучасних мікропроцесорах таку шину часто називають *шиною переднього плану* і позначають абревіатурою FSB (Front-Side Bus). Інтенсивний трафік між процесором і пам'яттю вимагає, щоб смуга пропускання шини, тобто кількість інформації, що проходить по шині в одиницю часу, була найбільшою. Роль цієї шини іноді виконує системна шина (див. нижче), проте в плані ефективності значно вигідніше, якщо обмін між ЦП і ОП ведеться по окремій шині. До даного виду можна віднести також шину, що пов'язує процесор з кеш-пам'яттю другого рівня, відому як *шина заднього плану,* - BSB (Back-Side Bus). BSB дозволяє вести обмін з більшою швидкістю, чим FSB, і повністю реалізувати можливості швидкіснішої кеш-пам'яті.

Оскільки у фон-нейманівських машинах саме обмін між процесором і пам'яттю багато в чому визначає швидкодію ОМ, розробники приділяють зв'язку ЦП з пам'яттю особливу увагу. Для забезпечення максимальної пропускної спроможності шини «процесор-пам'ять» завжди проектуються з урахуванням особливостей організації системи пам'яті, а довжина шини – по можливості мінімальною.

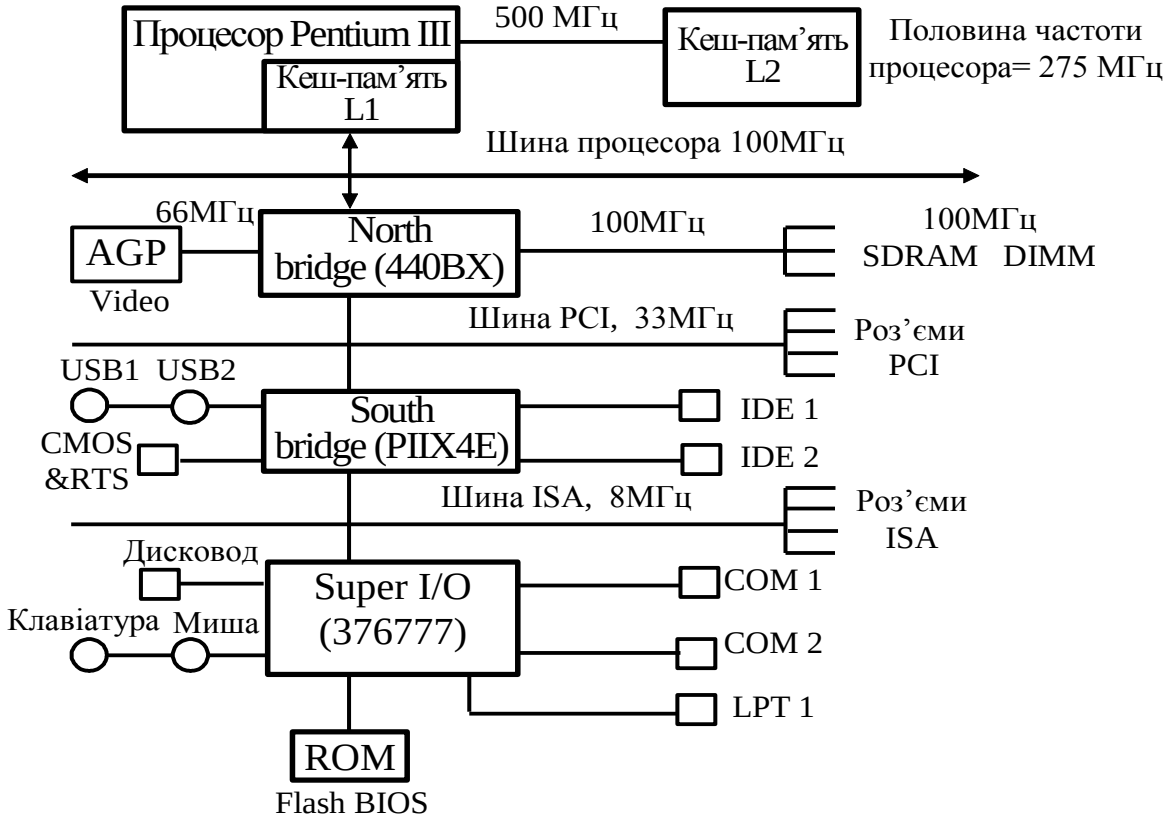


Рис. 5.11. Підключення шин у комп'ютерах з процесором Pentium III

Шина вводу/виводу служить для з'єднання процесора (пам'яті) з пристроями вводу/виводу. Враховуючи різноманітність таких пристроїв, шини вводу/виводу уніфікуються і стандартизуються. Зв'язки з більшістю пристроїв вводу/виводу (але не з відеосистемами) не вимагають від шини високої пропускної спроможності. Під час проектування шин вводу/виводу враховується вартість конструктиву і з'єднувальних роз'ємів. Такі шини містять

менше ліній в порівнянні з варіантом «процесор-пам'ять», але довжина ліній може бути дуже великою. Типовими прикладами подібних шин можуть служити шини PCI і SCSI.

Шина розширення PCI була розроблена фірмою Intel для процесора Pentium. Розробники її відмовились від традиційної концепції, ввівши ще одну шину між процесором і шиною вводу/виводу. Вона не підключається безпосередньо до шини процесора, яка дуже чутлива до зовнішніх втручань. Було розроблено новий комплект мікросхем контролерів для розширення шини. Підключення пристроїв до шини PCI подано на рис. 5.12.

Ця шина доповнює традиційну конфігурацію розташування шин ще одним рівнем. При цьому звичайна шина вводу/виводу не використовується, а фізично створюється ще одна високошвидкісна системна шина з розрядністю, що дорівнює розрядності процесора.

У материнських платах тактова частота шини PCI задається як половина тактової частоти системної шини (при тактовій частоті системної шини 66 МГц шина PCI працює на 33 МГц, при 75 МГц – на частоті 37, 5 МГц). Висока пропускна спроможність шини досягається за рахунок її паралельної роботи з шиною процесора, PCI не звертається зі своїми запитами до неї. В той час, як шина PCI обмінюється інформацією з іншими пристроями, процесор працює з даними, що містяться в кеш-пам'яті.

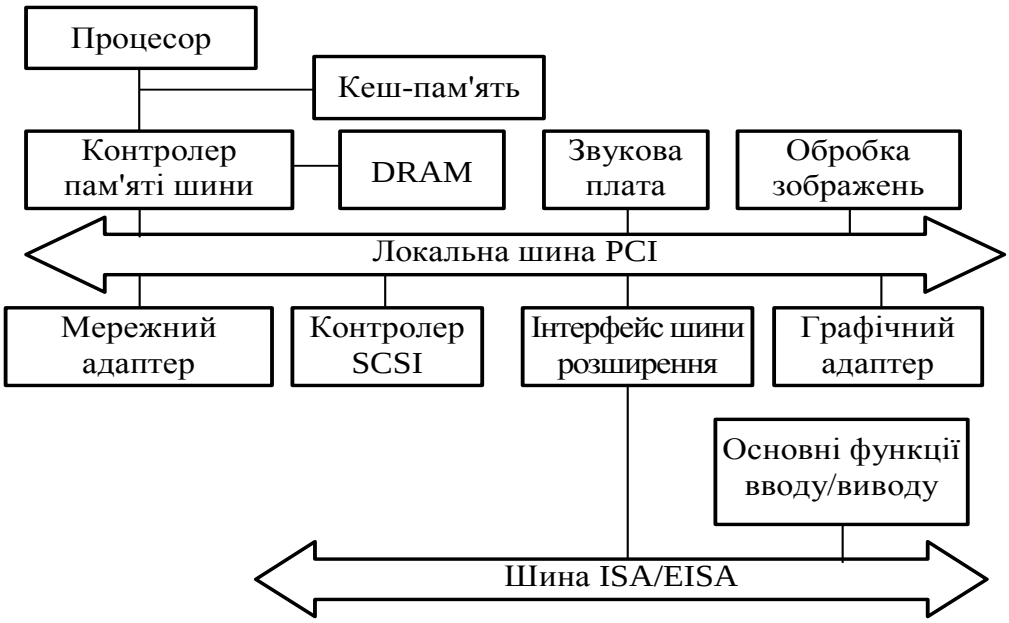


Рис. 5.12. Структурна схема підключення пристроїв до локальної шини PCI

Основним принципом, покладеним в основу локальної шини PCI, є застосування мостів (Bridges), які здійснюють зв'язок між шиною PCI та іншими шинами.

У шині PCI реалізовано принцип Bus Mastering, який відображає здатність зовнішнього пристрою під час передачі даних керувати шиною без участі

процесора. Пристрій, що підтримує Bus Mastering, захоплює шину і стає головним. За такого підходу процесор під час передачі даних звільняється для виконання інших задач. Щодо пристроїв IDE (стандарт для підключення жорстких дисків до шини ISA PC – сумісних комп'ютерів) Bus Mastering вимагає наявності відповідних схем на материнській платі, які дають змогу здійснювати передачу даних з жорсткого диска поза процесором. Це важливо під час використання багатозадачних операційних систем (Windows 95, 98, 2000, XP, NT та ін.).

Для підключення адаптерів шини PCI використовуються спеціальні роз'єми. Їх можна розпізнати по тому, що вони встановлюються окремо від звичайних роз'ємів шин ISA, MCA чи EISA. Плати PCI можуть бути такі самі за розміром, як і плати для звичайної шини вводу/виводу [22]. Шина PCI стала стандартом серед шин вводу/виводу.

Шина SCSI. Абревіатура *SCSI* розшифровується, як Small Computer System Interface; це універсальний інтерфейс, використовуваний для підключення пристроїв різного типу до персонального комп'ютера і створений на основі стандарту SASI (Shugart Associates System Interface). Інтерфейс SCSI головним чином призначений для підключення високошвидкісних жорстких дисків до високопродуктивних ПК, таких, як робочі станції і мережні сервери. SCSI є не тільки дисковим, але і системним інтерфейсом, тобто дозволяє підключати пристрої самих різних типів, зокрема принтери і сканери. Шина підтримує в цілому від 7 до 15 пристроїв. Існують також багатоканальні адаптери, які забезпечують підтримку від 7 до 15 пристроїв на кожному каналі [13].

Один з пристроїв – основний (host) адаптер, виконує роль сполучної ланки між шиною SCSI і системною шиною персонального комп'ютера. Шина SCSI взаємодіє не з самими пристроями (наприклад, з жорсткими дисками), а з вбудованими в них контролерами.

Як уже згадувалося, шина SCSI може забезпечити роботу підключених до неї модулів, кожному з яких привласнюється ідентифікаційний номер – SCSI ID. Один з модулів є платою адаптера, встановленою в комп'ютері; останні сім – периферійними пристроями. До одного і того ж основного адаптера можна підк-лючати жорсткі диски, накопичувачі на магнітній стрічці, CD ROM, сканери та інші пристрої (не більше 7 або 15). Оскільки в більшості комп'ютерів можна встановлювати до чотирьох основних адаптерів, а до кожної шини SCSI можна підключати до 15 периферійних пристроїв, то загальна кількість пристроїв може досягати 60! Більше того, існують також двоканальні адаптери, що дозволяють подвоїти це число.

Будучи "швидким" інтерфейсом, SCSI добре підходить для високопродуктивних робочих станцій, серверів або інших систем, яким життєво необхідний ефективний інтерфейс для пристроїв зберігання даних. Версія інтерфейсу Ultra4 (Ultra320) SCSI підтримує швидкість передачі даних до 320 Мбайт/с. Швидший

інтерфейс Ultra5 (Ultra640), дозволяє передавати дані із швидкістю 640 Мбайт/с. Порівняйте це з показниками 133 Мбайт/с (ATA 6) і 150 Мбайт/с нового інтерфейсу Serial ATA.

Інтерфейс SCSI прийнятий як стандарт і використовується практично у всіх високорівневих PC – сумісних комп'ютерах. Основний адаптер SCSI або встановлюється в один з роз'ємів, або вмонтовується на системній платі. Така конструкція на перший погляд нагадує інтерфейс IDE, оскільки диск SCSI підключається до системної плати за допомогою одного єдиного кабелю. Істотна різниця полягає в тому, що до SCSI можна підключити до семи пристроїв (причому не обов'язково жорстких дисків), а до IDE – два, і їх вибір дуже обмежений. Крім того, кількість пристроїв різного типу (крім жорстких дисків), які підтримуються інтерфейсом SCSI, значно більше. До пристроїв ATA зазвичай відносяться: жорсткий диск, накопичувач CD ROM ATA типу, накопичувач на магнітній стрічці, дисковод SuperDisk моделей LS 120 або LS 240, Zip дисковод і тому подібне Використання SCSI спрощує модернізацію систем, забезпечуючи підключення і нормальне функціонування практично будь-якого накопичувача SCSI, створеного стороннім виробником.

Прискорений графічний порт (AGP) призначений для підвищення ефективності роботи з відео- і тривимірною графікою. Він подібний до PCI, але має деякі доповнення і розширення. AGP не залежить від шини PCI ні логічно, ні електрично, ні фізично. Роз'єм AGP подібний до роз'єму PCI, але має контакти для додаткових сигналів і відповідно іншу розводку контактів. AGP – по суті високоефективне з'єднання, розроблене спеціально для відеоадаптера. В системі для одного адаптера допускається тільки один роз'єм AGP. Шина AGP працює на частоті 66 МГц, нею можна передавати дані зі швидкістю приблизно 533 Мбайт/с.

В табл. 5.1 наведено дані всіх режимів AGP [22].

Таблиця 5.1

Параметри режимів роботи AGP

Режим AGP	Базова частота, МГц	Робоча частота, МГц	Швидкість передачі даних, Мбайт / с
1x	66	66	266
2x	66	133	533
4x	66	266	1066

У специфікації AGP 2.0 можливий режим 4-х, за якого дані передаються зі швидкістю 1066 Мбайт/с за рахунок 4-х передач за цикл.

Місце порту AGP як каналу передачі даних між відеокартою і RAM у структурі пристроїв комп'ютера показано на рис. 5.13.

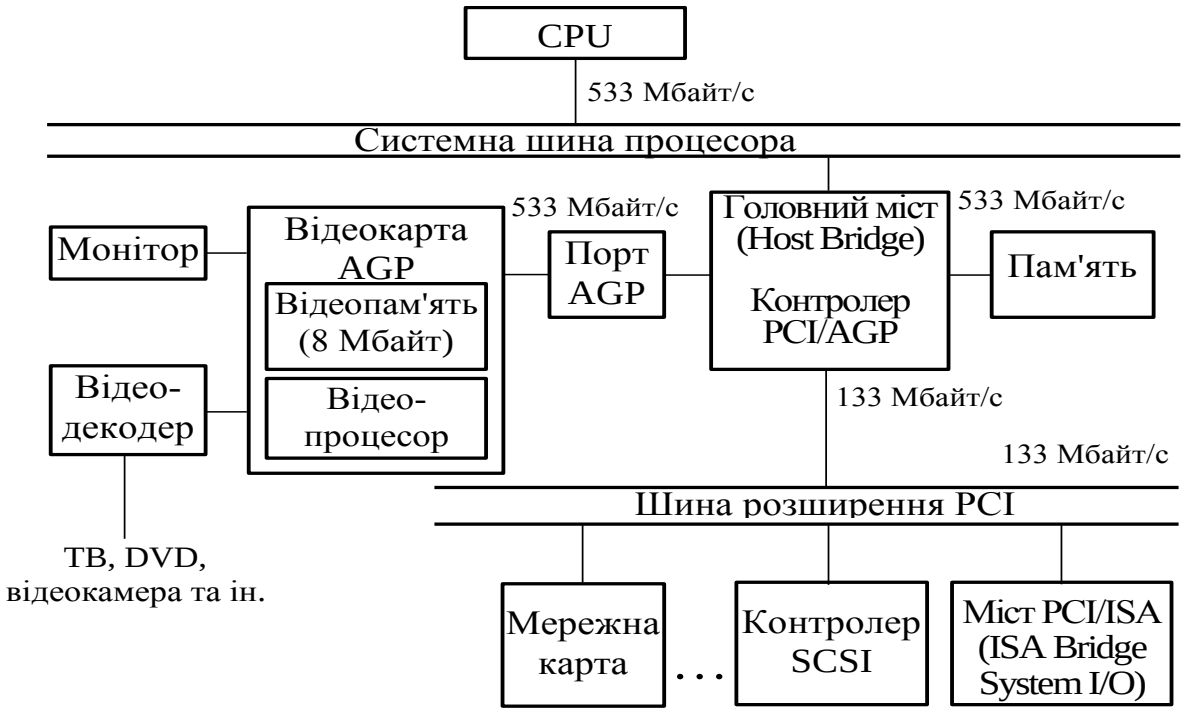


Рис. 5.13. Структура пристроїв комп'ютера з портом AGP

Під час використання відеоадаптера AGP шина PCI вивільняється для виконання традиційних функцій вводу/виводу (для контролерів IDE/ATA, SCSI чи USB, звукових плат та ін.). Крім підвищення ефективності роботи відеоадаптера порт AGP дає змогу отримати швидкий доступ безпосередньо до оперативної пам'яті.

Однією з головних особливостей стандарту AGP є здатність розподілити RAM між процесором і чипсетом відеокарти. Обробка тривимірних зображень виконується RAM центрального процесора і відеопроцесором.

Механізм доступу відеокарти до пам'яті називають *безпосереднім виконанням у пам'яті DIME* (Direct Memory Execute).

Іншими характерними особливостями шини AGP є:

а) конвеєрна передача даних, яка полягає в тому, що цією шиною спочатку передається послідовність адрес, а потім – пакет даних. На шині PCI, на відміну від AGP, адреси з даними чергуються;

б) в AGP використовується режим *адресування бічною смугою* (Sideband addressing) за допомогою спеціальних сигналів SBA (Side Band Addressing).

Системна шина. З метою зниження вартості деякі ОМ мають загальну шину для пам'яті і пристроїв вводу/виводу. Така шина часто називається системною. Системна шина служить для фізичного і логічного об'єднання всіх пристроїв ОМ. Оскільки основні пристрої машини, як правило, розміщуються на загальній монтажній платі, системну шину часто називають об'єднувальною шиною (backplane bus), хоча ці терміни не можна вважати строго еквівалентними.

Системна шина може містити декілька сотень ліній. Сукупність ліній шини можна підрозділити на три функціональні групи: шину даних, шину адреси і шину управління [23]. До останньої зазвичай відносять також лінії для подачі напруги живлення на модулі, що підключаються до системної шини.

Функціонування системної шини можна описати таким чином. Якщо один з модулів хоче передати дані в інший, він повинен виконати дві дії: отримати в своє розпорядження шину і передати по ній дані. Якщо якийсь модуль хоче отримати дані від іншого модуля, він повинен отримати доступ до шини і за допомогою відповідних ліній управління і адреси передати в інший модуль запит. Далі він повинен чекати, поки модуль, що отримав запит, пошле дані.

Фізично системна шина є сукупністю паралельних електричних провідників. Цими провідниками служать металеві смужки на друкарській платі. Шина підводиться до всіх модулів, і кожен з них під'єднується до всіх або деяких її ліній. Якщо ОМ конструктивно виконана на декількох платах, то всі лінії шини виводяться на роз'єми, які потім об'єднуються провідниками на загальному шасі.

Серед стандартизованих системних шин універсальних ОМ найбільш відомі шини Unibus, Fastbus, Futurebus, VME, NuBus, Multibus-II. Персональні комп'ютери, як правило, будуються на основі системної шини в стандартах ISA, EISA або MCA.

Шина ISA. Протягом багатьох років шина ISA була стандартом у галузі PC – комп'ютерів. Вона є однією з перших у сімействі шин, але використовується до цього часу. Це зумовлено тим, що для багатьох пристроїв, зокрема миші, клавіатури, модемів, ручних сканерів та інших, швидкодія цієї шини більш ніж достатня. Свого часу, коли частота ISA перевищила 8 МГц, здійснювались спроби відділити шину ISA від шини процесора, яка була тоді основною. Раніше вони працювали на одній частоті. Згодом, щоб шини не розділяти, було розроблено розширений варіант шини ISA з новою назвою – VESA Local Bus (чи VL–Bus). Так відбувся поворот до архітектури локальних шин.

Фірма Intel сумісно з Microsoft розробила стратегію поступової відмови від шини ISA (згідно із специфікацією PC98 і PC99). За специфікації PC99 надалі в комп'ютері повинні використовуватись тільки дві шини – PCI і AGP. 16–розрядну шину ISA інколи називають AT BUS. Її слот складається з двох частин, одна з яких повністю відповідає слоту 8 – розрядної шини ISA, а на контакти другої частини виведено лінії для додаткових адрес вводу/виводу, переривань та каналів DMA (Direct Memory Access). Тому короткі 8–розрядні карти можна встановити в 16–розрядний слот. Додатковий слот має 36 контактів.

Передача байта даних шиною ISA відбувається так: на адресній шині виставляється адреса комірки RAM чи порту пристрою вводу/виводу, в який необхідно передати байт. Потім на шину даних виставляється байт даних, і

однією з ліній шини керування передається сигнал запису WR, який є стробом. Контроль запису не здійснюється, тому тактова частота шини ISA вибрана 8,33 МГц. Це зроблено для того, щоб навіть найповільніші пристрої встигали здійснювати обмін даними чи командами шиною.

Основним недоліком шини ISA є те, що вже у разі частоти процесорів i386 та i486 дані не можуть передаватися шиною з тією самою швидкістю, з якою їх обробляє процесор. Тому в очікуванні даних він вимушений простоювати. Це стало причиною появи нових стандартів.

Шина MCA. Шина MCA була розроблена фірмою IBM в 1987 році і встановлена в комп'ютерах класу PS/2. У ній було підвищено пропускну здатність до 20 Мб/с за рахунок збільшення тактової частоти до 10 МГц і розрядності до 32 біт. Відпала необхідність вручну конфігурувати зовнішні пристрої, встановлені в слоти розширення MCA. Але шина MCA не знайшла значного поширення. Причиною цього була повна її несумісність із шиною ISA і необхідність заміни материнської плати і карт розширення.

Шина EISA є подальшим вдосконаленням шини ISA. Вона була розроблена фірмами Epson, Hewlett–Packard, NEC, Compaq і Wyse і має такі переваги:

- повна сумісність слота EISA із слотом ISA, що дає можливість встановлювати карти ISA в слоті EISA, а це, в свою чергу, відкидає необхідність замінювати всі карти розширення;
- шина EISA є 32–розрядною, що дає можливість використання відповідних карт – мережних, графічних, жорсткого диска;
- шина EISA (як і MCA) є інтелектуальною, тобто конфігурація карт розширення виконується не апаратно за допомогою DIP – перемикача і джамперів, а програмно.

У слоті EISA “перший поверх” роз'єму залишається без змін стосовно шини EISA. Для недопущення електричного контакту роз'єму карт ISA з контактами “другого поверху” слота EISA встановлюється заглушка.

Шина EISA не отримала значного поширення через високу вартість і відсутність у достатній кількості карт розширення EISA та нижчу пропускну спроможність порівняно з локальною шиною VESA.

Шина VESA була розроблена для зв'язку процесора зі швидкодіючими периферійними пристроями. Вона є розширенням шини ISA для обміну відеоданими. Обмін інформацією з процесором здійснюється під керуванням контролерів, розташованих на картах, що встановлюються в слот VESA (VLB) в обхід стандартної шини вводу/виводу. Ця шина 32–розрядна і працює на тактовій частоті процесора, яка не повинна перевищувати 40 МГц, але вона працює і на частоті процесора 50 МГц, встановленого на материнській платі. На

ній є два слоти VLB, а також відповідні контролери VLB, у тому числі і для вінчестера.

Шина VESA широко застосовувалась для процесорів i486, але на сьогодні вона повністю витіснена продуктивнішою шиною PCI.

5.3.2. Послідовний, паралельний та інші інтерфейси вводу/виводу

Основними засобами комунікації в сучасних ПК, є послідовні і паралельні порти.

Послідовні порти (вони ж комунікаційні або COM порти) спочатку використовувалися пристроями, яким була потрібна двонаправлена взаємодія з системою. Сюди відносяться модеми, миші, сканери, дігітайзери і будь-які інші пристрої, які "говорять" з ПК і отримують відповідну "відповідь". Нові послідовні порти дозволяють здійснювати високошвидкісну двонаправлену передачу даних.

Асинхронний послідовний інтерфейс – це основний тип інтерфейсу, за допомогою якого здійснюється взаємодія між комп'ютерами (рис. 5.2). Термін асинхронний означає, що під час передачі даних не використовуються ніякі синхронізуючі сигнали і окремі символи можуть передаватися з довільними інтервалами.

Кожному символу, що передається через послідовне з'єднання, повинен передувати стандартний стартовий сигнал, а завершувати його передачу повинен стоповий сигнал.

Стартовий сигнал – це нульовий біт, що називається стартовим бітом. Він повинен повідомити приймаючий пристрій про те, що наступні вісім біт є байт даних. Після символу передаються один або два стопових біта, таких, що сигналізують про закінчення передачі символу. У приймаючому пристрої символи розпізнаються по появі стартових і стопових сигналів.

Термін «послідовний» означає, що передача даних здійснюється по одиночному провіднику, а біти при цьому передаються послідовно, один за другим.

Паралельний порт використовується для підключення до комп'ютера принтера, звідси і пішла його назва – LPT (Line Printer Terminal – порт порядкового принтера). Традиційний, він же стандартний, *LPT-порт* (що називається ще *SSP-портом*) орієнтований на виведення даних, хоча з деякими обмеженнями дозволяє і вводити дані. Проте, незважаючи на таку вузьку спеціалізацію, паралельні порти почали застосовуватися як відносний швидкий інтерфейс передачі даних між пристроями.

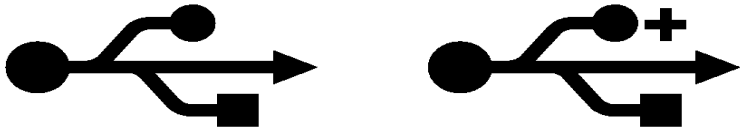
У паралельних портах для одночасної передачі байта інформації використовується вісім ліній. Цей інтерфейс відрізняється високою швидкодією, часто застосовується для підключення до комп'ютера принтера, а також для з'єднання комп'ютерів.

Істотним недоліком паралельного порту є те, що з'єднальні провідники не можуть бути дуже довгими. У разі великої довжини сполучного кабелю в нього доводиться вводити проміжні підсилювачі сигналів, оскільки інакше виникає багато перешкод.

В даний час для настільних і портативних комп'ютерів розроблено два високошвидкісні пристрої з послідовною шиною: USB і IEEE 1394, що називається також i.Link або FireWire. Ці високошвидкісні комунікаційні порти відрізняються від стандартних паралельних і послідовних портів, встановлених у більшості сучасних комп'ютерів, ширшими можливостями. Перевага нових портів полягає в тому, що їх можна використовувати як альтернативу SCSI для високошвидкісних з'єднань з периферійними пристроями, а також під'єднувати до них всі типи зовнішніх периферійних пристроїв.

Периферійна шина USB призначена для периферійних пристроїв поза корпусом PC. Швидкість обміну інформацією шиною USB – 12 Мбіт/с. Для підключення до комп'ютера всі периферійні пристрої повинні бути обладнані роз'ємами USB і підключатися до PC через окремий блок – USB-хаб чи концентратор, який знаходиться на материнській платі.

До комп'ютерів з шиною USB підключають клавіатуру, мишу, джойстик, принтер та інші периферійні пристрої, не вимикаючи живлення. У разі підключення кожного з них автоматично здійснюється його конфігурування. За його допомогою до комп'ютера можна підключити до 127 периферійних пристроїв. Кабелі, роз'єми, концентратори та периферійні пристрої, що підтримують USB, можна визначити по знаку, показаному на рис. 5.14. Зверніть увагу на символ плюс, доданий до другого знака – він означає стандарт USB 2.0 (високошвидкісний стандарт).



Підтримка USB 1.x Підтримка USB 2.0 і 1.x

Рис. 5.14. Логотип пристроїв USB

Основним ініціатором розробки стандарту USB виступила фірма Intel. Починаючи з набору мікросхем системної логіки Triton II (82430HX), в якому стандарт USB був утілений в мікросхемі PIIX3 South Bridge, компанія Intel підтримує цей стандарт у всіх своїх наборах мікросхем системної логіки.

Спільно з Intel над створенням універсальної послідовної шини працювали ще сім компаній, серед яких Compaq, Digital, IBM, Microsoft, NEC і Northern Telecom. Ними був створений USB Implement Forum (USB IF), метою якого є розвиток, підтримка і розповсюдження архітектури USB.

Перша версія USB анонсована в січні 1996 року, а версія 1.1 - у вересні 1998 року. У цій специфікації детальніше описані концентратори та інші пристрої.

Більшість USB пристроїв повинні бути сумісні із специфікацією 1.1, навіть якщо вони випущені до її офіційної публікації. У специфікації USB 2.0, що з'явилася відносно недавно, швидкість передачі даних в 40 разів вища, ніж в оригінальній USB 1.0; крім того, забезпечується повна зворотна сумісність пристроїв.

Шина Fire Wire (IEEE 1394) це стандарт високошвидкісної локальної послідовної шини, розроблений фірмами Apple та Texas Instruments, який є частиною стандарту SCSI-3. Шина IEEE 1394 призначена для обміну цифровою інформацією між шиною розширення PCI та іншими пристроями, зокрема жорсткими дисками, пристроями обробки аудіо- і відеоінформації та ін. Вона також ідеально підходить для роботи мультимедійних додатків у реальному часі.

Шина IEEE 1394 передає дані зі швидкістю 12,5; 25; 50; 100 та 200 Мбайт/с, а під час роботи з окремими файлами – до 1 Гбіт/с. Пакетний режим передачі інформації забезпечує її високу швидкість. Для передачі інформації використовується простий 6-провідний кабель. Дві різні пари ліній кабелю призначені для передачі тактових імпульсів та інформації, а дві лінії є лініями живлення. Шина повністю підтримує технологію Plug and Play, в тому числі забезпечує можливість "гарячого" підключення – встановлення і вилучення компонентів без вимкнення живлення комп'ютера.

Шина побудована за розгалуженою топологією і дає можливість використовувати до 63 вузлів у ланцюжку. До кожного вузла, в свою чергу, можна під'єднати до 16 пристроїв. Додаткове підключення 1023 шинних перемичок дає змогу під'єднати понад 64 000 вузлів. Для передачі сигналів без перекручень довжина стандартного кабелю, який з'єднує два вузли, не повинна перевищувати 4,5м. До шини IEEE 1394 можна під'єднувати практично всі пристрої, здатні працювати з SCSI. До них належать накопичувачі на жорстких і оптичних дисках, CD – ROM, DVD – диски, цифрові відеокамери, пристрої запису на магнітну стрічку та інші високошвидкісні периферійні пристрої. Адаптери IEEE 1394 випускаються для шини PCI.

Починаючи з Windows 98, у наступних версіях Windows додаються драйвери для портів IEEE 1394. Фірма Intel згідно із специфікацією PC98 і PC99 планує повністю замінити шину ISA шиною USB для підключення низькошвидкісних периферійних пристроїв вводу/виводу і шиною IEEE 1394 для підключення пристроїв зберігання інформації CD–ROM, HDD та інших, а також введення відеоданих.

5.4. АРХІТЕКТУРА СИСТЕМНОЇ ПЛАТИ

Найважливішим вузлом комп'ютера є системна плата (system board), яку іноді називають материнською (motherboard), основною або головною платою.

Спільно з процесором і оперативною пам'яттю вона утворює платформу, що визначає основні функціональні можливості комп'ютера і його

продуктивність [5]. Ключовий елемент системної плати – набір мікросхем системної логіки (НМСЛ), який забезпечує взаємодію інших компонентів і функціонування базових інтерфейсів. Крім НМСЛ системна плата містить базову систему вводу/виводу (*BIOS*), ланцюги електричного живлення компонентів, механічні елементи для монтажу в корпус.

Основними параметрами системної плати є:

- форм-фактор;
- підтримуваний інтерфейс процесора;
- тип і максимальний об'єм підтримуваної оперативної пам'яті;
- підтримувані інтерфейси;
- тип і можливості *BIOS*.

Як додаткові параметри часто виступають вбудовані графічні, звукові, комунікаційні можливості.

Форм-фактори системних плат

Існує декілька найбільш поширених форм-факторів, що враховуються У ході розробки системних плат. Форм-фактор (form factor) є фізичним параметром плати і визначає тип корпусу, в якому вона може бути встановлена. Форм-фактори системних плат можуть бути стандартними (тобто взаємозамінними) або нестандартними. Нестандартні форм-фактори, на жаль, є перешкодою для модернізації комп'ютера, тому від їх використання краще відмовитися.

Найбільш відомі форм-фактори системних плат перераховані нижче:

- ATX;
- micro-ATX;
- flex-ATX;
- mini-ITX (різновид flex-ATX);
- NLX;
- Інші: незалежні конструкції (розробки компаній Compaq, Packard Bell, Hewlett Packard, портативні/мобільні системи і так далі).

За останні декілька років відбувся перехід від системних плат оригінального форм-фактора *Baby AT*, який використовувався в перших комп'ютерах IBM PC і XT, до плат форм-фактора *ATX* і *NLX*, використовуваним в більшості повнорозмірних настольних і вертикальних систем. Існує декілька варіантів форм-фактора *ATX*, до числа яких входять *micro-ATX* (зменшена версія форм-фактора *ATX*, використовуваного в системах малих розмірів) і *flex-ATX* (ще більш зменшений варіант, призначений для домашніх комп'ютерів нижчого цінового рівня). Існує також і новий форм-фактор *mini-ITX*, що є зменшеною версією форм-фактора *flex-ATX*, призначеного для систем мінімального розміру. Форм-фактор *NLX* призначений для корпоративних настільних систем. Сучасні форм-фактори фактично є промисловим стандартом,

що гарантує сумісність кожного типу плат. Це означає, що системна плата будь-якого типу може бути замінена іншою платою того ж типу.

Стисло розглянемо основні стандартні форм-фактори системних плат.

Форм-фактор ATX. Конструкція *ATX* була розроблена порівняно недавно. В даний час *ATX* є найбільш поширеним форм-фактором системних плат, що рекомендується для більшості нових систем [13].

На рис. 5.15 показано, як виглядає конструкція системи *ATX* в настільному виконанні із знятою верхньою кришкою або у вертикальному з видаленою бічною панеллю.

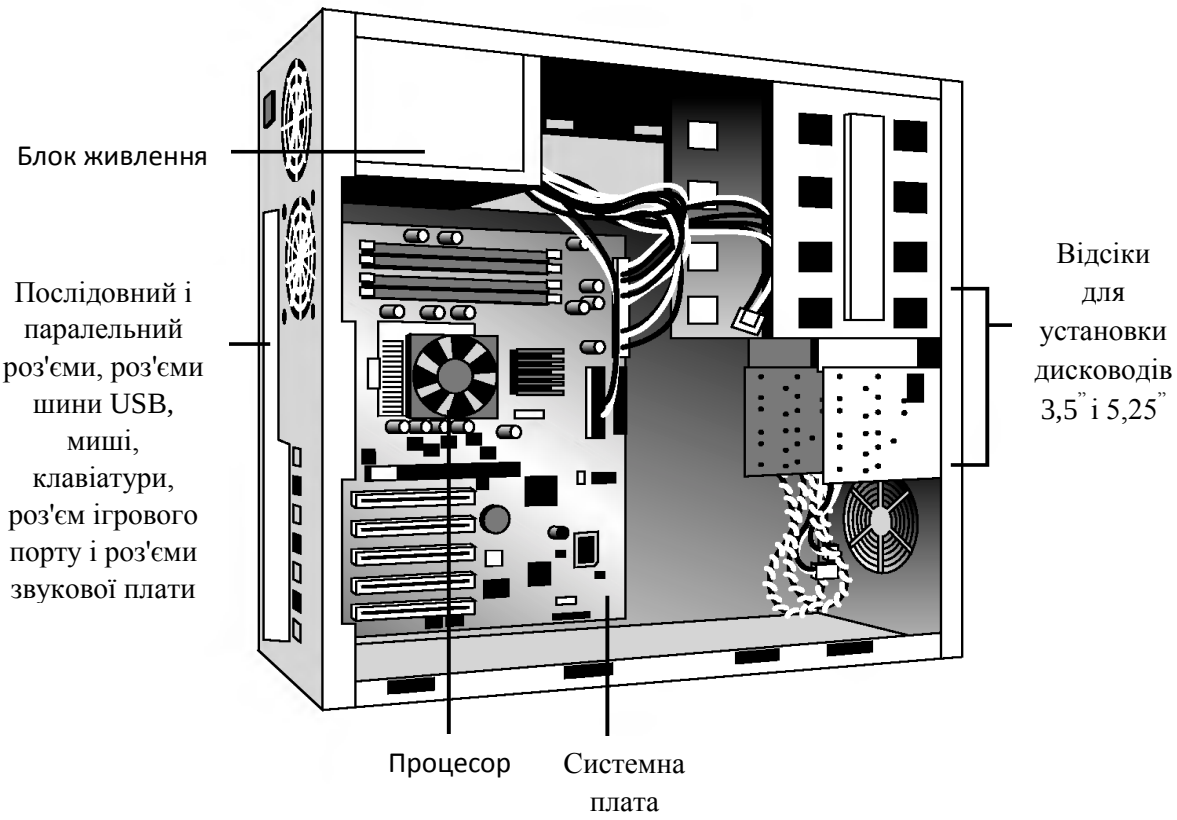


Рис. 5.15. Системна плата ATX

Зверніть увагу: системна плата практично не закривається відсіками для установки дисководів, що забезпечує вільний доступ до різних компонентів системи (процесор, модулі пам'яті, внутрішні роз'єми дисководів), не заважаючи доступу до роз'ємів шини.

У цій системній платі поєднуються якнайкращі риси раніших стандартів *Baby AT* і *LPX* і закладено багато додаткових удосконалень.

Наявність вбудованої подвійної панелі роз'ємів. На тильній стороні системної плати є область з роз'ємами вводу/виводу шириною 6,25 і заввишки 1,75 дюйма. Це дозволяє розташувати зовнішні роз'єми безпосередньо на платі і виключає необхідність використання кабелів, що з'єднують внутрішні роз'єми і задню панель корпусу.

Наявність одноключового внутрішнього роз'єму джерела живлення. Це спрощує заміну роз'ємів на джерелі живлення типу *Baby AT*. Специфікація *ATX*

містить одноключовий роз'єм джерела живлення, яке легко вставляється і який неможливо встановити неправильно. Цей роз'єм має контакти для підведення до системної плати напруги 3,3В, а це означає, що для системної плати *ATX* не потрібні вбудовані перетворювачі напруги, які часто виходять з ладу.

Переміщення процесора і модулів пам'яті. Змінені місця розташування цих пристроїв: тепер вони не заважають платам розширення, і їх легко замінити новими, не виймаючи при цьому жодного зі встановлених адаптерів. Процесор і модулі пам'яті розташовані поряд з джерелом живлення і обдуваються одним вентилятором, що дозволяє обійтися без спеціального вентилятора для процесора, який не завжди ефективний і часто схильний до поломок.

Вдаліше розташування внутрішніх роз'ємів вводу/виводу. Ці роз'єми для накопичувачів на гнучких і жорстких дисках зміщені і знаходяться не під роз'ємами розширення або самими накопичувачами, а поряд з ними. Тому можна зменшити довжину внутрішніх кабелів до накопичувачів, а для доступу до роз'ємів не потрібно прибирати одну з плат або накопичувач.

Покращуване охолоджування. Процесор і оперативна пам'ять сконструйовані і розташовані так, щоб максимально поліпшити охолоджування системи в цілому.

При цьому необхідність в окремому вентиляторі для охолоджування корпусу або процесора знижується (правда, не настільки, щоб відмовитися від нього зовсім). Одна з особливостей оригінальної специфікації *ATX* полягала в тому, що вентилятор блока живлення направляє потік повітря всередину корпусу. Зворотний потік або схема нагнітання повітря приводить до підвищення тиску в корпусі, що перешкоджає проникненню грязі і пилу.

Зниження вартості. Конструкція *ATX* не вимагає наявності гнізд кабелів до роз'ємів зовнішніх портів, що зустрічаються на системних платах *Baby AT*, додаткового вентилятора для процесора і 3,3 вольтного стабілізатора на системній платі. У цій конструкції використовується один єдиний роз'єм живлення. Крім того, можна укоротити внутрішні кабелі дискових накопичувачів. Все це істотно знижує вартість не тільки системної плати, але і всього комп'ютера, включаючи корпус і джерело живлення.

Системна плата *ATX*, по суті, є конструкцією *Baby AT*, перевернутою на бік. Роз'єми розширення паралельні коротшій стороні і не заважають гніздам процесора, пам'яті і роз'ємам вводу/виводу (рис. 5.16).

Окрім повнорозмірної схеми *ATX*, компанія Intel описала конструкцію *mini-ATX*, яка розміщується в такому ж корпусі:

- повнорозмірна плата *ATX* має розміри 305×244 мм (12×9,6 дюйма);
- плата *mini-ATX* - 284×208 мм (11,2×8,2 дюйма).

Крім того, існує два зменшені варіанти системної плати *ATX*, які носять назви *micro-ATX* і *flex-ATX*.

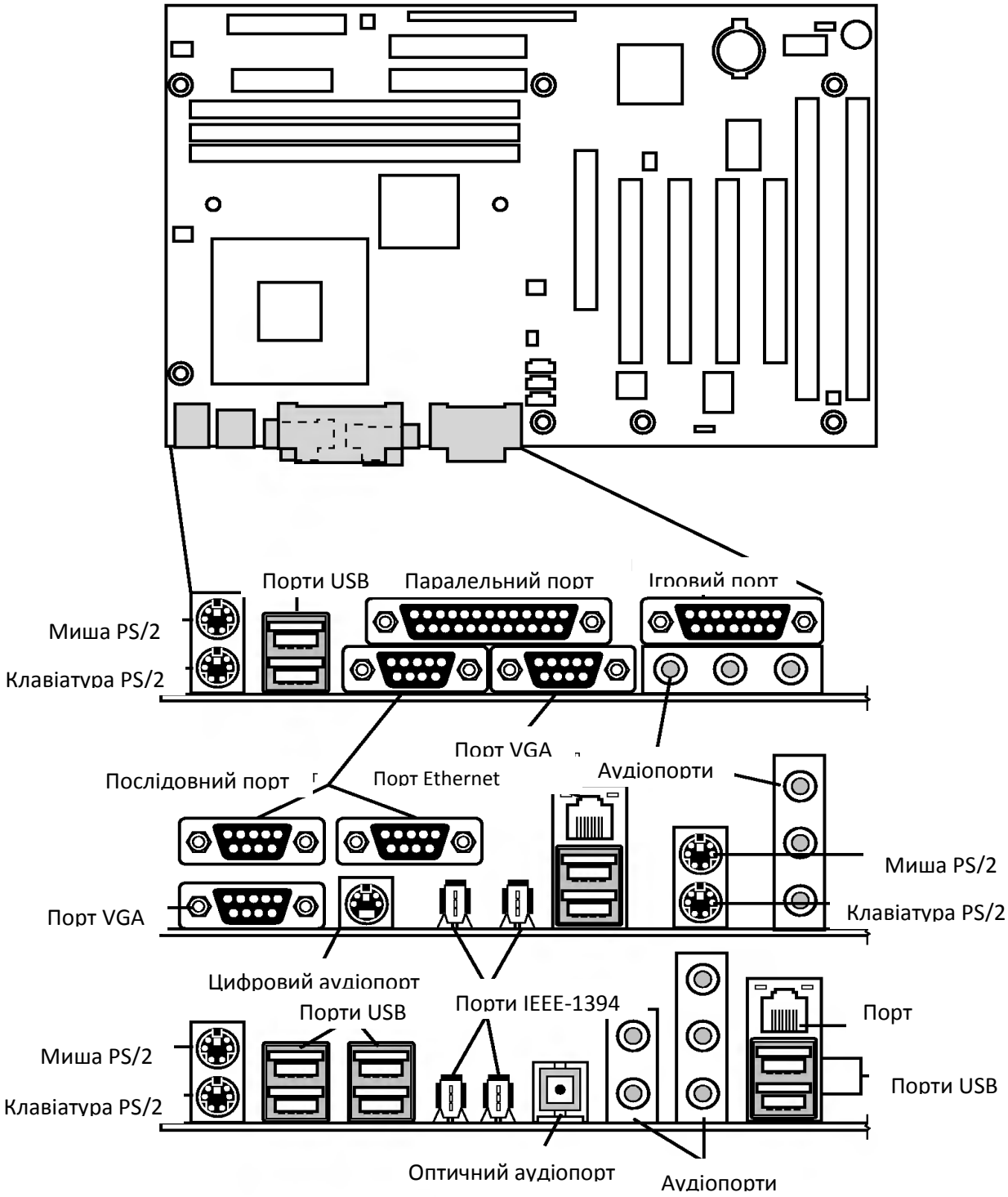


Рис. 5.16. Типове розташування роз'ємів на платі ATX
та її задній панелі

Micro-ATX. Форм-фактор системної плати *micro-ATX* був уперше представлений компанією *Intel* як варіант зменшеної плати *ATX*, призначений для невеликих і недорогих систем. Зменшення форм-фактора стандартної плати *ATX* привело до зменшення розмірів корпусу, системної плати і блока живлення і врешті-решт до зниження вартості системи в цілому. Крім того, форм-фактор *micro-ATX* сумісний з форм-фактором *ATX*, що дозволяє використовувати системну плату *micro-ATX* у повнорозмірному корпусі *ATX*.

Системні плати форм-факторів *micro-ATX* і *ATX* мають такі основні відмінності:

- зменшена ширина (244 мм (9,6 дюйма) замість 305 мм (12 дюймів) або 284 мм (11,2 дюйма);
- зменшене число роз'ємів;
- зменшений блок живлення (форм-фактора SFX/TFX).

Системні плати форм-факторів *micro-ATX* і *ATX* мають такі основні відмінності:

- зменшена ширина (244 мм (9,6 дюйма) замість 305 мм (12 дюймів) або 284 мм (11,2 дюйма);
- зменшене число роз'ємів;
- зменшений блок живлення (форм-фактора SFX/TFX).

Сумісність плат *micro-ATX* з *ATX* означає наступне:

- використання одного і того ж 20-контактного роз'єму живлення;
- стандартне розташування роз'ємів вводу/виводу;
- однакове розташування кріпильних гвинтів.

Flex-ATX. У березні 1999 року Intel опублікувала доповнення до специфікації *micro-ATX*, назване *flex-ATX*. У цьому доповненні описувалися системні плати ще меншого розміру, чим *ATX*, які дозволяють виробникам створювати невеликі і недорогі системи. Зменшений розмір плат *flex-ATX* призначений для використання в багатьох сучасних ПК, особливо тих, які відрізняються невисокою ціною, розміром і орієнтовані на користувачів, що працюють з офісними додатками. У деяких платах *flex-ATX* навіть немає слотів розширення і замість них використовуються тільки порти USB або IEEE 1394/FireWire. Форм-фактор *flex-ATX* визначає системну плату, яка є найменшою з сімейства *ATX*. Розміри цієї плати всього 229×191 мм (9,0×7,5 дюйма).

ITX і mini-ITX. Підрозділ Platform Solutions компанії VIA Technologies поставив завдання створити системну плату з мінімальними розмірами (зрозуміло, наскільки можливо), причому не придумуючи для цього нового, не сумісного з уже існуючими форм-факторами. У березні 2001 року була створена плата дещо меншої ширини, чим *flex-ATX* (21,6 см замість 22,8 см), проте тієї ж глибини. Плата, що в результаті вийшла, була на 6% менше плати *flex-ATX* і при цьому по колишньому відповідала стандартам *flex-ATX*. Нова плата отримала назву *ITX*, проте зменшення розмірів всього на 6% виявилося недостатнім для промислового виробництва, тому плати форм-фактора *ITX* так і не побачили світ.

У квітні 2002 року компанія VIA представила плату з меншими габаритами, яка характеризувалася мінімальною глибиною і шириною, допустимими в рамках стандарту *flex-ATX*. Новий форм-фактор називався

mini-ITX. По суті, всі зменшені варіанти плат стандарту *ATX* є платами *flex-ATX* з максимально зменшеними габаритами. Всі інші характеристики, будь то розмір і розташування портів вводу/виводу, розміщення монтажних отворів і типи/кількість роз'ємів блока живлення аналогічні стандарту *flex-ATX*. Проте плати більшого розміру не можна встановити в корпус *mini-ITX*. Розмір плат *mini-ITX* складає 170×170 мм (6,7"×6,7"), а значить, ці плати на 34% менше максимальних габаритів, вказаних в стандарті *flex-ATX*.

NLX. Конструкція *NLX*, представлена компанією Intel у листопаді 1996 року, є низькопрофільним форм-фактором, призначеним для заміни раніше використовуваної нестандартної конструкції *LPX*. Форм-фактор *NLX* почав використовуватися в корпоративних системах Slimline таких компаній, як Compaq, HP, Toshiba та ін.

Численні удосконалення, що відрізняють форм-фактор *NLX* від конструкції *LPX*, дозволяють повною мірою використовувати найостанніші технології в області системних плат. *NLX* - це покращувана і, що саме головне, повністю стандартизована версія незалежної конструкції *LPX*, тобто одну плату *NLX* можна замінити платою іншого постачальника, що було неможливим для плат форм-фактора *LPX*.

Застосування системних плат *LPX* обмежене фізичними розмірами сучасних процесорів і відповідних їм тепловідводів, а також новими типами шин (наприклад, AGP). Ці проблеми були враховані під час розробки форм-фактора *NLX*.

У форм-факторі *LPX* додаткова вертикальна плата підключається до системної плати. Основна особливість системи *NLX* полягає в тому, що, на відміну від *LPX*, системна плата підключається до роз'єму вертикально розташованої додаткової плати. Подібна конструкція дозволяє витягувати системну плату без відключення вертикальної плати або підключених до неї адаптерів. Крім того, системна плата *NLX* не містить будь-яких внутрішніх кабелів або підключених до неї роз'ємів. Пристрої, що зазвичай підключаються до системної плати (кабелі дисковода, блоки живлення, індикаторні лампи лицьової панелі, роз'єми вимикачів і тому подібне), підключені замість цього до додаткової вертикальної плати. Використовуючи те, що основні роз'єми знаходяться на додатковій платі, можна зняти верхню кришку корпусу комп'ютера і без особливих зусиль витягувати системну плату, не відключивши при цьому жодного кабелю або роз'єму.

Системна плата *NLX*, як і велика частина форм-факторів, відрізняється унікальною ступінчастою конструкцією гнізд вводу/виводу і схемою розташування роз'ємів (рис. 5.17).

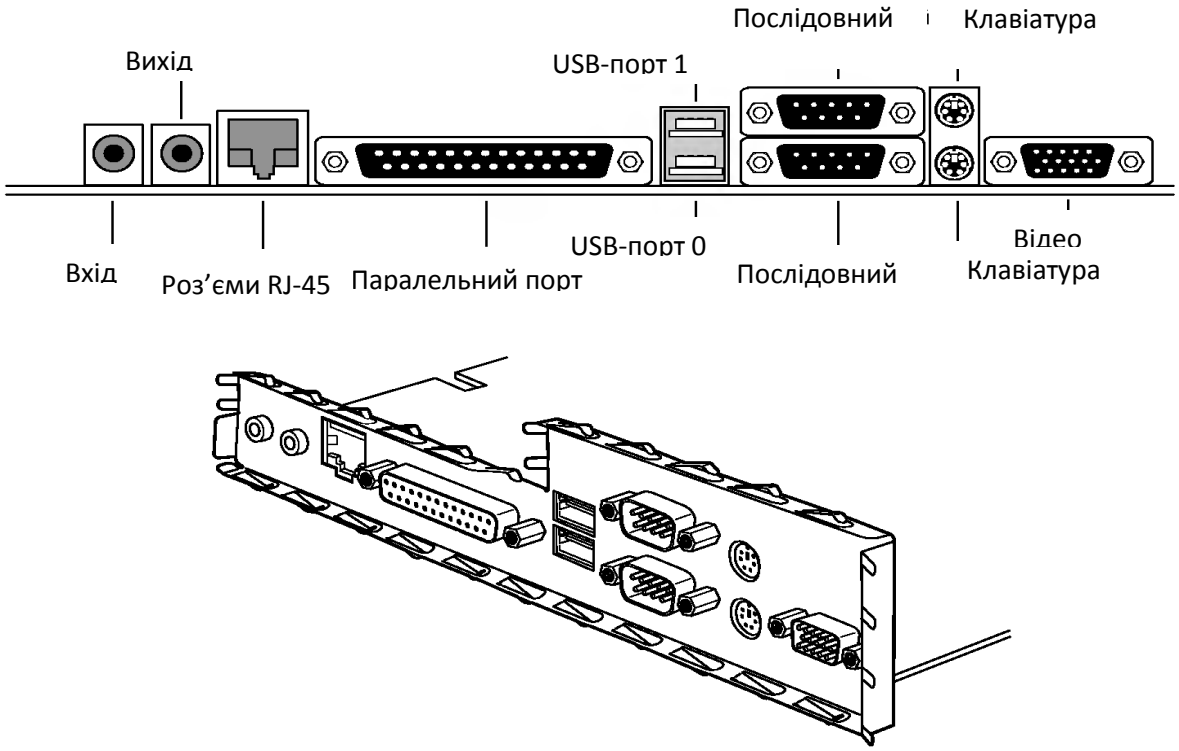


Рис.5.17. Область роз'ємів системної плати форм-фактора NLX

Ця конструкція передбачає розташування основних роз'ємів у нижньому ряду, а також забезпечує можливість підключення здвоєних роз'ємів.

BTX. Офіційне представлення специфікації *The Balanced Tech-nology Extended (BTX) 1.0 Public Release* відбулося у липні 2004 року. Призначення *BTX* офіційно сформульоване компанією *Intel* таким чином: специфікації розроблені з метою стандартизації інтерфейсів і визначення форм-фактора для обчислювальних систем в області їх електричних, механічних і термічних властивостей. Специфікації описують механічні і електричні інтерфейси для розробки системних плат, шасі, блока живлення та інших системних компонентів.

Головні переваги форм-фактора *BTX* перед *ATX* виглядають так:

- можливість застосування низькопрофільних компонентів для збірки мініатюрних систем;
- продумане розміщення елементів системи всередині корпусу з урахуванням шляхів проходження потоків повітря і термобалансу;
- масштабованість у рамках доступних модифікацій: повноформатного *BTX, micro-BTX, pico-BTX*;
- можливість використання невеликих блоків живлення;
- оптимізована конструкція кріплення системної плати, якісні механічні елементи для установки масивних компонентів.

5.5. АРХІТЕКТУРА СИСТЕМИ ВВОДУ / ВИВОДУ

5.5.1. Призначення та структура системи вводу / виводу

Крім центрального процесора (ЦП) і пам'яті, третім ключовим елементом архітектури комп'ютера є *система вводу/виводу* (СВВ). В даний час продуктивність та ефективність використання ОМ визначається не тільки можливостями її процесора та ОЗП, а в більшій мірі складом її периферійних пристроїв (ПП), їх технічними даними та способом організації їх сумісної роботи з центральною частиною ОМ. Система вводу/виводу призначена для забезпечення обміну інформацією між ядром ОМ і різноманітними периферійними пристроями. Технічні і програмні засоби СВВ здійснюють, у загальному вигляді, перетворення форми уявлення даних, які використовує процесор, у форму, придатну для сприймання користувачем або зовнішнім об'єктом.

Програмне забезпечення СВВ включає способи подання даних та інформації управління (формати команд обміну та управляючих слів інформаційної взаємодії) і сукупність управляючих програм обміну даними.

Технічні або апаратні засоби СВВ, крім безпосереднього перетворення форми даних, здійснюють функції зі зміни форматів та управління передачею даних.

Основна задача СВВ – це звільнення процесора від участі в операціях обміну між ОЗП та ПП. Для розв'язання цієї задачі СВВ повинна забезпечувати:

- функціонування обчислювальної системи зі змінним складом обладнання;
- паралельну роботу ПП як по відношенню до роботи процесора, так і по відношенню один до одного;
- уніфікацію управління ПП різних типів;
- пріоритетність звернення до ОЗП різних пристроїв ОМ та ПП.

Змінність складу обладнання полягає в тому, що система команд, апаратура зв'язку з ПП та логіка управління ними повинні бути такими, щоб приєднання такого пристрою не викликало в існуючої частини системи ніяких інших змін, крім зміни програм та з'єднання кабелів. При цьому кожен користувач може вибирати таку конфігурацію (склад) своєї моделі ОМ, яка відповідає його потребам з мінімальними затратами. По мірі збільшення потреб користувач має можливість простого під'єднання додаткових ПП, навіть тих, які ще тільки будуть розроблені.

Вимога паралельної роботи ПП та процесора викликала необхідність виділення схем управління ПП зі складу процесора та надання їм достатнього ступеня автономності. Однак недоцільно включати в кожен ПП усі схеми управління ним, оскільки це приведе до росту обсягу обладнання: багато функцій управління ПП принципово не залежать від типу пристрою і

виконуються з різницею у часі; ці функції можуть бути покладені на загальну для своїх ПП апаратуру. Таким чином, у складі сучасних ОМ з'явились спеціальні пристрої управління вводом/виводом, які забезпечують обмін інформацією між основною пам'яттю та ПП з використанням паралельної роботи перемінного набору ПП різних типів та процесорів. Ці пристрої одержали назву каналів вводу/виводу (КВВ).

Технічно система вводу/виводу в рамках обчислювальної машини реалізується комплексом *модулів вводу/виводу* (МВВ). Модуль вводу/ виводу виконує з'єднання ПП з ядром ОМ і різноманітні комунікаційні операції між ними. Дві основні функції МВВ:

- забезпечення інтерфейсу з ЦП та пам'яттю («*великий*» *інтерфейс*);
- забезпечення інтерфейсу з одним або декількома периферійними пристроями («*малий*» *інтерфейс*).

Існують три основних способи підключення СВВ до ядра процесора (рис. 5.18) [25].

У варіанті *з роздільними шинами пам'яті і вводу/виводу* (рис. 5.18, *а*) обмін інформацією між ЦП та пам'яттю фізично відділений від вводу/виводу, оскільки забезпечується повністю незалежними шинами.

Це надає можливість здійснювати звернення до пам'яті одночасно з виконанням вводу/виводу. Крім того, даний архітектурний варіант ОМ дозволяє спеціалізувати кожну з шин, враховувати формат даних, які пересилаються, особливості синхронізації обміну та ін. Зокрема, шина вводу/виводу, з урахуванням характеристик реальних ПП, може мати меншу пропускну спроможність, що дозволяє знизити витрати на її реалізацію. Недоліком рішення можна вважати велику кількість точок підключення до ЦП.

Другий варіант – *з сумісно використовуваними лініями даних і адреси* (рис. 5.18, *б*). Пам'ять та СВВ мають загальні для них лінії адреси і лінії даних, які розділені у часі. У той же час управління пам'яттю і СВВ, а також синхронізація їх взаємодії з процесором здійснюється незалежно за роздільними лініями управління. Це дозволяє враховувати особливості процедур звертання до пам'яті та до модулів вводу/виводу і досягати найбільшої ефективності доступу до комірок пам'яті та периферійних пристроїв.

Останній тип архітектури ОМ припускає підключення СВВ до системної шини *на загальних правах з процесором і пам'яттю* (рис. 5.18, *в*). Потенціально можливий також варіант підключення периферійних пристроїв до системної шини напряму, без використання МВВ, але проти нього можна висунути одразу декілька аргументів. По-перше, у цьому випадку ЦП знадобилось би оснащувати універсальними схемами для управління будь-яким ПП. У разі великого різноманіття периферійних пристроїв, які мають до того ж різні принципи дії, такі схеми виявляються занадто складні і надмірні.

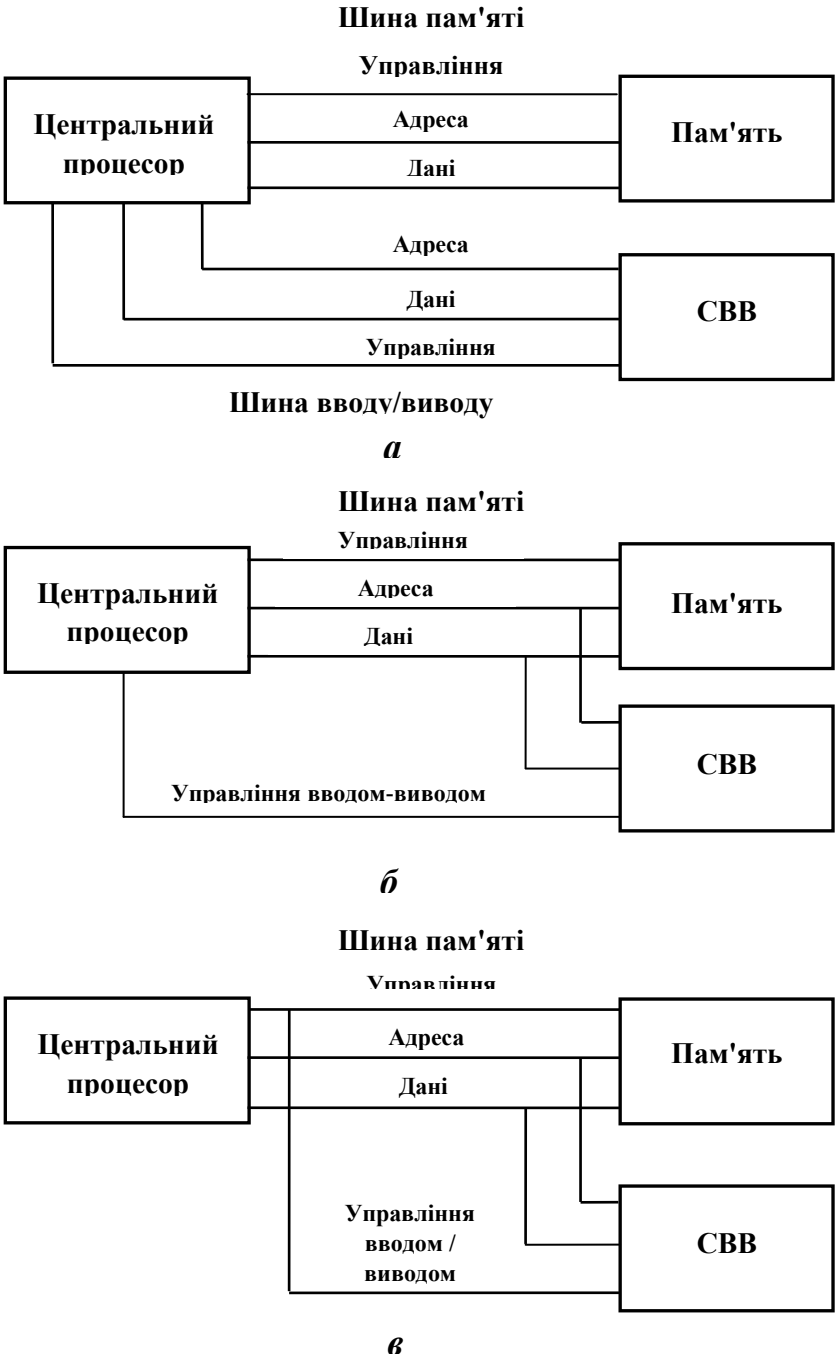


Рис. 5.18. Місце системи вводу/виводу в архітектурі ОМ:
а – з роздільними шинами пам'яті і вводу/виводу;
б – з сумісно використовуваними лініями даних і адреси;
в – підключення на загальних правах з процесором і пам'яттю

По-друге, пересилка даних під час вводу і виводу проходить значно повільніше, чим під час обміну між ЦП і пам'яттю, і було б невигідно використовувати для обміну інформацією з ПП високошвидкісну системну шину. І, нарешті, в ПП часто використовуються інші формати даних і довжина слова, ніж в ОМ, до яких вони підключені.

Адресний простір СВВ. Операції вводу/виводу припускають наявність деякої системи адресації, яка дозволяє вибрати один з модулів СВВ, а також

один з підключених до нього периферійних пристроїв. Адреса модуля і ПП є складовою частиною команди.

Адресний простір СВВ може бути суміщеним з адресним простором пам'яті або виділеним. У разі суміщеня адресного простору для адресації модулів відводиться певна область адрес. Звичайно усі операції з модулем вводу/виводу здійснюються з використанням регістрів, які до нього входять: управління, стану, даних. Фактично процедура вводу/виводу зводиться до запису інформації в одні регістри МВВ та зчитування її з інших регістрів. Це дозволяє розглядати регістри МВВ як комірки основної пам'яті та працювати з ними за допомогою звичайних команд звернення до пам'яті, при цьому в системі команд ОМ взагалі можуть бути відсутніми спеціальні команди вводу/виводу.

У випадку виділеного адресного простору для звернення до модулів вводу/виводу застосовуються спеціальні команди і окрема система адрес. Це дозволяє розділити шини для роботи з пам'яттю і шини вводу/виводу, що дає можливість суміщати у часі обмін з пам'яттю і вводом/виводом. Крім того, адресний простір пам'яті може бути використаним за прямим призначенням у повному обсязі. В обчислювальних машинах фірми IBM і мікро-ЕОМ на базі процесорів фірми Intel система вводу/виводу, як правило, організовується відповідально до концепції виділеного адресного простору.

Периферійні пристрої. Зв'язок ОМ із зовнішнім середовищем здійснюється за допомогою різноманітних периферійних пристроїв. Кожний ПП під'єднується до МВВ за допомогою індивідуальної шини. Інтерфейс, за допомогою якого здійснюється така взаємодія МВВ і ПП, часто називають *малим*. Індивідуальна шина забезпечує обмін даними і управляючими сигналами, а також інформацією про стан учасників обміну (рис. 5.19).

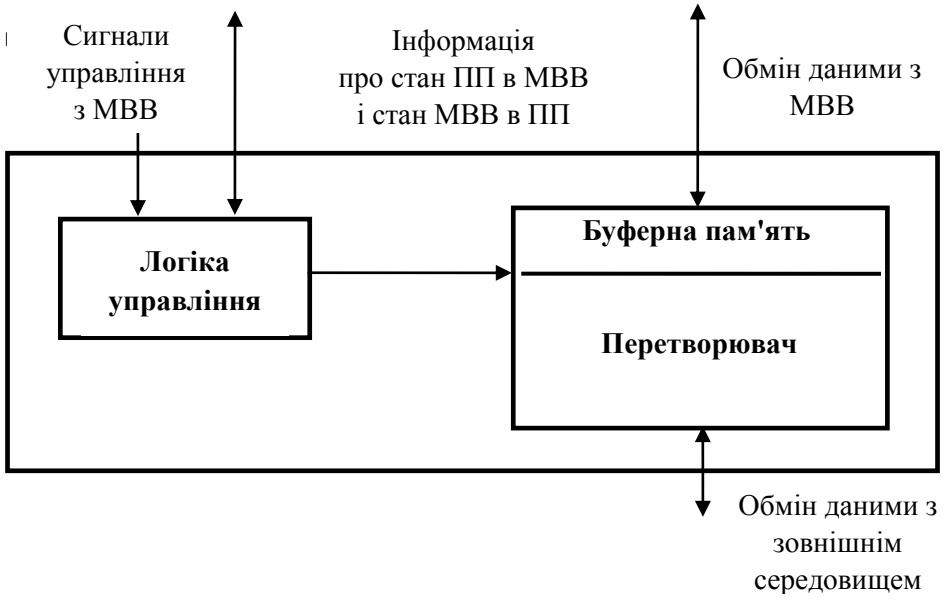


Рис. 5.19. Структура периферійного пристрою

Інтерфейс з МВВ реалізується у виді сигналів управління, стану і даних. *Дані* представлені сукупністю бітів, які повинні бути переданими в модуль вводу/виводу або отриманими з нього. *Сигнали управління* визначають функцію, яка повинна бути виконана периферійним пристроєм. *Сигнали стану* характеризують поточний стан пристрою, зокрема чи включений ПП та чи готовий він до передачі даних.

Логіка управління – це схеми, які координують роботу ПП відповідно до напрямку передачі даних. Задачею *перетворювача* є трансформація інформаційних сигналів, що мають різноманітну фізичну природу, у електричні сигнали та навпаки. Звичайно сумісно з перетворювачем використовується *буферна пам'ять*, яка забезпечує тимчасове зберігання даних, що пересилаються між МВВ і ПП.

Модулі вводу/виводу. Модуль вводу/виводу у складі ОМ відповідає за управління одним або декількома ПП та за обмін даними між цими пристроями з одного боку, і основною пам'яттю або регістрами ЦП – з іншого. Основні функції МВВ можна сформулювати таким чином:

- локалізація даних;
- управління і синхронізація;
- обмін інформацією;
- буферизація даних;
- виявлення помилок.

Локалізація даних – це можливість звертання до одного з ПП, а також адресація даних на ньому. Адреса ПП звичайно міститься в адресній частині команди вводу/виводу. Як було відмічено раніше, у складі СВВ може бути декілька модулів вводу/виводу. Кожному модулю призначається певний діапазон адрес.

Управління і синхронізація. Функція управління і синхронізації полягає в тому, що МВВ повинен координувати переміщення даних між внутрішніми регістрами ОМ і периферійними пристроями. Під час розробки системи управління і синхронізації модуля вводу/виводу необхідно враховувати ряд факторів.

Перш за все, необхідно приймати до уваги, що ЦП може взаємодіяти одночасно з декількома ПП, причому їх швидкодія змінюється в дуже широких межах – від декількох байтів у секунду в терміналах до десятків мільйонів байтів у секунду під час обміну з магнітними дисками. Якщо в системі використовуються шини, кожна взаємодія між ЦП і МВВ включає в себе одну або декілька процедур арбітражу.

На відміну від обміну з пам'яттю, процеси вводу/виводу і робота ЦП протікають не синхронно. Чергова порція інформації може бути видана на пристрій виводу лише тоді, коли цей пристрій готовий її прийняти. Аналогічно,

введення від пристрою вводу можливе тільки у випадку доступності інформації на пристрої вводу. Таким чином, МВВ зобов'язаний забезпечити центральний процесор інформацією про власну готовність до обміну, а також про готовність ПП, що підключені до модуля. Крім того, процесор повинен володіти оперативною інформацією про інші події, що відбуваються в СВВ.

Обмін інформацією. Основною функцією МВВ є забезпечення обміну інформацією. З боку «великого» інтерфейсу – це обмін з ЦП, а з боку «малого» інтерфейсу – обмін з ПП. У цьому випадку під час вводу/виводу процесор виконує такі операції:

1. Вибір потрібного периферійного пристрою.
2. Визначення стану МВВ та ПП.
3. Видача вказівки МВВ на підключення потрібного ПП до процесора.
4. Отримання від МВВ підтвердження про підключення потрібного ПП до процесора.
5. Розпізнавання сигналу готовності пристрою до передачі чергової порції інформації.
6. Приймання (передача) порції інформації.
7. Циклічне повторення двох попередніх пунктів до завершення передачі інформації у повному обсязі.
8. Логічне відключення ПП від процесора.

Буферизація. Швидкість обміну інформацією усіх ПП значно менше швидкодії ЦП і пам'яті. Ця різниця компенсується за рахунок буферизації. Під час виводу інформації на ПП дані пересилаються з основної пам'яті в МВВ з великою швидкістю. В модулі ці дані буферизуються і потім прямують в ПП зі швидкістю, яка потрібна для нього. При вводі з ПП дані буферизуються так, щоб не заставляти пам'ять працювати в режимі повільної передачі. Таким чином, МВВ повинен володіти здатністю функціонувати як зі швидкістю пам'яті, так і зі швидкістю ПП.

Виявлення помилок. Виявлення помилок, які виникають у процесі вводу/виводу – це одна з найважливіших функцій МВВ. Центральний процесор слід оповіщати про кожний випадок виявлення помилки. Причинами виникнення останніх бувають різноманітні фактори, які можна звести до таких груп:

- вплив зовнішнього середовища;
- старіння елементної бази;
- системне програмне забезпечення;
- користувацьке програмне забезпечення.

Способи виявлення і виправлення помилок вводу/виводу будуть розглянуті в останньому розділі.

Структура модуля вводу/виводу. Структура МВВ у значній мірі залежить від числа та складності периферійних пристроїв, але в загальному виді він має форму, яка показана на рис. 5.20.

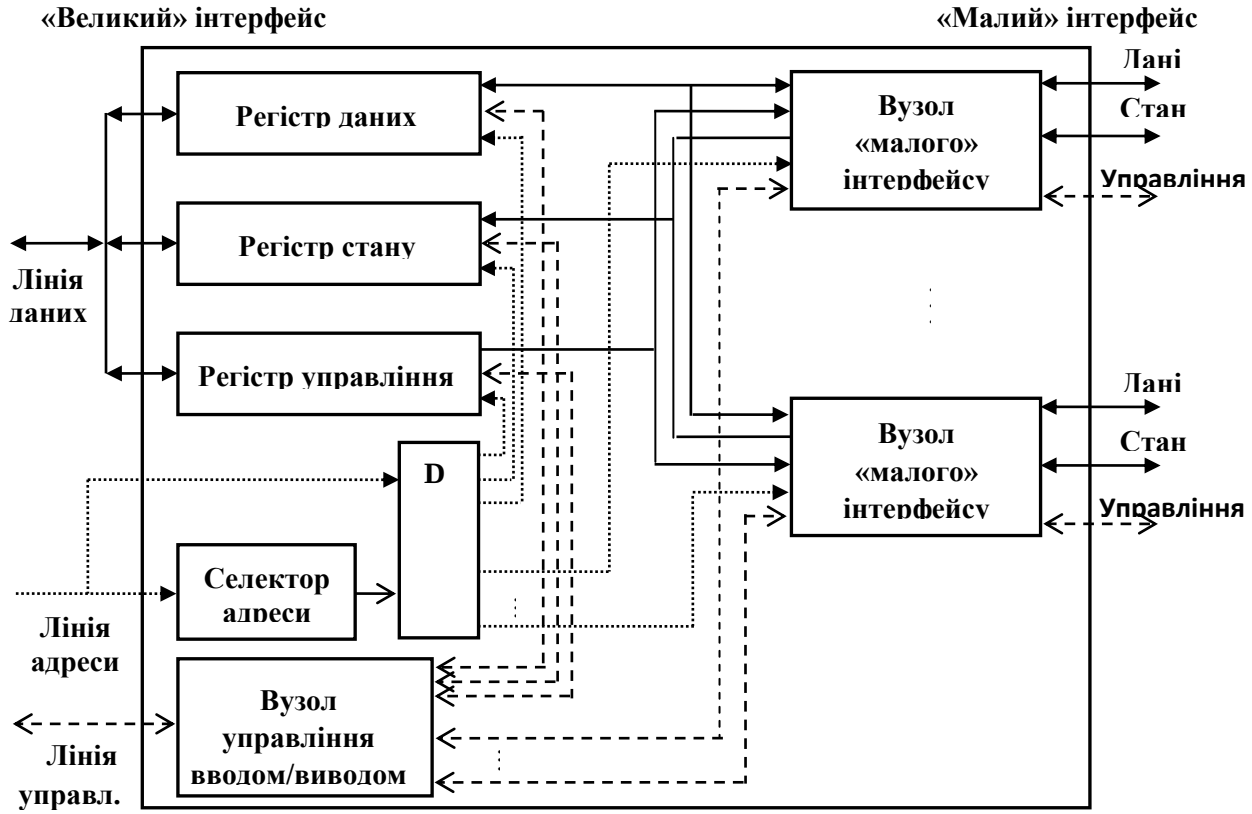


Рис. 5.20. Структура модуля вводу/виводу

Зв'язок МВВ з ядром ОМ здійснюється за допомогою системної або спеціалізованої шини. З цього боку в МВВ реалізується так званий «великий» інтерфейс.

Дані, що передаються в модуль та з нього, буферизуються в *регістрі даних*. Буферизація дозволяє компенсувати відмінності у швидкодії ядра ОМ і периферійних пристроїв. Більшість периферійних пристроїв орієнтовано на побайтний обмін інформацією. Побайтова пересилка інформації по «широкій» системній шині – вкрай неефективне рішення, тому з боку «малого» інтерфейсу регістр даних часто доповнюють вузлом упаковки-розпаковки (на схемі не показаний). Цей вузол під час вводу забезпечує послідовне побайтове заповнення регістра даних (упаковку), а під час виводу – послідовну побайтову видачу вмісту регістру на ПП (розпаковку). В результаті під час обміну даними через «великий» інтерфейс зайнята вся ширина шини даних.

Крім регістру даних у складі МВВ є також регістр управління і регістр стану (або суміщений регістр управління-стану).

У *регістрі управління* (РУ) фіксуються команди управління модулем або підключеними до нього ПП. Ці команди надходять з ЦП. У складних МВВ присутні декілька регістрів управління.

Регістр стану (РС) служить для зберігання бітів стану МВВ та ПП, що підключені до нього. Вміст певного розряду регістра може характеризувати, наприклад, готовність пристрою вводу до приймання чергової порції даних, зайнятість пристрою вводу або знаходження ПП в автономному режимі (*offline*).

Процедура вводу/виводу припускає можливість роботи з кожним регістром МВВ або периферійним пристроєм окремо. Така можливість забезпечується системою адресації. Кожному модулю в адресному просторі вводу/виводу надається унікальний набір адрес, кількість яких залежить від числа елементів, що адресуються. Адреса, що поступає з ЦП, за допомогою селектора перевіряється на належність до діапазону, який є виділеним даному МВВ. У випадку підтвердження дешифратор *DC* виконує розкодування адреси і тим самим дозволяє роботу з відповідним регістром модуля або ПП.

Вузол управління вводом/виводом грає роль місцевого пристрою управління МВВ. На нього покладається дві задачі: забезпечення взаємодії з ЦП та координація роботи усіх складових МВВ. Зв'язок з ЦП реалізований лініями управління, по яких з ЦП в модуль поступають сигнали, що служать для синхронізації операцій вводу і виводу. В зворотному напрямку передаються сигнали, які інформують про події, що відбуваються в модулі, наприклад сигнали переривання. Друга функція вузла управління реалізована за допомогою внутрішніх сигналів управління.

З боку «малого» інтерфейсу МВВ забезпечує підключення зовнішніх пристроїв та взаємодію з ними. Кожний з зовнішніх пристроїв «обслуговується» власним вузлом «малого» інтерфейсу, який реалізує прийнятий для даного ПП стандартний протокол взаємодії.

Під час управління широким спектром ПП модуль повинен по можливості звільняти ЦП від знання деталей конкретних ПП, так щоб ЦП міг управляти будь-яким пристроєм за допомогою простих команд читання і запису. МВВ бере на себе функції синхронізації, узгодження форматів даних та ін.

Модуль вводу/виводу, який бере на себе детальне управління ПП і спілкується з ЦП тільки за допомогою команд високого рівня, часто називають *каналом вводу/виводу* (КВВ). Найбільш простий МВВ, який потребує детального управління з боку ЦП, називають *контролером вводу/виводу* або *контролером пристрою*. Як правило, контролери вводу/виводу типові для мікро-ЕОМ, а канали вводу/виводу – для універсальних ОМ.

Методи управління вводом/виводом.

В ОМ знайшли застосування три способи організації вводу/виводу:

- програмно керований ввід/вивід;
- ввід/вивід за перериваннями;
- прямий доступ до пам'яті.

Під час *програмно керованого вводу/виводу* всі зв'язані з цим події проходять за ініціативою центрального процесора та під його повним контролем.

ЦП виконує програму, яка забезпечує пряме управління процесом вводу/виводу, включаючи перевірку стану пристрою, видачу команд вводу або виводу. Після видачі команди в МВВ центральний процесор повинен очікувати завершення її виконання, і, оскілки ЦП працює швидкісніше, ніж МВВ, це приводить до втрати часу.

Ввід/вивід за перериваннями багато в чому збігається з програмно керованим методом. Відмінність у тому, що після видачі команди вводу/виводу ЦП не повинен циклічно опитувати МВВ для з'ясування стану пристрою. Замість цього процесор може продовжувати виконання інших команд до тих пір, доки не отримає запиту переривання від МВВ, що сповіщає про завершення виконання попередньої команди вводу/виводу. Як і під час програмно керованого вводу/виводу, ЦП відповідає за витягання даних з пам'яті (під час виводу) та запис даних у пам'ять (під час вводу).

Підвищення як швидкості вводу/виводу, так і ефективності використання ЦП забезпечує третій спосіб – *прямий доступ до пам'яті* (ПДП). У цьому режимі основна пам'ять і МВВ обмінюються інформацією напряму, безпосередньо минувши процесор. ПДП припускає наявність на системній шині додаткового модуля – *контролера прямого доступу до пам'яті* (КПДП), який здатний брати на себе функції ЦП по управлінню системною шиною і забезпечувати пряму пересилку інформації між оперативною пам'яттю і ПП без участі центрального процесора. По суті, КПДП – це і є модуль вводу/виводу, який реалізує режим прямого доступу до пам'яті.

Якщо ЦП бажає прочитати або записати блок даних, він повинен помістити в КПДП інформацію, що характеризує майбутню дію. Цей процес має назву ініціалізація КПДП і включає в себе занесення в контролер таких параметрів:

- вид запиту (читання або запис);
- адреса пристрою вводу/виводу;
- адреса початкової комірки пам'яті, з якої буде витягатись або куди буде записуватись інформація;
- кількість слів, що підлягають читанню або запису.

Після ініціалізації процес пересилання інформації може бути початим у будь-який час. Ініціатором обміну може бути як ЦП, так і ПП. Пристрій, який бажає почати ввід/вивід, сповіщає про це контролер подачею відповідного сигналу. Отримавши такий сигнал, КПДП видає в ЦП сигнал «Запит ПДП». У відповідь ЦП звільняє шини адреси і даних, а також ті лінії шини управління, по котрим передаються сигнали, що управляють операціями на шині адреси (ША) і шині даних (ШД). До таких, перш за все, відносяться лінії *ЧтЗП* (читання запам'ятовуючого пристрою), *ЗпЗП* **(**запис у запам'ятовуючий пристрій), *Вив, Вв* та лінія видачі адреси на ША. Далі ЦП відповідає контролеру сигналом «Підтвердження ПДП», який для останнього означає, що йому даються права на управління системною шиною і можна приступати до пересилання даних.

5.5.2. Канали та процесори вводу/виводу

Основна ціль системи вводу/виводу – максимальне звільнення ЦП від управління процесами вводу/виводу. Деякі шляхи розв'язання цієї задачі вже були розглянуті. Наступними кроками в подоланні проблеми можуть бути:

1. Розширення можливостей МВВ та надання йому прав процесора зі спеціалізованим набором команд, які орієнтовані на операції вводу/виводу. ЦП надає такому МВВ вказівку виконувати програму вводу/виводу, яка зберігається в пам'яті ЕОМ. МВВ витягує і виконує команди цієї програми без участі центрального процесора і перериває ЦП тільки після завершення всієї програми вводу/виводу.
2. МВВ, який розглянуто вище, надається власна локальна пам'ять, при цьому можливе управління багатьма пристроями вводу/виводу з мінімальним використанням ЦП.

У першому випадку МВВ називають *каналом вводу/виводу* (КВВ), а в другому – *процесором вводу/виводу* (ПВВ). У принципі різниця між каналом і процесором вводу/виводу достатньо умовна, тому надалі будемо застосовувати термін «канал».

В ЕОМ з каналами вводу/виводу центральний процесор практично не бере участі в управлінні зовнішніми пристроями. Усі функції ЦП зводяться до запуску і зупинки операцій в КВВ, а також перевірки стану каналу і ПП, що підключені до нього. Для цих цілей ЦП використовує тільки декілька (від 4 до 7) команд. Наприклад, в IBM 360 таких команд чотири:

- Почати ввід/вивід;
- Зупинити ввід/вивід;
- Перевірити ввід/вивід;
- Перевірити канал.

Обмін інформацією між КВВ і основною пам'яттю здійснюється через системну шину ОМ. ПП підключається до каналу не безпосередньо, а через блоки управління периферійними пристроями (БУПП). БУПП приймає від каналу команди по управлінню периферійним пристроєм (читання, запис, переміщення носія або магнітної головки та ін.), і перетворює їх в сигнали управління, які властиві даному типу ПП. Звичайно один БУПП може обслуговувати декілька однотипних ПП, але для підключення швидкодіючих периферійних пристроїв часто використовують індивідуальні блоки управління. В свою чергу, деякі ПП можуть підключатись одночасно до кількох БУПП. Це дозволяє використовувати вільний тракт іншого БУПП при зайнятості даного БУПП, що обслуговує інший підключений до нього ПП.

З введенням СВВ на центральний процесор ЕОМ, головним чином, покладається тільки ініціювання операції вводу/виводу (ВВ). Процесор видає в канал команду ВВ, після чого звільняється і може продовжувати виконання

основної програми. Управління операцією ВВ бере на себе канал. Він повинен визначити вид операції, вибрати ПП, визначити напрямок обміну, можливість обміну з указаним ПП та запустити його у роботу. Під запуском розуміють повідомлення ПП коду режиму роботи та переведення його в режим автоматичного управління. Для розв'язання усіх цих задач канал повинен одержати від процесора інформацію про операцію вводу/виводу, про номер каналу, про адресу ПП, з якими повинен бути проведений обмін даними, та про першу команду канальної програми, яка буде управляти виконанням заданої операції.

КВВ реалізує операції вводу/виводу шляхом виконання так званої *канальної програми.* Канальні програми для кожного ПП, з яким буде вестись обмін інформацією, описують потрібну послідовність операцій вводу/виводу та зберігаються в основній пам'яті ОМ. Роль команд у канальних програмах виконують *управляючі слова каналу* (УСК). Адреса першого УСК може знаходитись у форматі команди ВВ або знаходитись у фіксованій комірці оперативної пам'яті. Схема командної взаємодії для випадку, коли адреса першого УСК міститься у форматі команди ВВ, показана на рис. 5.21.

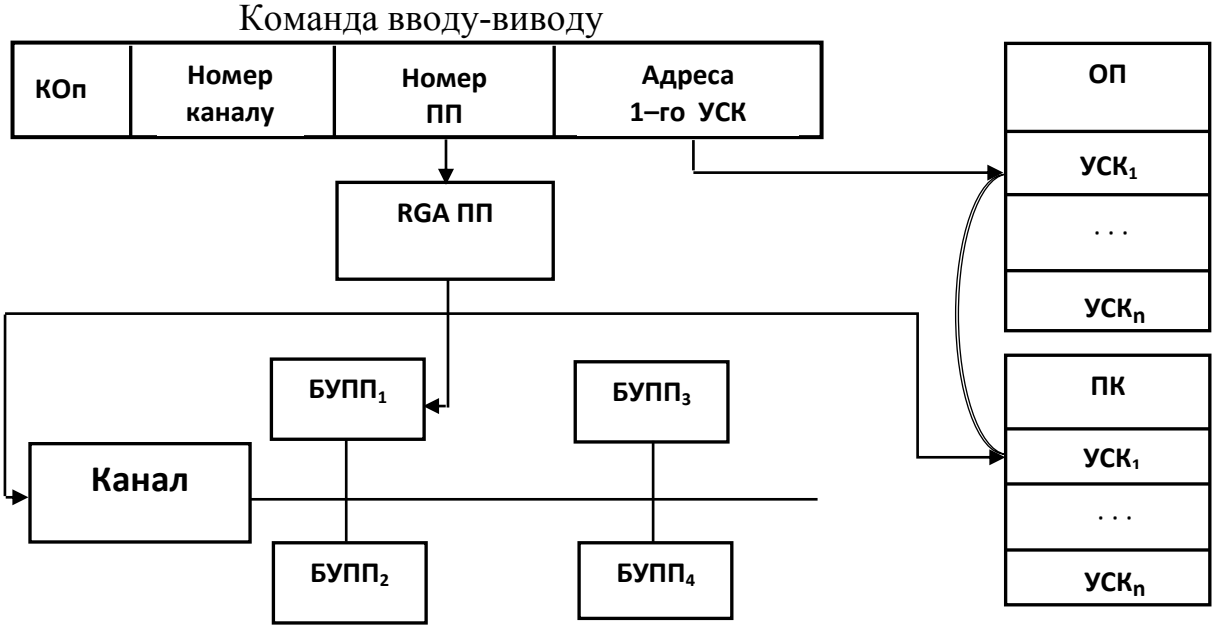


Рис. 5.21. Схема командної взаємодії СВВ

Після запуску вибраного ПП та передачі першого УСК в канал відбувається паралельна робота процесора з реалізації основної програми та каналу разом з ПП з організації обміну даними між ПП та ОЗП. У разі готовності ПП до обміну він посилає сигнал запиту на одержання або видачу чергового слова інформації.

З цього моменту починається інформаційна взаємодія процесора та каналу, в основі якого лежить безпосередній обмін інформацією у режимі затримок процесора. Робота процесора призупиняється, наприклад, після кожного такту

виконання команди відбувається обмін інформацією та її компоновка в масиві пам'яті під взаємодією команд, ознак та адрес, закодованих в управляючих словах каналів.

Після обміну проходить модифікація УСК з метою формування адрес наступних управляючих слів та даних, а також визначення кінця обміну інформацією.

Спосіб організації взаємодії ПП з каналом визначається співвідношенням швидкодії ОЗП та ПП. За цією ознакою всі ПП прийнято ділити на два класи: швидкодіючі (накопичувачі на магнітних дисках та стрічках) зі швидкістю приймання і видачі інформації приблизно 1 Мбайт/с і повільнодіючі (монітори, друкувальні пристрої та ін.) зі швидкістю близько 1 Кбайт/с. Швидкодія основної пам'яті зазвичай вища. З урахуванням продуктивності ПП в КВВ реалізуються два режими роботи: *мультиплексний* (режим розділення часу) та *монопольний*.

У *мультиплексному режимі* декілька периферійних пристроїв розділяють канал у часі, при цьому кожний з паралельно працюючих з каналом ПП зв'язується з КВВ на короткі проміжки часу тільки після того, як ПП буде готовий до прийому або видачі чергової порції інформації (байта, групи байтів і т. д). Така схема прийнята у *мультиплексному каналі вводу/виводу*. Якщо протягом сеансу зв'язку пересилається один байт або декілька байтів, що утворюють одне машинне слово, канал називається *байт-мультиплексним*. Канал, у якому в межах сеансу зв'язку пересилка даних відбувається поблочно, має назву *блок мультиплексний*.

У *монопольному режимі* після встановлення зв'язку між каналом і ПП останній монополізує канал на весь час до завершення ініційованої процесором канальної програми та всіх передбачених цією програмою пересилок даних між ПП і ОЗП. На весь час виконання канальної програми канал стає недоступним для інших ПП. Дану процедуру забезпечує *селекторний канал вводу/виводу*. Відмітимо, що в блок-мультиплексному каналі в рамках сеансу зв'язку пересилка блока даних відбувається у монопольному режимі.

Канальна підсистема. В сімействах універсальних ОМ, що прийшли на зміну IBM 360 (сімейства IBM 370 і особливо IBM 390), концепція системи вводу/ виводу на базі каналів отримала подальший розвиток і вилилася в так звану *канальну підсистему вводу/виводу* (КПВВ). Головна ідея полягає в інтеграції окремих КВВ в єдиний спеціалізований процесор вводу/ виводу з великим числом канальних трактів і підканалів. Блоки управління зовнішніми пристроями зазвичай підключаються до декількох канальних трактів, що дозволяє динамічно міняти шлях пересилки інформації з урахуванням поточної їх завантаженості. Крім того, різні канальні тракти можуть володіти різною пропускною спроможністю, і під час вибору трактів для підключення певних ПП може бути врахована їх швидкодія.

Одну з найбільш досконалих канальних підсистем мають ОМ сімейства IBM 390. У ній передбачено використання до 65536 підканалів і до 256 канальних трактів. Реалізовано два типи канальних трактів: паралельний і послідовний.

Паралельні канальні тракти за своїми можливостями і принципом дії аналогічні розглянутим раніше мультиплексному і селекторному каналах, але на відміну від них є універсальними, тобто можуть працювати в байт-мультиплексному, блок-мультиплексному і селекторному режимах. Такі канальні тракти в КПВВ називають паралельними, оскільки вони забезпечують пересилку інформації паралельним кодом.

Для роботи з ПП, зв'язаними з КПВВ волоконно-оптичними лініями, використовуються *послідовні канальні тракти*, що реалізовують протокол ESCON (Enterprise Systems Connection Architecture). Послідовний канальний тракт розрахований на передачу інформації тільки в послідовному коді і лише в селекторному режимі. Для підключення блоків управління зовнішніми пристроями (БУЗП) до ESCON-тракту служать спеціальні пристрої, що називаються ESCON-директорами. Кожний такий пристрій може забезпечити одночасне підключення до 60 БУЗП і одночасну передачу інформації від 30 з них з швидкістю до 10 Мбайт/с.

Крім того, в КПВВ передбачені спеціальні комунікаційні канальні тракти для підключення до мереж ОМ, модемів, інших систем.

В принципі, основна перевага КПВВ – *динамічний перерозподіл канальних трактів* – у якійсь мірі може бути реалізовано і в рамках кожного окремого каналу. Проте об'єднання всіх канальних ресурсів у єдину канальну підсистему дозволяє застосувати оптимальну стратегію динамічного розподілу та використання цих ресурсів і завдяки цьому досягти якісно нового рівня ефективності системи вводу/виводу.

КОНТРОЛЬНІ ПИТАННЯ

1. Дайте визначення поняття «Апаратний інтерфейс».
2. Що називають шиною інтерфейсу?
3. Що значить «уніфікація і стандартизація» інтерфейсу?
4. У чому суть «фізичної» і «логічної» реалізацій інтерфейсу?
5. Які відмінності між «апаратним інтерфейсом» і «інтерфейсом корис-тувача»?
6. Чим відрізняється інтерфейс великих ОМ від інтерфейсу міні- і мікро-ЕОМ?

7. Охарактеризуйте основні параметри інтерфейсу.
8. У чому відмінність послідовного і паралельного інтерфейсу?
9. Чим відрізняється робота синхронного і асинхронного інтерфейсів?
10. Охарактеризуйте основні типи інтерфейсів, які визначаються способом з'єднання пристроїв.
11. Чим обумовлена організація комбінованих інтерфейсів?
12. Яку функцію виконують шини комп'ютера?
13. Яку функцію виконує шина «процесор-пам'ять?
14. Дайте коротку характеристику шинам вводу/виводу.
15. Дайте характеристику основним стандартам системної шини.
16. Яку функцію виконують послідовний і паралельний порти?
17. Дайте характеристику периферійної шини USB.
18. Охарактеризуйте шину Fire Wire (IEEE 1394).
19. Яку функцію виконує системна плата комп'ютера?
20. Що таке форм-фактор системної плати?
21. Дайте порівняльну характеристику найбільш відомих форм-факторів системних плат.
22. Дайте коротку характеристику форм-фактору *ATX*.
23. Яку функцію несуть форм-фактори *micro-ATX* і *flex-ATX*?
24. Назвіть призначення і дайте коротку характеристику форм-фактора *NLX*.
25. Яка основна функція системи вводу/виводу?
26. Яку функцію виконує модуль вводу/ виводу?
27. У чому відмінність «*великого*» і «*малого*» інтерфейсів?
28. Дайте характеристику основних способів підключення СВВ до ядра процесора.
29. Охарактеризуйте поняття «Адресний простір СВВ».
30. Дайте характеристику основних функцій МВВ.
31. Які операції виконує процесор під час вводу/виводу?
32. Назвіть призначення основних пристроїв модуля вводу/ виводу.
33. Дайте характеристику основних способів організації вводу/виводу.
34. Поясніть принцип дії каналу вводу/виводу.
35. Дайте характеристику основних режимів роботи каналів вводу/ виводу.
36. Назвіть призначення і дайте коротку характеристику канальній підсистемі вводу/виводу.

6. ПАРАЛЕЛЬНІ КОМП'ЮТЕРНІ СИСТЕМИ

В умовах постійно зростаючих вимог до продуктивності обчислювальної техніки все більш явними стають обмеження класичної фон-нейманівської архітектури. Подальший розвиток обчислювальної техніки зв'язаний з переходом до паралельних обчислень як у рамках однієї обчислювальної машини, так і шляхом створення багатопроцесорних систем і мереж, які об'єднують велику кількість окремих процесорів або окремих обчислювальних машин. Для такого підходу замість терміну «обчислювальна машина» більш підходить термін «обчислювальна система» (ОС). Головною особливістю такої системи є наявність у ній засобів, які реалізують паралельну обробку інформації.

6.1. РІВНІ ПАРАЛЕЛІЗМУ

Паралельна обробка інформації являє собою одночасне рішення двох або більшої кількості частин однієї й тієї ж програми двома чи більшою кількістю ЕОМ (процесорними елементами) обчислювальної системи. Реалізують три основних способи організації паралельної обробки:

1) суміщення у часі різноманітних етапів різних задач – це мультипрограмна обробка, яка широко використовується як у однопроцесорних ЕОМ, так і в складних ОС;

2) одночасне розв'язання різноманітних задач або частин однієї задачі (можливо тільки за наявності декількох обробляючих пристроїв);

3) конвеєрна обробка інформації.

Перші два способи використовують особливості паралельних задач або потоків задач, що дозволяє здійснювати той або інший паралелізм. Перший тип паралелізму – це *природний паралелізм незалежних задач*, який полягає у тому, що в систему поступає безперервний потік не зв'язаних між собою задач. У цьому випадку розв'язання будь-якої задачі не залежить від результатів розв'язання інших задач, що дозволяє підвищити продуктивність ОС у разі використання декількох обробляючих пристроїв.

Одним з найбільш розповсюджених типів паралелізму є *паралелізм незалежних гілок*. Суть його полягає у виділенні окремих незалежних частин великої задачі (гілок програми), які можуть виконуватись паралельно окремими обробляючими пристроями незалежно один від одного. Причому обробляючі пристрої ОС функціонують в однопрограмному режимі паралельної обробки інформації. Двома незалежними гілками програми визнаються гілки програми, що відповідають таким умовам:

- ні одна з вхідних для гілки програми величин не є вихідною величиною іншої програми (відсутність функціональних зв'язків);
- для двох гілок програми не повинен здійснюватись запис у одні й ті ж комірки пам'яті (відсутність зв'язку по оперативній пам'яті);
- умови виконання однієї гілки не залежать від результатів, що отримані під час виконання іншої гілки (незалежність по управлінню);
- обидві гілки повинні виконуватись у різних блоках програми (програмна незалежність).

Виділення незалежних гілок широко використовується під час паралельної обробки в задачах матричної алгебри, лінійного програмування, спектральної обробки сигналів, прямого та зворотного перетворення Фур'є та ін.

Методи та засоби реалізації паралелізму залежать від того, на якому рівні він повинен забезпечуватись. Зазвичай розрізняють такі *рівні паралелізму*:

- ***рівень завдань –*** декілька незалежних завдань одночасно виконуються на різних процесорах, які практично не взаємодіють один з одним. Цей рівень реалізується в ОС з множиною процесорів у багатозадачному режимі.
- ***рівень програм*** – частини однієї задачі виконуються множиною процесорів. Даний рівень досягається на паралельних ОС.
- ***рівень команд*** – виконання команди розділяється на фази, а фази декількох послідовних команд можуть бути перекриті за рахунок конвеєризації. Даний рівень використовується в ОС з одним процесором.
- ***рівень бітів (арифметичний рівень)*** – біти слова обробляються один за одним. Цей процес має назву *біт-послідовна операція*. Якщо біти слова обробляються одночасно, кажуть про *біт-паралельну операцію*. Даний рівень реалізується в звичайних і суперскалярних процесорах.

Паралелізм рівня завдання можливий між незалежними завданнями або їх фазами. Основним засобом реалізації паралелізму на рівні завдань є багатопроцесорні і багатомашинні обчислювальні системи, в яких завдання розподіляються за окремими процесорами або машинами. Однак, якщо кожне завдання трактувати як сукупність незалежних задач, реалізація даного рівня можлива і в рамках однопроцесорної ОС. У цьому випадку декілька завдань можуть одночасно знаходитись в основній пам’яті ОС, за умови, що в кожний момент виконується тільки одне з них. Коли завдання, яке виконується, потребує вводу/виводу, ця операція запускається, а до її завершення інші ресурси ОС передаються другому завданню. По завершенні вводу/виводу ресурси ОС повертаються до завдання, яке ініціювало цю операцію. В цьому випадку паралелізм забезпечується за рахунок того, що центральний процесор і система вводу/виводу функціонують одночасно і обслуговують різні завдання.

Паралелізм виникає також, коли у незалежних завдань, які виконуються в ОС, є декілька фаз, наприклад обчислення, запис у графічний буфер, системні виклики. За те, як різні завдання впорядковуються і витрачають загальні ресурси відповідає операційна система.

Паралелізм рівня програм. Про паралелізм на рівні програми можна говорити у двох випадках. По перше, коли в програмі можна виділити незалежні ділянки, які допустимо виконувати паралельно. Другий тип паралелізму програм можливий у межах окремого програмного циклу, якщо в ньому окремі ітерації не залежать один від одного. Програмний паралелізм можна реалізувати за рахунок великої кількості процесорів або множини функціональних блоків.

Загальна форма паралелізму на рівні програм організується розбиттям даних, які програмуються на підмножини. Цей розподіл називають *декомпозицією області* (domain decomposition), а паралелізм, який при цьому виникає, має назву *паралелізму даних*. Підмножини даних призначаються різним обчислювальним процесам, і цей процес має назву *розподілення даних* (data distribution). Процесори виділяються певним процесам або за ініціативою програми, або в процесі роботи операційною системою. На кожному процесорі може виконуватись більше одного процесу.

Паралелізм рівня команд. Паралелізм на рівні команд має місце, коли обробка декількох команд або виконання різних етапів однієї і тієї ж команди може перекриватись у часі. Розробники обчислювальної техніки ще здавна зверталися до методів, відомих під загальною назвою «поєднання операцій», при якому апаратура ОМ у будь-який момент часу виконує одночасно більше однієї операції. Цей загальний принцип включає два поняття: *паралелізм* і *конвеєризацію*. Хоча у них багато загального і їх часто важко розрізняти на практиці, ці терміни відображають два принципово різних підходи.

У першому варіанті поєднання операцій досягається за рахунок того, що в складі обчислювальної системи окремі пристрої присутні в декількох копіях. Так, до складу процесора може входити декілька АЛП, і висока продуктивність забезпечується за рахунок одночасної роботи всіх цих АЛП. Другий підхід був описаний у розділі 4.

Під час організації паралельного обчислювального процесу виникає задача вибору побудови розкладу.

Розклад паралельних обчислювальних процесів визначає порядок виконання програми в ОС, включаючи розподіл частин програми по обробляючих пристроях (процесорах, ОМ), і служить основою алгоритмів планувальника операційної системи та різноманітних управляючих програм. Як критерії оптимальних розкладів для паралельної програми можна назвати:

- мінімізацію часу виконання програми;
- мінімізацію кількості потрібних пристроїв обробки;
- мінімізацію середнього часу закінчення виконання завдань;
- максимізацію завантаження пристроїв ОС;
- мінімізацію часу простоювання пристроїв.

Найчастіше використовують перший критерій.

6.2. КЛАСИФІКАЦІЯ АРХІТЕКТУР КОМП'ЮТЕРНИХ СИСТЕМ

Існує декілька класифікацій архітектур комп'ютерних систем. Найбільш вдалою є *класифікація М. Флінна*.

Архітектурні особливості комп'ютерних систем описано з погляду потоку команд (інструкцій) та потоку даних. Такий підхід дає змогу відносити архітектури комп'ютерів до одного з чотирьох класів (табл. 6.1, рис. 6.1).

Класифікація здійснена з погляду не структури, а того, як у комп'ютері його машинні команди взаємодіють з даними.

Таблиця 6.1

Класифікація архітектур комп'ютерних систем

Потік команд	**Одиничний потік даних (ОД)**	**Множинний потік даних (МД)**
Одиничний (ОК)	**ОКОД (SISD)** (однопроцесорні комп'ютери)	**ОКМД (SIMD)** (комп'ютери з паралельними або асоціативними процесорами)
Множинний (МК)	**МКОД (MISD)** (конвеєрні магістральні комп'ютери)	**МКМД (MIMD)** (багатопроцесорні або багатомашинні комплекси)

SISD (Single Instruction Stream/Single Data Stream) - одиничний потік команд і одиничний потік даних. Представник цього класу – класичний фон-нейманівський комп'ютер. Команди обробляються послідовно і кожна команда ініціює одну операцію з одним потоком даних. До цього класу комп'ютерів можна віднести також конвеєрні комп'ютери. Деякі спеціалісти вважають, що до SISD-систем можна віднести і векторно-конвеєрні ОС, якщо розглядати вектор як неподільний елемент даних для відповідної команди.

MISD (Multiple Instruction Stream/Single Data Stream) **–** множинний потік команд і одиничний потік даних. В архітектурі присутня множина процесорів, які обробляють один і той же потік даних. Ряд дослідників відносять до даного класу конвеєрні системи. Прийнято вважати, що доки даний клас незадіяний, однак може бути корисним для розробки принципово нових концепцій побудови обчислювальних систем [25].

SIMD (Single Instruction Stream/Multiple Data Stream) **–** одиничний потік команд і множинний потік даних. Ця архітектура дозволяє виконати одну арифметичну операцію відразу над багатьма даними – елементами вектору. Представниками цього класу є системи з матрицею процесорів, де один управляючий пристрій контролює множину процесорних елементів. Усі процесорні елементи отримують від пристрою управління однакову команду і виконують її над власними локальними даними. В цей клас можуть бути включеними і векторно-конвеєрні ОС, якщо кожний елемент вектора розглядати як окремий елемент даних.

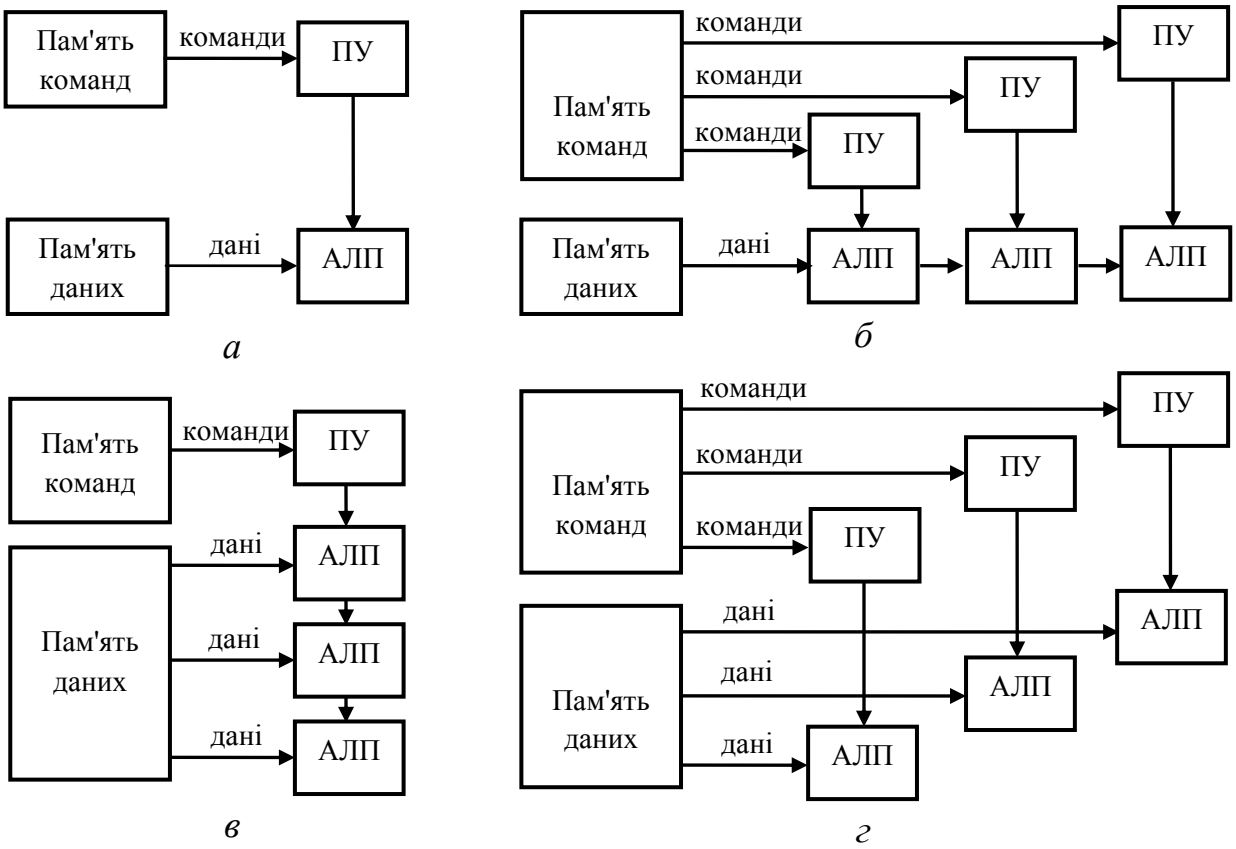


Рис. 6.1. Архітектура комп'ютерних систем за Флінном:
а - SISD; *б* - MISD; *в* - SIMD; *г* - MIMD

MIMD (Multiple Instruction Stream/Multiple Data Stream) – множинний потік команд і множинний потік даних. До цього класу відносяться системи з множиною пристроїв обробки команд, які об'єднані в єдиний комплекс і кожний працює з власним потоком команд. Цей клас надзвичайно широкий, бо включає в себе різного роду мультипроцесорні системи.

Класифікація Флінна надто загальна, вона має деякі недоліки, наприклад відносить усі паралельні комп'ютери, крім мультипроцесорних, до одного класу і не вказує ніякої відмінності між конвеєрними комп'ютерами і матрицею МП.

Використовуються й інші класифікації архітектур, зокрема систематика Ф. Шара, структурна систематика Р. Хокні та К. Джессхоупа.

Структурна систематика Р. Хокні та К. Джессхоупа

На першому рівні всі обчислювальні системи поділяються за принципом множинності (кількості) на однокомп'ютерні та багатокомп'ютерні. Обчислювальні системи з одним комп'ютером, у свою чергу, поділяються на ОМ з одним конвеєрним МП та з багатьма МП.

Перші з них є традиційними послідовними комп'ютерами, а другі утворюють клас паралельних комп'ютерів, які поділяються на конвеєрні, неконвеєрні та мікропроцесорні матриці.

Прикладом однієї з перших неконвеєрних обчислювальних машин з паралелізмом є комп'ютер CDC – 6600, побудований на основі декількох скалярних процесорів.

Конвеєрні ЕОМ поділяються на такі, що виконують тільки скалярні команди, наприклад комп'ютери CDC – 7800, FPC AP-120B, і такі, що виконують векторні команди. Комп'ютери, що використовують векторні команди, поділяють, у свою чергу, на комп'ютери із спеціалізованим конвеєром, наприклад CRAY-1, та з універсальним конвеєром – комп'ютер CYBER 205.

Комп'ютери з класу машин з матрицею процесорів поділяють за зв'язаністю процесорів у матриці, розрядністю тощо. Першими машинами такого типу були ILLIAC-IV, BSP, STA-RAN, ICL DAP, OMEN та ін.

6.3. ОБЧИСЛЮВАЛЬНІ СИСТЕМИ КЛАСУ SIMD (ОКМД)

SIMD-системи були першими обчислювальними системами, що складаються з великого числа процесорів, і серед перших систем, де була досягнута продуктивність порядку GFLOPS. Згідно з класифікацією Флінна, до класу SIMD відносяться ОС, де множина елементів даних піддається паралельній, але однотипній обробці.

На рис. 6.2 показані приблизні характеристики продуктивності деяких типів обчислювальних систем класу SIMD.

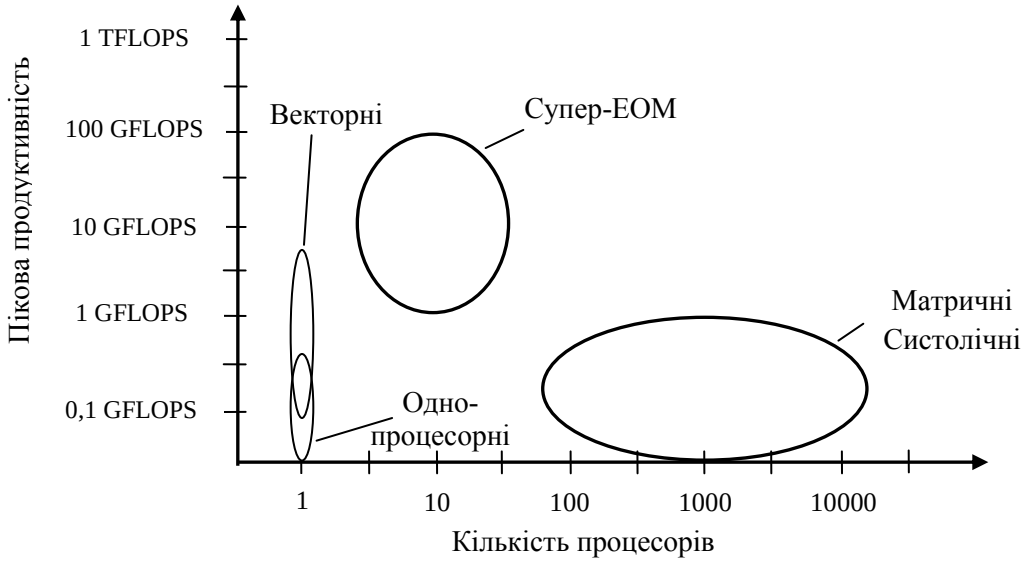


Рис. 6.2. Продуктивність SIMD-систем як функція їх типу та кількості процесорів

SIMD-системи багато в чому схожі на класичні фон-нейманівські ОМ: у них також є один пристрій управління, який забезпечує послідовне виконання команд програми. Відмінність стосується стадії виконання, коли загальна команда транслюється великій кількості процесорів (у простому випадку – АЛП), кожен з яких обробляє свої дані.

Раніше вже наголошувалася нечіткість класифікації Флінна, через що різні типи ОС можуть бути віднесені до того або іншого класу. Проте в теперішній час прийнято вважати, що клас SIMD складають векторні (векторно-конвеєрні), матричні, асоціативні, систолічні і VLIW-обчислювальні системи.

6.3.1. Векторні і векторно–конвеєрні комп'ютерні системи

Під час розв'язання на комп'ютері науково-технічних та інших задач часто зустрічається необхідність визначення значень однієї і тієї ж функції для групи даних, розташованих у комірках пам'яті (регістрах) з упорядкованими адресами (номерами). Такі набори даних називаються векторами.

Структурна схема векторного процесора показана на рис. 6.3.

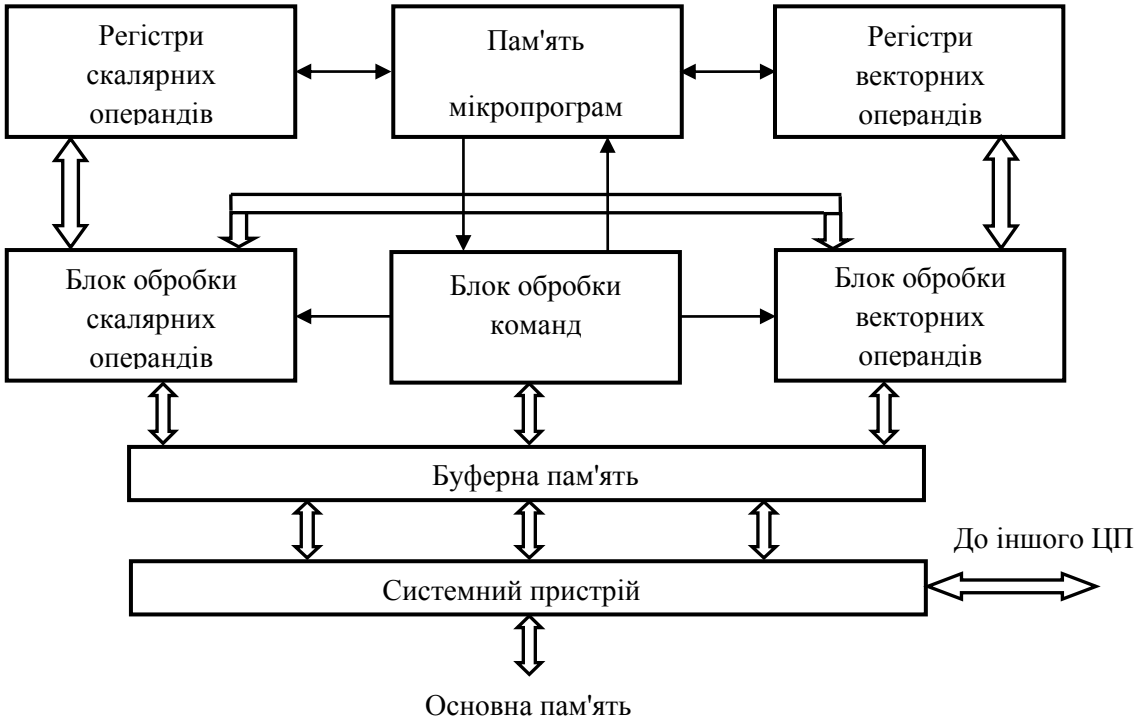


Рис. 6.3. Структурна схема векторного процесора

З метою підвищення продуктивності комп'ютера до складу його процесора разом зі скалярними засобами включають засоби *векторної обробки* даних або вводять спеціалізований векторний співпроцесор. Векторні засоби включають векторні команди, векторні регістри та іншу апаратуру для реалізації цих команд.

Засоби векторної обробки дозволяють за допомогою єдиної команди виконувати дії над усіма елементами масивів. Кожний елемент такого масиву (вектор) обробляється незалежно від інших елементів на конвеєрному операційному пристрої (ОП), який за рахунок спеціальної структурної організації може приймати в кожному такті пару елементів векторів-операндів і через деякий час, який називається *стартом конвеєра*, видавати відповідний елемент вектора-результату. Оскільки пари елементів векторів-операндів

надходять у конвеєрний ОП в кожному такті, то через час, який визначає довжину часу старту конвеєра після запуску векторної операції, з виходу ОП кожного такту будуть видаватись елементи вектора-результату.

До апаратних векторних засобів відносяться:

- арифметичні конвеєри (кожний такий конвеєр може містити декілька незалежних спеціалізованих конвеєрних арифметичних пристроїв);
- векторні регістри, які зберігають векторну інформацію, що використовується для обробки в арифметичних конвеєрах;
- конвеєри завантаження-запису, які виконують обмін інформацією між векторними регістрами і оперативною пам'яттю ЕОМ.

Векторні засоби обробки інформації дозволяють збільшити продуктивність ЕОМ на кілька порядків. Збільшення продуктивності можна оцінити за допомогою коефіцієнта підвищення продуктивності

$$P = \frac{t_{ср}\left(+k_{в}\right)\left(m-1\right)}{\left(-k_{в}\right)t_{ср}+k_{в}\left(t_{ст}+mt/DC\right)},$$

де $t_{ср}$ – середній час виконання скалярних (не векторних) операцій;

$k_{в}$ – коефіцієнт векторизації задачі, який дорівнює відношенню числа векторних операцій до загального числа операцій, що виконуються;

$t_{ст}$ – середній час старту конвеєра;

t – довжина машинного такту;

D – число арифметичних конвеєрів;

C – число векторних команд в одному арифметичному конвеєрі, що виконуються паралельно;

m – кількість розрядів.

З приведеного співвідношення видно, що для збільшення продуктивності ЕОМ необхідно дотримуватись таких вимог:

- використовувати арифметичні конвеєри з мінімально можливою довжиною такту;
- збільшувати число арифметичних конвеєрів;
- збільшувати число векторних команд, що паралельно опрацьовуються в одному арифметичному конвеєрі.

Необхідна умова використання двох останніх вимог – збільшення пропускної здатності під час передачі інформації між арифметичними конвеєрами і векторними регістрами, а також підвищення продуктивності конвеєрів завантаження/запису під час операції обміну між векторними регістрами і оперативною пам'яттю.

У ході розв'язання реальних задач на продуктивність ЕОМ істотно впливають:

• швидкість обробки скалярних величин і параметри векторного пристрою. Залежності збільшення продуктивності ЕОМ від коефіцієнта векторизації при зміні параметрів векторних засобів обробки приведені на рис. 6.4;

• об'єм векторних регістрів. У разі недостатнього об'єму векторні регістри не можуть згладити нерівномірність інформаційного потоку, що обробляється, невизначено збільшується інтенсивність обміну між векторними арифметичними конвеєрами і оперативною пам'яттю.

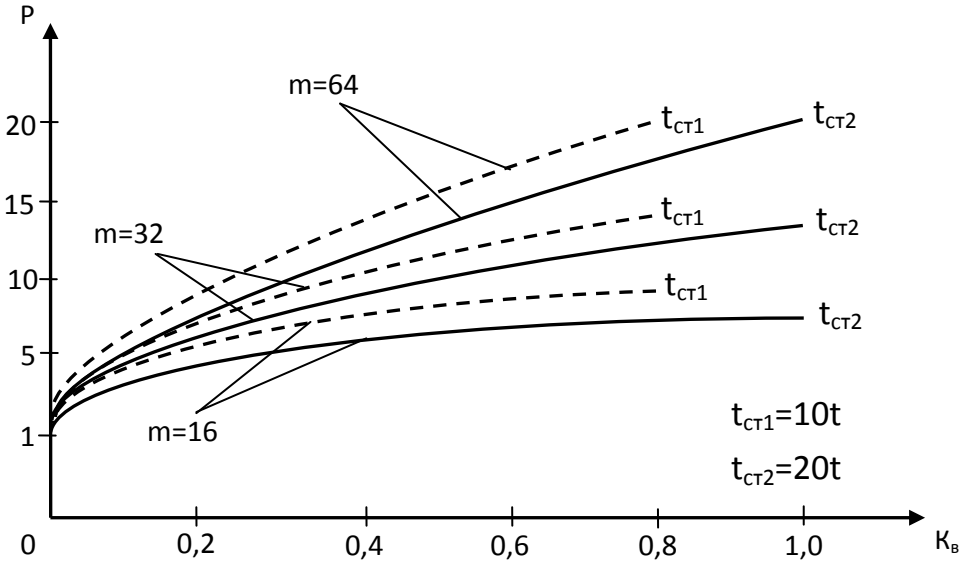


Рис. 6.4. Залежності P($k_в$, m)

Векторні команди дозволяють однією командою дати розпорядження на виконання деякої операції (векторної операції) над елементами вектора, наприклад, поелементне додавання векторів та організовують і управління обчислювальним циклом.

У машині, яка має векторні команди, виконуються також звичайні команди над одиничними даними (скалярні команди).

Наявність векторних команд сприяє підвищенню швидкодії процесора за рахунок зменшення витрат часу на організацію обчислювального циклу.

Звичайно, з метою досягнення підвищеної швидкодії виконання самих векторних операцій ставиться на конвеєр, до того ж може бути кілька арифметичних конвеєрів (ліній), а окремі пристрої конвеєрної лінії можуть, у свою чергу, містити конвеєри для виконання покладених на них підфункцій.

Векторні команди (звичайно їх порівняно мало) мають ряд особливостей. Їхній код операції не тільки задає операцію, яка підлягає виконанню, але і визначає необхідну конфігурацію активних частин конвеєра для даної операції. Команда вказує початок вектора (векторів) в пам'яті (або блоці регістрів), довжину вектора, розмір і тип елементів вектора (ціле число, число з плаваючою комою і т.д), визначає місце знаходження вектора-результату.

Прикладами конвеєрно-векторних процесорів можуть служити спеціалізовані на виконання векторних і матричних операцій співпроцесори,

при цьому продуктивність машини під час виконання векторних і матричних операцій збільшується до 30 Мфлоп/с.

Конвеєрно-векторні ОС в теперішній час є одним з основних напрямків у разі утворення супер-ЕОМ для широкого кола задач. До ОС такого типу відносяться найшвидкодіючі супер-ЕОМ сімейства CRAY (США). На рис. 6.5 подана спрощена структура конвеєрно-векторного процесора супер-ЕОМ CRAY-1.

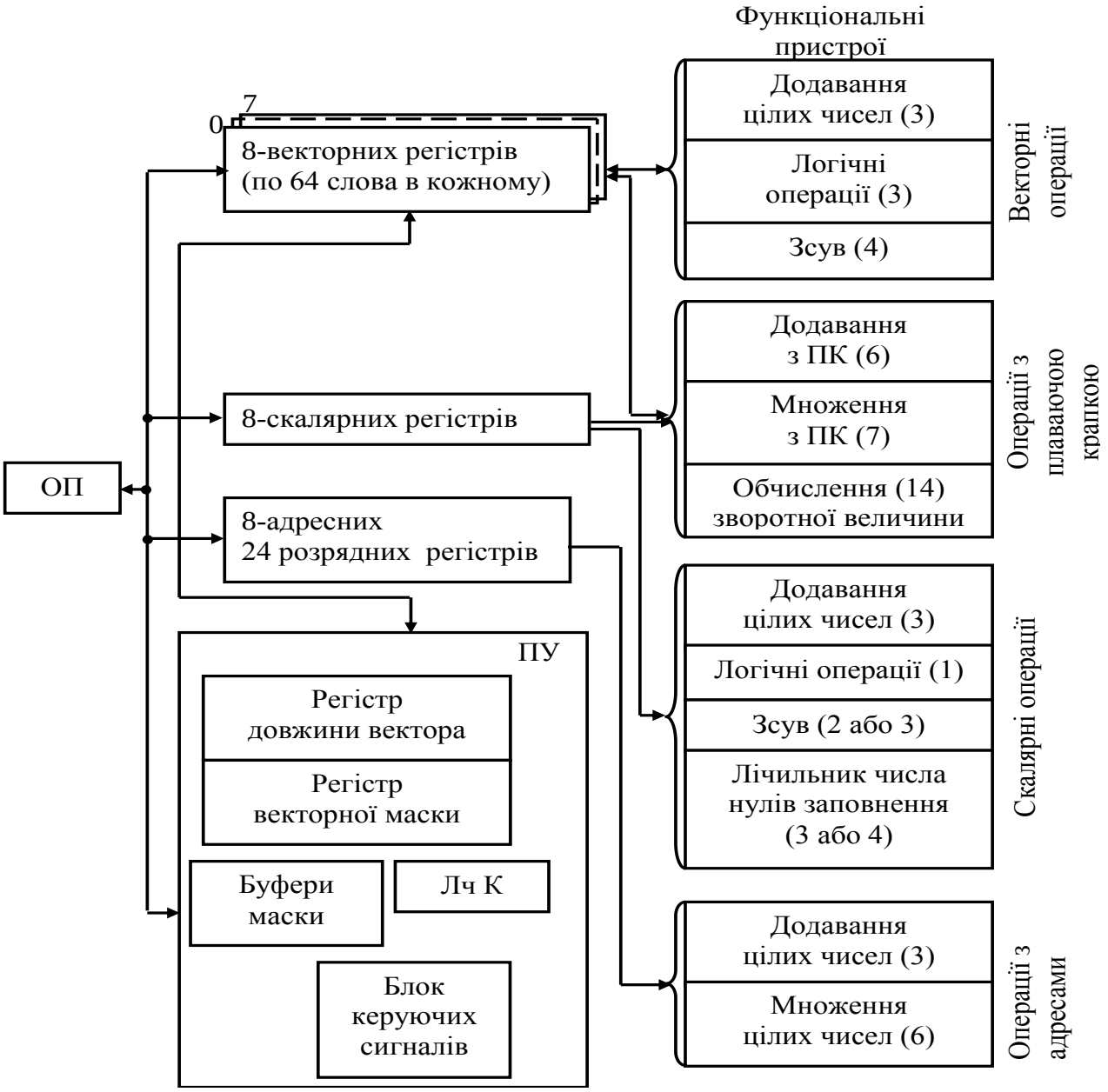


Рис. 6.5. Структура конвеєрно-векторної супер-ЕОМ (CRAY – 1)

Оскільки програми розв'язання науково-технічних задач, як правило, вимагають виконання як векторних, так і скалярних операцій, процесор має командні й апаратурні засоби для операцій двох типів. При цьому застосовані спеціальні заходи на зменшення часу виконання скалярних операцій, щоб останні вагомо не знижували загальну продуктивність системи.

Команди мають формат 16 або 32 розряди. Шістнадцятирозрядні команди основних, у тому числі векторних операцій мають 7-розрядне поле коду операції і три 3-розрядних поля для номера регістрів операндів і результату. В 32-розрядній команді під адресу основної пам’яті або безпосередній операнд відведено 22 розряди.

В системі реалізовані конвеєр команд і багаточисельні конвеєри для різних арифметичних і логічних операцій під час скалярного і векторного опрацювання 64-розрядних слів з плаваючою комою (порядок - 15, мантиса – 49 розрядів), 64 – або 24 – розрядних слів з фіксованою комою, 22 – розрядних адрес.

Ці операції виконуються 12 конвеєризованими функціональними пристроями, які можуть працювати паралельно в часі, виконуючи різні операції. При цьому можливе утворення послідовних з'єднань конвеєрів, у тому числі так зване зачеплення векторних операцій, за яких отримані в ході виконання векторної команди результати через регістри (без використання пам'яті) подаються для використання наступною векторною командою. Конвеєри мають небагато робочих позицій (кількість вказана в дужках), з тим щоб зменшити втрати продуктивності, які пов'язані з періодом «розгону» конвеєра. Такт роботи конвеєра 12,5 нс.

Для підтримки конвеєрної і векторної обробки необхідні швидкі регістрові засоби, які звільняють від необхідності звернення до пам'яті під час виконання векторних операцій і забезпечують високий темп завантаження конвеєрів і інформаційні зв'язки між ними.

Процесор CRAY містить швидкі регістри (час звертання 6 нс): 8 векторних V – регістрів, кожний з яких може зберігати 64 64-розрядних слова; 8 24-розрядних адресних регістрів; 8 скалярних 64-розрядних S-регістрів.

Управляючий пристрій крім традиційних блоків містить чотири буфери команд (на 64 командних слова кожний) і спеціальні регістри: регістр довжини вектора і регістр векторної маски. Регістр довжини вектора фіксує довжину вектора S, який обробляється (завжди S<64). Регістр векторної маски, який містить 64 розряди, значенням свого і-го розряду визначає, із якого із регістрів, які беруть участь в операції, і-й елемент видається в регістр результату. По-іншому використовує регістр векторної маски команда перевірки елементів вектора, яка встановлює розряди вектора маски в 1, якщо відповідні елементи вектора задовольняють задану умову. Результат цієї команди може ефективно використовуватись іншими командами у ході опрацювання даних.

У пам'яті системи, яка має ємність від 1 до 4 Мслів (слово 64 розряди + 9 контрольних), приблизно 16-кратне чергування адрес, за рахунок чого цикл звертання складає всього 50 нс.

В однопроцесорній ОС CRAY-1 досягнута продуктивність близько 80-130 Мфлоп/с. Для розгляду вибрана ця одна з перших конвеєрно-векторних супер-ЕОМ через її порівняну простоту і наочність відносно особливостей

організації комбінацій конвеєрного і векторного опрацювання даних. У більш нових і більш потужних моделях фірми CRAY використовується від 2 до 8 процесорів, і продуктивність складає від 1000 до 2500 Мфлоп/с.

6.3.2. Матричні комп'ютерні системи

Призначення *матричних комп'ютерних систем* багато в чому схоже з призначенням векторних комп'ютерних систем – обробка великих масивів даних. В основі матричних систем лежить *матричний процесор*, який складається з регулярного масиву процесорних елементів (ПЕ). Організація системи такого типу, на перший погляд, достатньо проста. Вона має загальний управляючий пристрій, який генерує потік команд, та велику кількість ПЕ, які функціонують паралельно і обробляють кожний свій потік даних. Проте на практиці, щоб забезпечити достатню ефективність системи у ході розв'язання широкого кола задач, необхідно організовувати зв'язки між процесорними елементами так, щоб найбільш повно завантажити процесори роботою. Саме характер зв'язків між ПЕ і визначає різні властивості системи.

Матричний процесор інтегрує множину ідентичних функціональних блоків (ФБ), які логічно об'єднуються в матрицю і працюють у SIMD-стилі. Матриця процесорних елементів може бути конструктивно реалізована на одному кристалі або на декількох. Важливим є принцип функціонування – ФБ логічно скомпоновані у матрицю і функціонують синхронно, тобто існує тільки один потік команд для усіх блоків.

Структуру матричної комп'ютерної системи можна подати у виді, який показаний на рис 6.6.

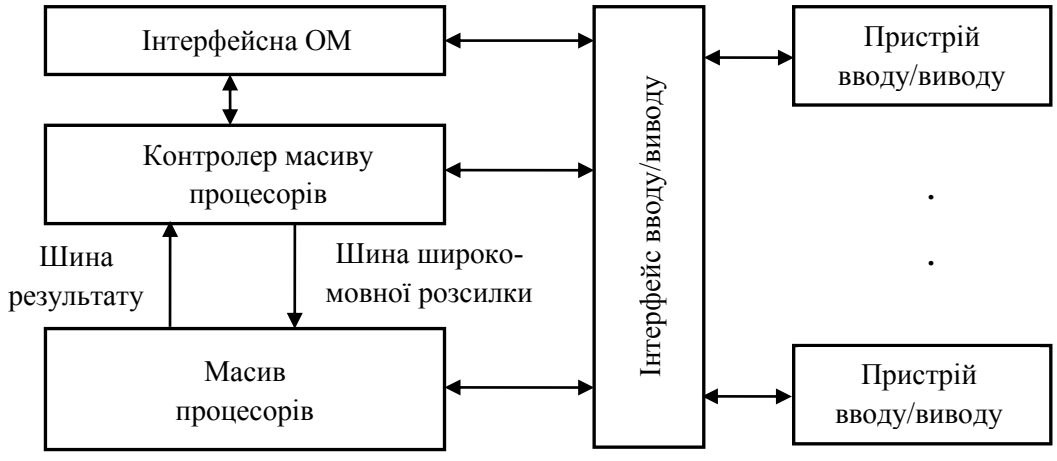


Рис. 6.6. Узагальнена модель матричної SIMD-системи

Власне паралельна обробка множинних елементів даних виконується *масивом процесорів* (МПр). Єдиний потік команд, який управляє обробкою даних у масиві процесорів, генерується *контролером масиву процесорів* (КМПр). КМПр виконує послідовний програмний код, реалізує операції

умовного і безумовного переходів, транслює в МПр команди, дані та сигнали управління. Команди обробляються процесорами в режимі жорсткої синхронізації. Сигнали управління використовуються для синхронізації команд і пересилок, а також для управління процесом обчислень, зокрема визначають, які процесори масиву повинні виконувати операцію, а які – ні. Команди, дані і сигнали управління передаються із КМПр у масив процесорів по *шині широкомовної розсилки*. Оскільки виконання операцій умовного переходу залежить від результатів обчислень, результати обробки даних у масиві транслюються в КМПр по *шині результату*.

Для забезпечення користувача зручним інтерфейсом під час створення та налаштування програм у склад подібних комп'ютерних систем звичайно включають *інтерфейсну обчислювальну машину* (ІОМ). Як таку обчислювальну машину використовують універсальну обчислювальну машину, на яку додатково покладається задача завантаження програм і даних в КМПр. Інтерфейсна обчислювальна машина функціонує під управлінням операційної системи, частіше це ОС UNIX. На ІОМ користувачі готують, компілюють і налаштовують власні програми. У процесі виконання програми спочатку завантажуються із інтерфейсної обчислювальної машини в контролер управління масивом процесорів, який виконує програму і розподіляє команди і дані по процесорних елементах масиву.

Крім того, завантаження програм і даних в КМПр може виконуватись і безпосередньо з *пристроїв вводу/виводу*, наприклад з магнітних дисків. Після завантаження КМПр приступає до виконання програми, транслюючи в МПр по широкомовній шині відповідні SIMD-команди.

Розглядаючи масив процесорів, слід враховувати, що для зберігання множинних наборів даних у ньому, крім множини процесорів, повинна бути присутньою і множина модулів пам'яті. Крім того, в масиві повинна бути реалізована мережа взаємозв'язків як між процесорами, так і між процесорами і модулями пам'яті. Таким чином, під терміном *масив процесорів* розуміють блок, який складається з процесорів, модулів пам'яті і мережі з'єднань.

У матричних SIMD-системах розповсюдження отримали два основних типи архітектурної організації масиву процесорних елементів.

У першому варіанті, який відомий як архітектура типу «*процесорний елемент − процесорний елемент*», *N* процесорних елементів (ПЕ) зв'язані між собою мережею з'єднань. Кожний ПЕ – це процесор з локальною пам'яттю. Процесорні елементи виконують команди, які отримують від КМПр по шині широкомовної розсилки, та обробляють дані як ті, що зберігаються у їх локальній пам'яті, так і ті, що потрапляють з КМПр. Обмін даними між процесорними елементами здійснюється по *мережі з'єднань*, в той час як *шина вводу/виводу* служить для обміну інформацією між ПЕ і пристроями вводу/виводу. Для трансляції результатів з окремих ПЕ в контролер масиву

процесорів служить *шина результату*. В багатьох алгоритмах дії по пересиланню інформації в більшості локальні, тобто виконуються між найближчими сусідами, тому дана архітектура, де кожний ПЕ зв'язаний тільки з сусідніми, достатньо ефективна.

Другий вид архітектури – «*процесор – пам'ять*». У такій архітектурі двонаправлена мережа з'єднань зв'язує *N* процесорів з *M* модулями пам'яті. КМП управляє процесорами через широкомовну шину. Обмін даними між процесорами здійснюється як через мережу, так і через модулі пам'яті. Пересилка даних між модулями пам'яті та пристроями вводу/виводу забезпечується шиною вводу/ виводу. Для передачі даних з певного модуля пам'яті в КМП служить шина результату.

Додаткову гнучкість під час роботи з матричною обчислювальною системою забезпечує механізм *маскування*, який дозволяє залучати до участі в операціях тільки певну підмножину процесорів з тих, які входять у масив. Маскування реалізується як на стадії компіляції, так і на етапі виконання, при цьому ті процесори, які виключаються шляхом встановлення в нуль відповідних бітів маски, протягом виконання команди простоюють.

6.3.3. Комп'ютерні системи з систолічною структурою

У фон-нейманівських комп'ютерах дані, які зчитуються з пам'яті, однократно обробляються в процесорному елементі, після чого знову повертаються в пам'ять (рис. 6.7, *а*). Автори ідеї систолічної матриці Кунг і Лейзерсон запропонували організувати обчислення так, щоб дані на своєму шляху від зчитування з пам'яті до повернення назад пропускалися через якамога більшу кількість ПЕ (рис. 6.7, *б*).

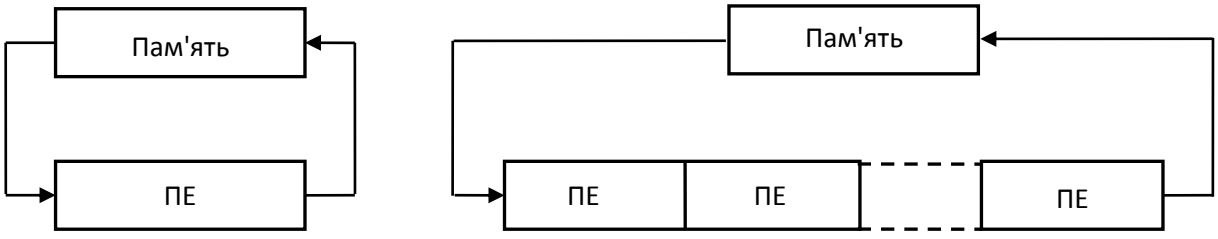


Рис. 6.7. Обробка даних у комп'ютерних системах:
а – фон-нейманівського типу; *б* – систолічної структури

Якщо порівняти положення пам'яті в комп'ютерній системі зі структурою живого організму, то за аналогією їй можна відвести роль серця, множині ПЕ – роль тканин, а потік даних розглядати як кров, що циркулює. Звідси і виходить назва *систолічна матриця* (систола – скорочення передсердя і шлуночків серця, при якому кров нагнітається в артерії). Систолічні структури ефективні під час виконання матричних обчислень, обробки сигналів, сортування даних і так далі.

Таким чином, *систолічна структура* – це однорідне обчислювальне середовище з процесорних елементів, яке суміщає в собі властивості конвеєрної та матричної обробки і володіє такими особливостями:

- обчислювальний процес у систолічних структурах являє собою неперервну і регулярну передачу даних від одного ПЕ до іншого без запам'ятовування проміжних результатів обчислення;
- кожний елемент вхідних даних вибирається з пам'яті однократно та використовується стільки разів, скільки необхідно за алгоритмом, введення даних здійснюється в крайні ПЕ матриці;
- ПЕ, які створюють систолічну структуру, однотипні і кожний з них може бути менш універсальним, ніж процесори звичайних багатопроцесорних систем;
- потоки даних та управляючих сигналів володіють регулярністю, що дозволяє об'єднувати ПЕ локальними зв'язками мінімальної довжини;
- алгоритми функціонування дозволяють суміщати паралелізм з конвеєрною обробкою даних;
- продуктивність матриці можна покращувати за рахунок додавання до неї певного числа ПЕ, причому коефіцієнт підвищення продуктивності при цьому є лінійним.

У теперішній час продуктивність систолічних процесорів перевищує 1000 млрд операцій /с.

Аналіз різних типів систолічних структур та тенденцій їх розвитку дозволяє класифікувати ці структури по декільком ознакам.

За ***ступенем гнучкості*** систолічні структури можуть бути згруповані так:

- спеціалізовані;
- алгоритмічно орієнтовані;
- програмовані.

Спеціалізовані структури орієнтовані на виконання певного алгоритму. Ця орієнтація відображається не тільки в конкретній геометрії систолічної структури, статичності зв'язків між ПЕ та кількості ПЕ, але і в виборі типу операції, що виконується усіма ПЕ. Прикладами є структури, які орієнтовані на рекурсивну фільтрацію, швидке перетворення Фур'є для заданої кількості точок, матричні перетворення.

Алгоритмічно орієнтовані системи володіють можливістю програмування або конфігурації зв'язків у систолічній матриці або самих ПЕ. Можливість програмування дозволяє виконувати на таких структурах деяку множину алгоритмів, які зводяться до однотипних операцій над векторами, матрицями та іншими числовими множинами.

У *програмованих систолічних структурах* є можливість програмування як самих ПЕ, так і конфігурації зв'язків між ними. При цьому ПЕ можуть володіти локальною пам'яттю програм. Команди, які зберігаються в пам'яті таких ПЕ, можуть змінювати і напрямок передачі операндів.

За ***розрядністю процесорних елементів*** систолічні структури діляться так:

- одноразрядні;
- багаторозрядні.

В одноразрядних матрицях ПЕ в кожний момент часу виконує операцію над одним двійковим розрядом, а в багаторозрядних – над словами фіксованої довжини.

За ***характером локально-просторових зв'язків*** систолічні структури бувають:

- одновимірні;
- двовимірні;
- тривимірні.

Вибір структури залежить від виду інформації, що обробляється. Одновимірні структури застосовують при обробці векторів, двовимірні – матриць, тривимірні – множин іншого типу.

У теперішній час розроблені систолічні матриці з різною геометрією звязків: лінійні, прямокутні, гексагональні, тривимірні та ін. (рис. 6.8).

Кожна конфігурація матриці найбільш пристосована для виконання певних функцій, наприклад лінійна матриця оптимальна для реалізації фільтрів в реальному масштабі часу; гексагональна – для виконання операцій обернення матриць, а також для операцій над матрицями спеціального типу. Найбільш універсальними і розповсюдженими є матриці з лінійною структурою.

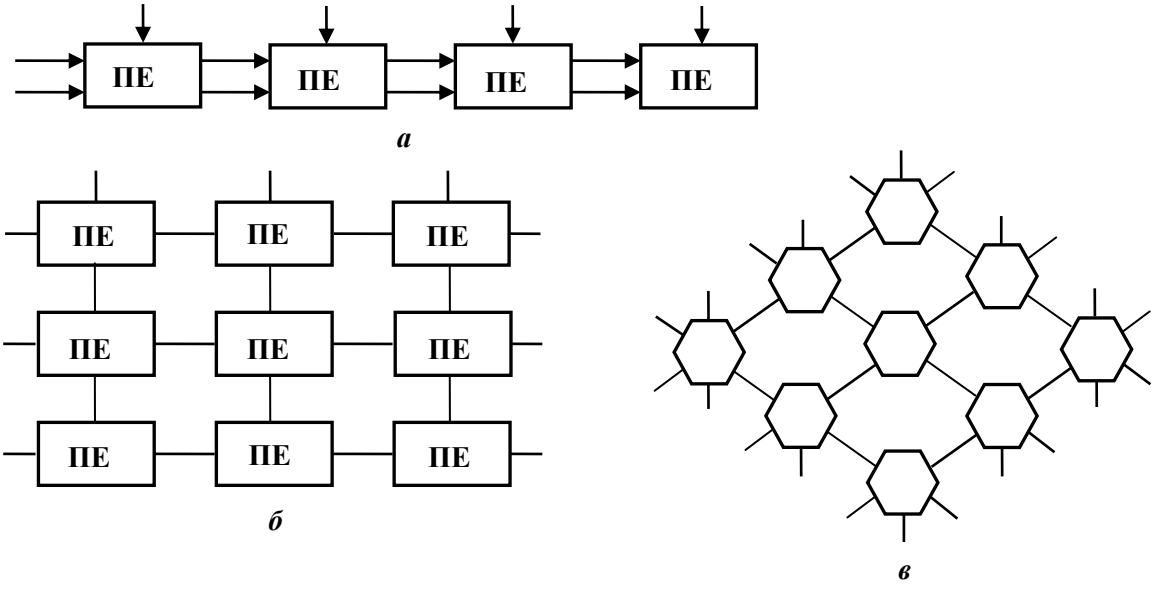


Рис. 6.8. Конфігурація систолічних матриць:
а – лінійна; *б* – прямокутна; *в* – гексагональна

Для розв'язання складних задач конфігурація систолічної структури може являти собою набір окремих матриць, складну мережу взаємозв'язаних матриць або оброблювану поверхню. Під *оброблюваною поверхнею* розуміють безкінечну прямокутну мережу ПЕ, де кожний ПЕ є з'єднаним зі своїми чотирма сусідами (або більшим числом ПЕ).

6.3.4. Обчислювальні системи з командними словами надвеликої довжини (VLIW)

VLIW (Very Long Instruction Word) – це набір команд, які організовані аналогічно організації горизонтальної мікрокоманди в мікропрограмному пристрої управління.

Ідея VLIW базується на тому, що задача ефективного планування паралельного виконання декількох команд покладається на «розумний» компілятор. Такий компілятор спочатку досліджує початкову програму з метою виявити всі команди, які можуть бути виконані одночасно, причому так, щоб це не приводило до виникнення конфліктів. У процесі аналізу компілятор може навіть частково імітувати виконання програми, яка розглядається. На наступному етапі компілятор намагається об'єднати такі команди в пакети, кожний з котрих розглядається як одна наддовга команда. Об'єднання декількох простих команд в одну наддовгу виконується за такими правилам:

- кількість простих команд, які об'єднуються в одну команду надвеликої довжини, дорівнює числу функціональних (виконавчих) блоків (ФБ);
- у наддовгу команду входять тільки прості команди, які виконуються різними ФБ, тобто забезпечується одночасне виконання усіх складових наддовгої команди.

Довжина наддовгої команди звичайно складає від 256 до 1024 біт. Така *метакоманда* містить декілька полів (по числу простих команд, які її утворюють), кожне з яких описує операцію для конкретного функціонального блока. На рис. 6.9 показаний можливий формат наддовгої команди та взаємозв'язок між її полями і ФБ, які реалізують окремі операції.

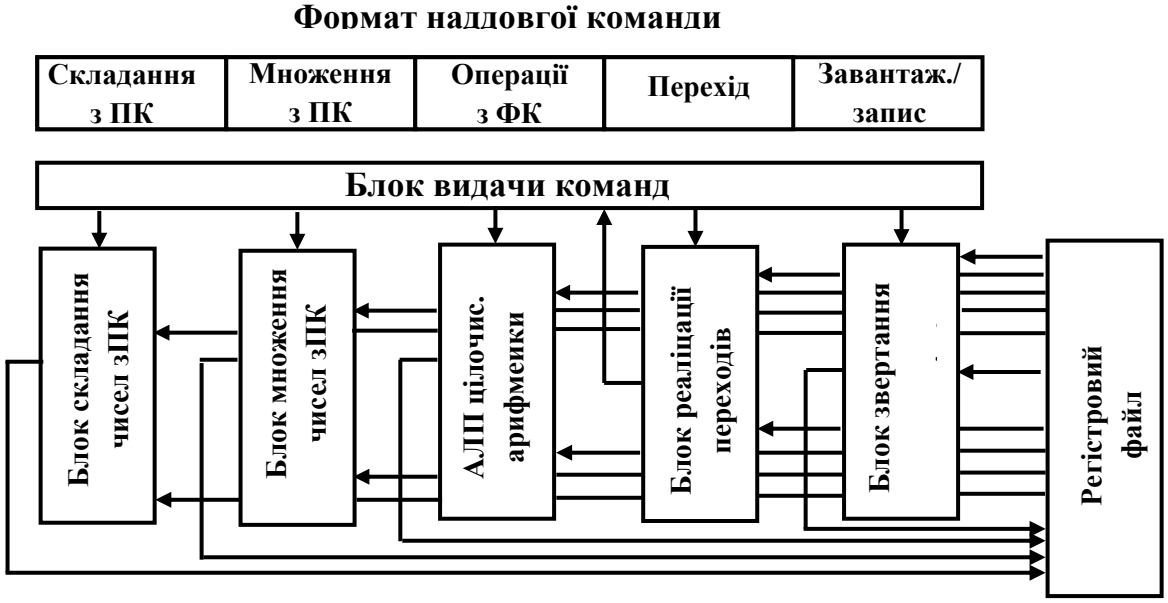


Рис. 6.9. Формат наддовгої команди та взаємозв'язок полів команди зі складовими блока виконання

Як видно з рисунка, кожне поле наддовгої команди відображається на свій функціональний блок, що дозволяє отримати максимальну віддачу від апаратури блока виконання команд.

VLIW-архітектуру можна розглядати як статичну суперскалярну архітектуру. Мається на увазі, що розпаралелювання коду здійснюється на етапі компіляції, а не динамічно в час виконання. Як уже було відмічено, у виконуваної наддовгої команди виключена можливість конфліктів, а це дозволяє значно спростити апаратуру VLIW-процесора та збільшити швидкодію.

Як прості команди, що створюють наддовгу, звичайно використовують команди RISC-типу, тому архітектуру VLIW іноді називають постRISC-архітектурою. Максимальне число полів у наддовгій команді дорівнює числу обчислювальних засобів і звичайно знаходиться в діапазоні від 3 до 20. Усі обчислювальні засоби мають доступ до даних, які зберігаються у єдиному багатопортовому регістровому файлі. Відсутність складних апаратних механізмів, які є характерними для суперскалярних процесорів (передбачення переходів, позачергове виконання і ін.), дає значний виграш у швидкодії та можливість більш ефективно використовувати площину кристалу. Переважна більшість цифрових сигнальних процесорів та мультимедійних процесорів з продуктивністю більш 1 млрд операцій/с базується на VLIW-архітектурі. Серйозна проблема VLIW-архітектури – ускладнення регістрового файла та зв'язків цього файла з обчислювальними пристроями.

6.4. КОМП'ЮТЕРНІ СИСТЕМИ КЛАСУ MIMD (МКМД)

6.4.1. Загальні відомості про обчислювальну систему класу MIMD

MIMD-системи володіють великою гнучкістю, зокрема вони можуть робити і як високопродуктивні однокористувацкі обчислювальні системи, і як багатопрограмні ОС, які паралельно виконують множину задач. Крім того, архітектура MIMD дозволяє найбільш ефективно розпорядитися усіма перевагами сучасної мікропроцесорної технології.

Приблизні значення пікової продуктивності для різних типів систем класу MIMD показані на рис. 6.10.

У MIMD-системі кожний процесорний елемент (ПЕ) виконує власну програму достатньо незалежно від інших ПЕ. У той же час ПЕ повинні якось взаємодіяти один з одним. Відмінність у способі такої взаємодії визначає умовне ділення MIMD-систем на ОС з загальною пам'яттю та системи з розподіленою пам'яттю. В *системах із загальною пам'яттю*, які характеризують як *сильно зв'язані*, є загальна пам'ять даних і команд, до якої мають доступ усі ПЕ за допомогою загальної шини або мережі з'єднань.

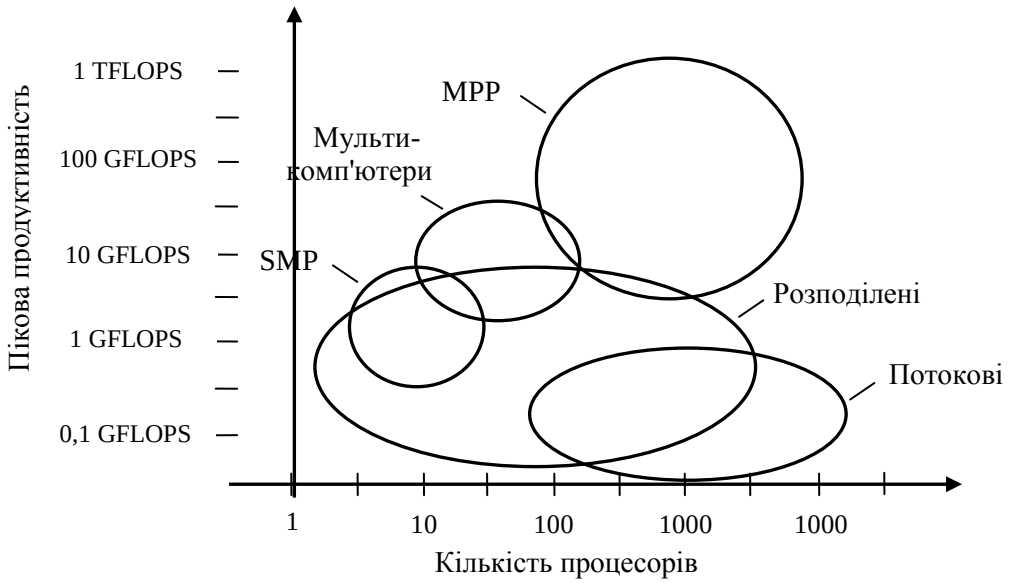


Рис. 6.10. Продуктивність MIMD-систем як функція їх типу та кількості процесорів

До цього типу, зокрема, відносяться *симетричні мультипроцесори* (SMP, Symmetric Multiprocessor) та *системи з неоднорідним доступом до пам'яті* (NUMA, Non-Uniform Memory Access).

У *системах з розподіленою пам'яттю* або *слабо зв'язаних* багатопроцесорних системах уся пам'ять розподілена між процесорними елементами, і кожний блок пам'яті доступний тільки «власному» процесору. Мережа з'єднань зв'язує процесорні елементи один з одним. Представниками цієї групи є *системи з масовим паралелізмом* (MPP, Massively Parallel Processing) та *кластерні обчислювальні системи*.

Базовою моделлю обчислень на MIMD-системі є сукупність незалежних процесів, що епізодично звертаються до сумісно використовуваних даних.

Існує багато варіантів цієї моделі. На одному кінці спектра – розподілені обчислення, у рамках яких програма ділиться на достатньо велике число паралельних задач, що складаються з множини підпрограм. На другому кінці – модель потокових обчислень, де кожна операція в програмі може розглядатись як окремий процес. Така операція очікує надходження вхідних даних (операндів), які повинні бути переданими їй іншими процесами. Після цього операція виконується, і результуюче значення передається тим процесам, які його потребують.

6.4.2. Симетричні мультипроцесорні системи (SMP)

Поняття *симетричні мультипроцесорні обчислювальні системи*, так звані *SMP-системи* (Symmetric Multiprocessor), відноситься як до архітектури обчислювальної системи, так і до поведінки операційної системи, яка відображає дану архітектурну організацію. SMP можна визначити як обчислювальну систему, що має такі характеристики:

- є два або більше процесорів порівнянної продуктивності;
- процесори сумісно використовують основну пам'ять і функціонують в єдиному віртуальному і фізичному адресному просторі;
- усі процесори зв'язані між собою за допомогою шини або за іншою схемою так, що час доступу до пам'яті будь-якого з них є однаковий;
- усі процесори розділяють доступ до пристроїв вводу/виводу або через одні й ті ж канали, або через різні канали, які забезпечують доступ до одного й того ж зовнішнього пристрою;
- усі процесори здатні виконувати однакові функції (цим пояснюється термін «симетричні»);
- будь-який з процесорів може обслуговувати зовнішні переривання;
- обчислювальна система управляється інтегрованою операційною системою, яка організовує і координує взаємодію між процесорами та програмами на рівні завдань, задач, файлів і елементів даних.

В порівнянні з однопроцесорними схемами SMP-системи мають переваги по таких показниках:

Продуктивність. Якщо задача, яка повинна бути розв'язана, підлягає розбиттю на декілька частин так, що окремі частини можуть виконуватись паралельно, то множина процесорів дає виграш у продуктивності відносно одиничного процесора того ж типу.

Готовність. У симетричному мультипроцесорі відмова одного з компонентів не приводить до відмови системи, тому що будь-який з процесорів здатний виконувати ті ж функції, що й інші.

Розширюваність. Продуктивність системи може бути збільшена додаванням додаткових процесорів.

Масштабованість. Варіюючи число процесорів у системі, можна створювати системи різної продуктивності і вартості.

На рис. 6.11 у загальному вигляді показана архітектура симетричної мультипроцесорної обчислювальної системи.

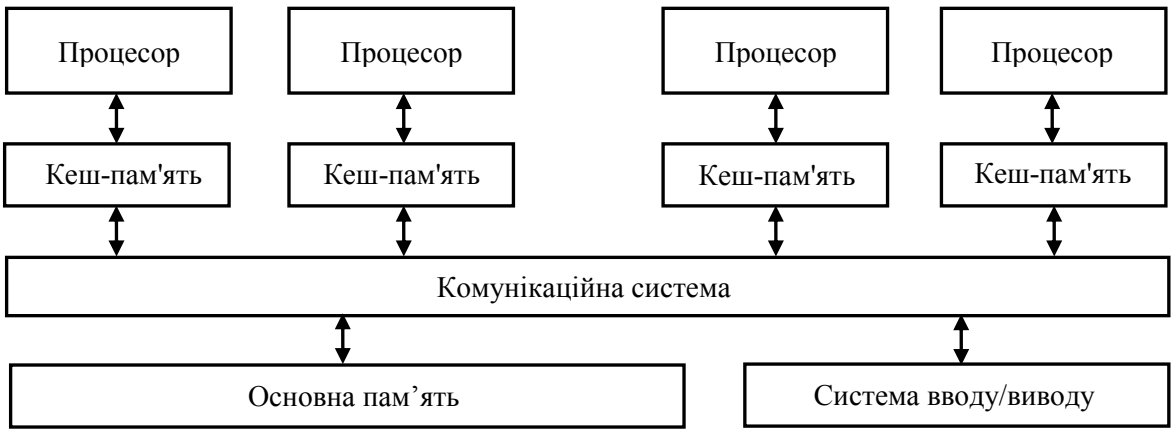


Рис. 6.11. Організація симетричної мультипроцесорної системи

Типова SMP-система містить від двох до 32 ідентичних процесорів, у ролі яких звичайно використовують дешеві RISC-процесори. В останній час намітилася тенденція оснащення SMP-систем також і CISC-процесорами, зокрема Pentium.

Кожний процесор забезпечений локальною кеш-пам'яттю, яка складається з кеш-пам’яті першого(L1) та другого (L2) рівнів. Узгодженість вмісту кеш-пам'яті всіх процесорів забезпечується апаратними засобами. В деяких SMP-системах проблема когерентності знімається за рахунок загальної кеш-пам'яті. Застосування загальної кеш-пам'яті супроводжується збільшенням вартості і зменшенням швидкодії кеш-пам'яті.

Усі процесори обчислювальної системи мають рівноправний доступ до основної пам'яті і пристроїв вводу/виводу, що розділяються. Така можливість забезпечується комунікаційною системою. Звичайно процесори взаємодіють між собою через основну пам'ять. В деяких SMP-системах передбачається також прямий обмін сигналами між процесорами.

Пам'ять системи будується по модульному принципу і організована так, що дозволяється одночасне звернення до різних її модулів (банків). В деяких конфігураціях в доповнення до ресурсів, які використовуються сумісно, кожний процесор володіє також власною локальною основною пам'яттю та каналами вводу/виводу.

6.4.3. Системи з масовою паралельною обробкою (MPP)

Основною ознакою, за якою обчислювальну систему відносять до *архітектури з масовою паралельною обробкою* (MPP - Massively Parallel Processing), служить кількість процесорів n. Строгої межі не існує, але при $n \geq 128$ вважається, що це вже MPP, а при $n \leq 32$ – ще ні.

Головні особливості, за якими обчислювальну систему відносять до класу MPP, можна сформулювати таким чином:

- стандартні мікропроцесори;
- фізично розподілена пам'ять;
- мережа з'єднань з високою пропускною здатністю і малими затримками;
- добра масштабованість (можливість варіювання числом процесорів (до тисяч процесорів));
- асинхронна MIMD-система з пересилкою повідомлень;
- програма являє собою множину процесів, які мають окремі адресні простори.

Узагальнена структура MPP-системи показана на рис. 6.12.

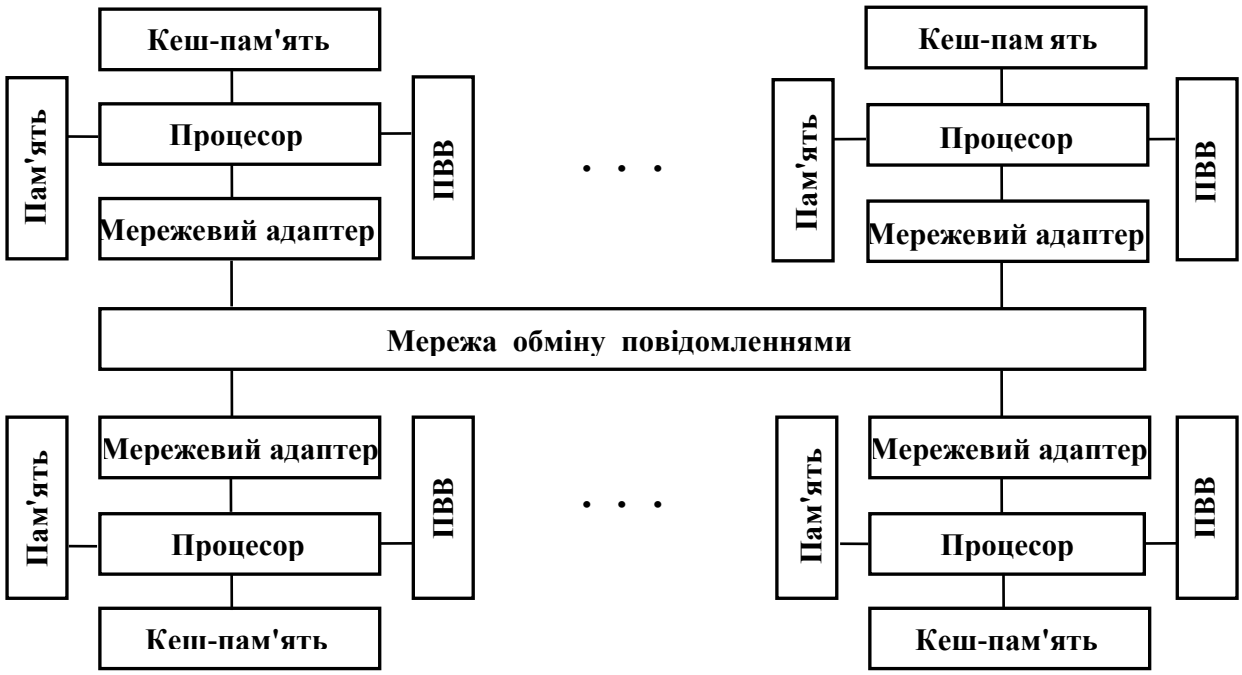


Рис. 6.12. Структура обчислювальної системи з масовою паралельною обробкою

Основні причини появи систем з масовою паралельною обробкою – це, по-перше, необхідність побудови ОС з дуже великою продуктивністю і, по-друге, зменшення її вартості.

Характерна риса MPP-систем – наявність єдиного управляючого пристрою (процесора), що розподіляє завдання між множиною підлеглих пристроїв. Схема взаємодії в загальних рисах достатньо проста:

- центральний управляючий пристрій формує чергу завдань, кожному з яких призначається деякий рівень пріоритету;
- по мірі звільнення підлеглих пристроїв їм передаються завдання з черги;
- підлеглі пристрої оповіщають центральний процесор про хід виконання завдань, зокрема про завершення виконання або про потребу в додаткових ресурсах;
- у центрального пристрою є засоби для контролю роботи підлеглих процесорів, зокрема для виявлення нештатних ситуацій, переривання виконання завдань у випадку появи більш пріоритетної задачі та ін.

У деякому наближенні є сенс вважати, що на центральному процесорі виконується ядро операційної системи (планувальник завдань), а на підлеглих – додатки. Підлеглість між процесорами може бути реалізована як на апаратному, так і на програмному рівні.

Завдяки властивості масштабованості, MPP-системи на сьогодні є лідерами по досягнутій продуктивності. З іншого боку, розпаралелювання в MPP-системах є складною задачею. Ефективність розпаралелювання у багатьох

випадках сильно залежить від деталей архітектури МРР-системи. Для синхронізації паралельно виконуваних процесів необхідний обмін повідомленнями, які повинні доходити з будь-якого вузла системи у будь-який інший вузол. Час передачі інформації від вузла до вузла залежить від стартової затримки та швидкості передачі. Продуктивність процесорів набагато більше пропускної здатності каналів зв'язку, тому інфраструктура каналів зв'язку в МРР-системах є найбільш проблемною.

Слабким місцем МРР було і є центральний управляючий пристрій (ЦУП) – при виході його зі строю вся система стає непрацеспроможною. Підвищення надійності ЦУП здійснюється за рахунок спрощення або дублювання його апаратури.

Сфера застосування ОС з масовим паралелізмом постійно розширюється. Різні системи цього класу експлуатуються у багатьох провідних суперкомп'ютерних центрах світу.

6.4.4. Кластерні обчислювальні системи

Один із самих найсучасніших напрямів у області створення обчислювальних систем – це *кластеризація*. За продуктивністю та коефіцієнтом готовності кластеризація являє собою альтернативу симетричним мультипроцесорним системам. Поняття *кластер* визначається як група взаємоз'єднаних обчислювальних систем (вузлів), що сумісно функціонують, складаючи єдиний обчислювальний ресурс і створюючи ілюзію наявності єдиної обчислювальної машини. Як вузол кластера може бути однопроцесорна ЕОМ і ОС типу SMP або МРР. Важливим є тільки те, що кожний вузол є здатним функціонувати самостійно і окремо від кластера. В плані архітектури сутність кластерних обчислень зводиться до об'єднання декількох вузлів високошвидкісною мережею. Для опису такого підходу, крім терміну «кластерні обчислення», достатньо часто використовують такі назви: *кластер робочих станцій, гіперобчислення, паралельні обчислення на базі мережі, ультраобчислення.*

Як вузли кластерів можуть використовуватись однакові ОС (гомогенні кластери), а також різні (гетерогенні кластери). За своєю архітектурою кластерна ОС є слабкозв'язаною системою.

На рівні апаратного забезпечення кластер – це просто сукупність незалежних обчислювальних систем, які об'єднані мережею. У разі з'єднання ЕОМ у кластер майже завжди підтримуються прямі міжмашинні зв'язки. Рішення можуть бути простими, що ґрунтуються на апаратурі Ethernet, або складними з високошвидкісними мережами з пропускною здатністю у сотні мегабайтів у секунду.

Вузли кластера контролюють працездатність один одного та ведуть обмін специфічною інформацією, що є характерною для кластера. Контроль працездатності здійснюється за допомогою спеціального сигналу, який часто називають *heart-beat*, що значить «серцебиття». Цей сигнал передається вузлами кластера один одному, щоб підтвердити їх нормальне функціонування.

Невід'ємна частина кластера – спеціалізоване програмне забезпечення (ПЗ), на яке покладається задача забезпечення безперебійної роботи у разі відмови одного або декількох вузлів. Таке ПЗ виконує перерозподіл обчислювального навантаження у разі відмови одного або декількох вузлів кластера, а також відновлення обчислень у випадку виникнення збою у вузлі. Крім того, якщо в кластері сумісно використовуються диски, кластерне ПЗ підтримує єдину файлову систему.

Можливість практично необмеженого нарощування числа вузлів та відсутність єдиної операційної системи робить кластерні архітектури успішно масштабованими, і навіть системи з сотнями і тисячами вузлів показують себе на практиці з позитивного боку.

6.5. АРХІТЕКТУРА ПОТОКОВИХ ОБЧИСЛЮВАЛЬНИХ СИСТЕМ

Існують труднощі, пов'язані з розв'язанням проблем автоматизації паралельного програмування, необхідного для забезпечення ефективного використання для широкого кола задач матричних (паралельних) ОС, тобто систем типу SIMD (ОКМД).

Конвеєрні і конвеєрно-векторні ОС найбільший ефект дають під час реалізації спеціалізованих систем з фіксованими алгоритмами, за яких досягається оптимальна структура для задачі, яка розв'язується. В ОС універсального призначення склад позицій конвеєра є далеко не оптимальним для кожної даної задачі, до того ж виникають витрати часу на комутацію позицій конвеєра під поточну команду.

В таких умовах актуальними є дослідження нових шляхів побудови високопродуктивних ОС, одними з яких є ОС з управлінням потоком даних (операндів), або, іншими словами, потокові ОС.

У системах з управлінням потоками даних передбачається наявність великої кількості спеціалізованих операційних блоків для певних видів операцій (додавання, множення тощо, окремих для різних типів даних). Дані наділяються покажчиками типу даних (тегами), на основі яких, у міру готовності даних (операндів), до обробки (а не в порядку послідовності команд у програмі) вони завантажуються у відповідні вільні операційні блоки. У разі достатньої кількості операційних блоків може бути досягнутий високий рівень розподілення обчислювального процесу (близький до «потенційного паралелізму» програми). До того ж у самих операційних блоках може бути

використана конвеєрна обробка. Таким чином, утворюються умови для реалізації високої продуктивності системи.

В усіх раніше розглянутих ЕОМ і обчислювальних системах порядок виконання операцій над даними у ході розв'язання задачі суворо детермінований, він однозначно визначається послідовністю команд програми.

Принципова відмінність потокових машин полягає в тому, що команди виконуються не за порядком чергування команд у тексті програми, а в міру готовності їх операндів. Як тільки будуть вирахувані операнди команди, вона може захоплювати вільний операційний простір і виконувати задану їй операцію. В цьому випадку послідовність, в якій виконуються команди, вже не є детермінованою.

Розглянемо як приклад обчислення на потоковій ОС коренів квадратного рівняння:

$$ax^2 - bx + c = 0,$$

при $b^2 - 4ac > 0;$

$$x_{1,2} = \frac{b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Будемо вважати, що на вхід системи надходить група даних a_i , b_i , c_i і за ними обчислюються корені рівняння. Розв'язання може бути отримане за допомогою операційних пристроїв, які виконують операції додавання, віднімання, множення, піднесення до степеня, ділення і вирахування квадратного кореня.

Процес розв'язання можна подати у такій послідовності:

K0: A1 := 2×a; K1: B1 := b^2<піднесення до степеня>;	K5: D1 := sqrt(D) <обчислення квадратного кореня>;
	K6: B2 := b + D1;
K2: C := 4×c;	K7: B3 := b – D1;
K3: C1 := C×a;	K8: X1 := B2/A1;
K4: D := B1 – C1;	K9: X2 := B3/A1.

У потокових обчислювальних моделях для описання обчислень використовують орієнтований граф, у якому вершини відображають операції, а дуги показують потоки даних між вершинами графа, які вони з'єднують.

На рис. 6.13 показаний граф потоку даних, які управляють розв'язанням задачі, що розглядається. Великими кружками показані операційні пристрої, а маленькими – мітки готовності даних. У початковому стані процесу готові всі вхідні дані.

Вершина графа активізується в довільному порядку у разі готовності їх операндів. Можна вважати, що активізація вершини супроводжується

поглинанням міток готовності на їх входах. По закінченні операції міткою готовності позначається вихідна дуга вершини. Переміщення міток за графом потоку даних відображає проходження обчислювального процесу.

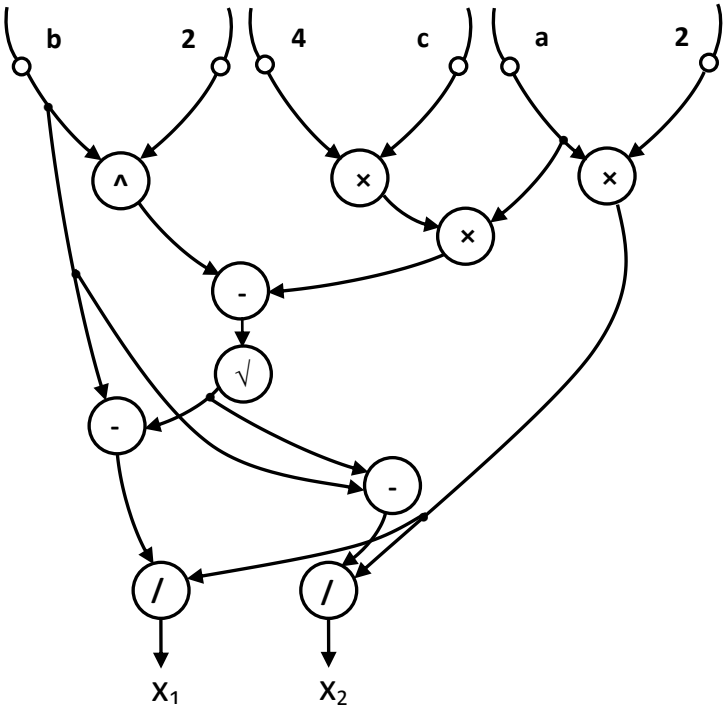


Рис. 6.13. Граф потоків операндів

Виникає запитання, як дані і відповідні команди знаходять один одного? Для відповіді на нього звернемося до рис. 6.14, який пояснює ідею процесора, що управляє потоком даних.

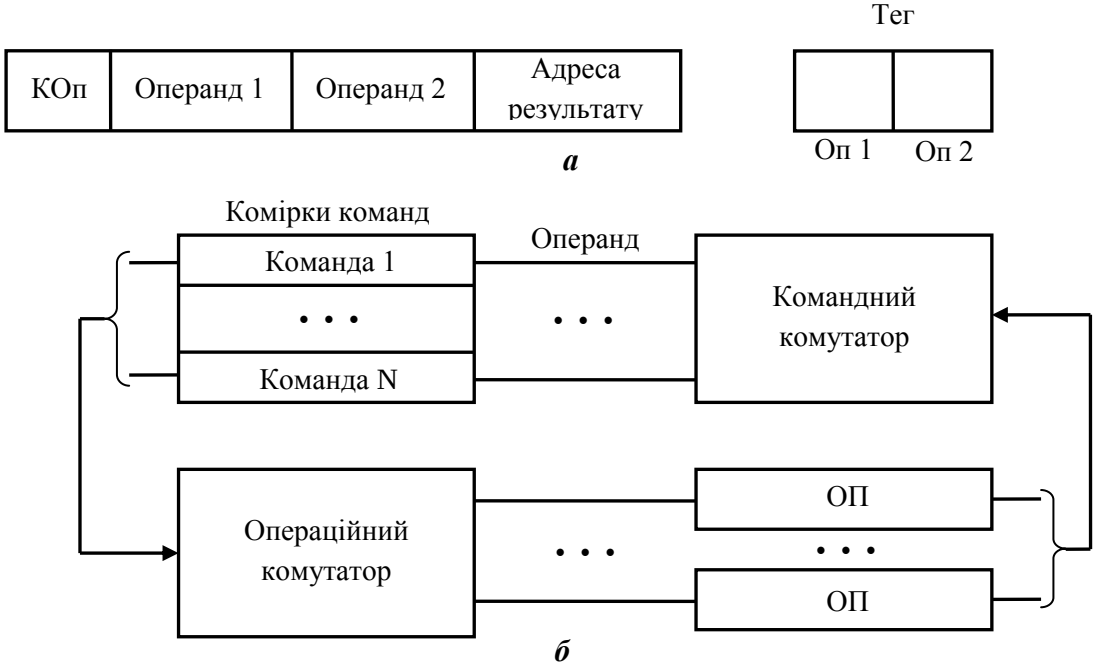


Рис. 6.14. Спрощена структура процесора з управлінням потоком даних:
а – структура команди; б – ОП – операційний пристрій

«Потокова програма» розташовується в масиві комірок команд. Команда, наряду з кодом операції, містить поля, куди заносяться готові операнди, і поле, яке містить адреси команд, в які повинен бути направлений як операнд результат операції. Крім того, кожній команді поставлений у відповідність дворозрядний тег (що розташований в управляючому пристрої), розряди якого встановлюються в 1 при занесенні в тіло команди відповідних операндів.

У стані тегу 11 (обидва операнди готові) ініціюється запит до операційного комутатора на передачу готової команди у відповідний коду операції (і тегу операнда, який визначає тип даних) операційний пристрій.

Результат виконання команди над її безпосередньо адресованими операндами направляється через командний комутатор згідно з вказаними в команді адресами в комірки команд і розташовується в їх поля операндів. Далі вказана процедура циклічно повторюється, до того ж управління цим процесом повністю децентралізоване.

6.6. БАГАТОМАШИННІ ТА БАГАТОПРОЦЕСОРНІ КОМП'ЮТЕРНІ СИСТЕМИ

Обчислювальні системи можуть будуватися на базі декількох комп'ютерів або на базі декількох процесорів. В першому випадку ОС буде *багатомашинною*, в другому - *багатопроцесорною*.

Багатомашинна ОС містить деяке число комп'ютерів, інформаційно взаємодіючих між собою. Комп'ютери можуть знаходитися поряд один з одним, а можуть бути віддаленими один від одного на деяку, іноді значну відстань (обчислювальні мережі).

В ***багатомашинних ОС*** кожний комп'ютер працює під управлінням своєї операційної системи (ОпС). А оскільки обмін інформацією між комп'ютерами, що взаємодіють один з одним, виконується під управлінням ОпС, динамічні характеристики процедур обміну дещо погіршуються (потрібен час на узгодження роботи самих ОпС). Інформаційна взаємодія комп'ютерів в багатомашинній ОС може бути організовано на рівні:

- процесорів;
- оперативної пам'яті;
- каналів зв'язку.

При безпосередній взаємодії процесорів один з одним інформаційний зв'язок реалізується через регістри процесорної пам'яті і вимагає наявності в ОпС дуже складних спеціальних програм.

Взаємодія на рівні оперативної пам'яті (ОП) зводиться до програмної реалізації загального поля оперативної пам'яті, що дещо простіше, але також вимагає суттєвої модифікації ОпС. Під загальним полем мається на увазі

рівнодоступність модулей пам’яті: всі модулі пам’яті доступні всім процесорам і каналам зв’язку.

На рівні каналів зв’язку взаємодія організовується найбільш просто і може бути досягнуто зовнішніми по відношенню до ОпС програмами-драйверами, що забезпечують доступ від каналів зв’язку однієї машини до зовнішніх пристроїв інших (формується загальне поле зовнішньої пам’яті і загальний доступ до пристроїв вводу-виводу).

Все вищесказане ілюструється схемою взаємодії комп’ютерів в двохмашинній ОС, представленої на рис. 6.15.

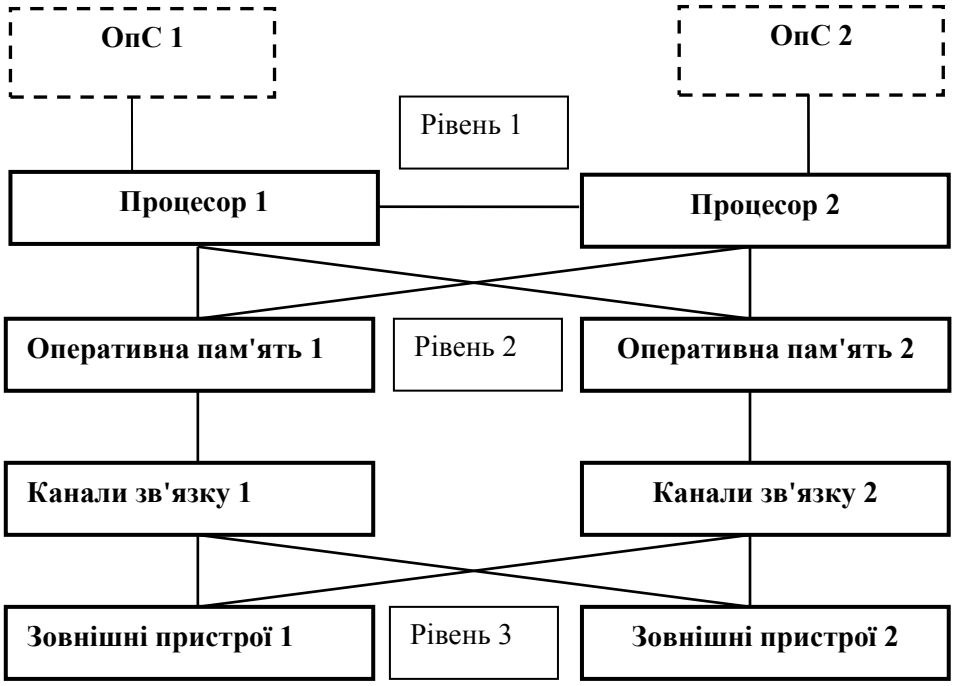


Рис. 6.15. Схема взаємодії комп’ютерів в ОС

Зважаючи на складність організації інформаційної взаємодії на 1-у і 2-у рівнях в більшості багатомашинних ОС використовується 3-й рівень, хоча і динамічні характеристики (в першу чергу швидкодія), і показники надійності таких систем істотно нижче.

В ***багатопроцесорній ОС*** є декілька процесорів, інформаційно взаємопов’язаних між собою або на рівні регістрів процесорної пам’яті, або на рівні ОЗП. Цей тип взаємодії використовується в більшості випадків, бо організовується значно простіше і зводиться до створення загального поля оперативної пам’яті для всіх процесорів. Загальний доступ до зовнішньої пам’яті і пристроїв вводу-виводу забезпечується звичайно через канали ОЗП. Важливим є і те, що багатопроцесорна обчислювальна система працює під управлінням єдиної ОпС, загальної для всіх процесорів. Це істотно покращує динамічні характеристики ОС, але вимагає наявності спеціальної, дуже складної ОпС.

Схема взаємодії процесорів в ОС показана на рис. 6.16.

Рис. 6.16. Схема взаємодії процесорів в ОС

Швидкодія і надійність багатопроцесорних ОС в порівнянні з багатомашинними, взаємодіючими на 3-у рівні, істотно підвищуються,

по – перше, із-за більш швидкого обміну інформацією між процесорами, більш швидкого реагування на ситуації, що виникають в системі;

по – друге, зважаючи на більший ступінь резервування пристроїв системи (система зберігає працездатність, поки працездатні хоча б по одному модулю кожного типу пристроїв).

Типовим прикладом масових багатомашинних ОС можуть служити комп’ютерні мережі, прикладом багатопроцесорних ОС - суперкомп'ютери.

6.7. СУЧАСНІ СУПЕРКОМП'ЮТЕРИ

Нові наукові області вимагають все більше обчислювальної потужності. Сучасне прогнозування погоди, моделювання ядерних випробувань, моделювання клітин на молекулярному рівні і навіть імітація людського мозку з кожним днем вимагають все більше і більше від потужних суперкомп'ютерів. Існує дуже багато компаній які конкурують між собою у створенні найпотужнішого суперкомп'ютера в світі. У процесі такої своєрідної гонки інженери розробляють, модифікують багато частин і компонентів комп'ютера. Більшість з них дуже схожі на компоненти звичайного настільного комп'ютера.

Центральний процесор. Сучасні суперкомп'ютери часто об'єднують десятки тисяч споживчих процесорів в масиви.

Охолодження. Потужні суперкомп'ютери споживають дуже багато енергії. Наприклад, для нормальної роботи Tianhe 2 потрібно стільки енергії скільки споживає невелике місто. Велика частина цієї енергії виділяється як тепло,

тому інженери повинні подбає про те щоб їх суперкомп'ютера для нормальної роботи було досить прохолодно.

Транзистори. Чим більше транзисторів на інтегральній схемі, тим більше її обчислювальна потужність і здатність виконувати більшу кількість операцій.

Розглянемо п’ять перших суперкомп'ютерів зі списку Top500 в 2017році.

Проект Топ-500 був запущений в 1993-му, і двічі на рік (у червні і листопаді) публікує актуальний рейтинг найпотужніших суспільно відомих комп'ютерних систем світу.

Sunway TaihuLight

Найпотужнішим суперкомп'ютером в світі на 2017 рік є Sunway TaihuLight.

Продуктивність Sunway TaihuLight становить 93 петафлопс (10 в 15-му ступені обчислювальних операцій з плаваючою комою в секунду).

Sunway TaihuLight, який використовується для кліматичного моделювання і проведення медико-біологічних досліджень, містить 10,65 мільйонів ядер - близько 41 тисячі вузлів. Суперкомп'ютер розроблений Національним дослідницьким центром паралельної обчислювальної техніки і технологій КНР, його розмістили в Національному центрі суперкомп'ютерів в місті Усі провінції Цзяньсу.

Втім, комп'ютер TaihuLight, з його продуктивністю в 93 петафлопса, здасться блідою плямою на тлі ексафлопсного комп'ютера, який китайський уряд планує створити до 2020 року. Але вже в цьому році, згідно з повідомленнями офіційних ЗМІ країни, Китай планує розробити першу робочу модель ексафлопсной кібернетичної машини. Один ексафлоп дорівнює тисячі петафлоп і означає мільярд мільярдів (або квінтильйон, або 10 000 000 000 000 000 000) операцій в секунду.

Tianhe-2

Tianhe-2 раніше був відомий як MILKYWAY-2 був розроблений Національним університетом технологій оборони Китаю. Став найпотужнішим суперкомп'ютером у світі в 2013 році коли обігнав свого конкурента Titan. На сьогоднішній день він стоїть на другому місці. Максимальна швидкість досягає 33,86 petaFLOPS. Така величезна продуктивність завдяки 16 тисячам вузлів які складаються з Intel Ivy Bridge і Xeon Phi. Китай створив цей суперкомп'ютер спеціально для моделювання різних додатків безпеки.

Piz Daint

Третю позицію займає потужний швейцарський суперкомп'ютер Piz Daint, який в 2016 році отримав велике оновлення. Модифікація дозволила збільшити продуктивність в три рази, таким чином Piz Daint отримав швидкість 25,3 petaFLOPS. Таким чином зараз цей суперкомп'ютер найпотужніший за межами Азії. Незабаром вчені планують підключити його до Великого адронного колайдера.

Gyoukou

На четвертому місці знаходиться система Gyoukou, що належить Японському агентству науки і технологій з вивчення морських надр (Japan Agency for Marine-Earth Science and Technology). Суперкомп'ютер має 1250 16-ядерних процесорів Xeon і шину Infiniband EDR, але більшу частину обчислювальної потужності йому забезпечують 19,84 млн ядер прискорювачів Pezy-SC2. Максимальна стійка продуктивність системи становить 19,14 PFLOPS, а пікова − 28,19 PFLOPS. Дуже суттєвою особливістю Gyoukou стала висока ефективність енергоспоживання, яка становить 14,17 GFLOPS / Вт і в два рази перевершує ефективність сусідів цього суперкомп'ютера по рейтингу Top500.

Gyoukou змістив з раніше займаних місць американський суперкомп'ютер Titan, побудований компанією Cray, а також Sequoia, створений IBM.

Titan

Titan є самим відомим суперкомп'ютером на заході. Знаходиться він в Національній лабораторії Oak Ridge в Теннессі. Довгий час був найпотужнішим суперкомп'ютером у світі, поки Tianhe-2 не обігнав його в 2013 році. Titan перший хто об'єднав в собі процесори AMD Opteron і графічні процесори NVIDIA Tesla, таким чином його продуктивність становить 19,14 PFLOPS, а пікова - 27 petaFLOPS.

6.8. АРХІТЕКТУРА НЕЙРОКОМП'ЮТЕРІВ

6.8.1. Визначення поняття "нейрокомп'ютер"

В середині 80–х років минулого століття в США, а потім в Японії і країнах ЄЕС були розгорнуті широкомасштабні національні і міжнародні програми досліджень і розробок, які були направлені на створення нейрокомп'ютерів – ЕОМ на основі штучних нейронних мереж, які володіють розвиненим інтелектом і програмуються шляхом навчання на прикладах розв'язання задач.

У 1995 році була завершена розробка першого нейрокомп'ютера на стандартній мікропроцесорній елементній базі.

В останні роки, у зв'язку з бурхливим розвитком обчислювальної техніки, теорії хаосу і теорії самоорганізації, а також на підставі досягнень синергетики і теорії диссипативних структур (структур, фазовий обсяг яких зменшується з часом) спостерігається якісний бум у розвитку *нейрокомп'ютерних технологій.*

У світі існує декілька десятків спеціалізованих фірм, які випускають продукцію в області нейроінформатики, крім того, багато спеціалізованих комп'ютерних фірм – IBM, Siemens Nocdorff, Mitsubisi – ведуть дослідження і мають власні розробки в даній області. Фірма Siemens в останні роки випускає спеціальні *нейрочипи.* Дані пристрої складаються з великого числа нейропро-

цесорів, здатних, на відміну від звичайних процесорів, робити послідовно–паралельні обчислення. Така схема обчислень пов'язана з особливістю роботи головного мозку людини, аналогом котрого і є нейрочипи.

В основу побудови нейрокомп'ютерів ліг штучний нейрон (*перцептрон*). Кожний нейрон отримує сигнали від сусідніх нейронів за допомогою спеціальних нервових волокон. Ці сигнали можуть бути збудливими або гальмуючими. Їх сума складає електричний потенціал у середині тіла нейрона. Коли потенціал перевищує деякий поріг, нейрон переходить у збуджений стан і посилає сигнал по вихідному нервовому волокну. Окремі штучні нейрони з'єднуються один з одним різноманітними методами. Це дозволяє створювати різноманітні нейронні мережі з різною архітектурою, правилами навчання і можливостями.

Нейрокомп'ютер – це обчислювальна система з архітектурою MSIMD (паралельно-векторна модифікація), в якій реалізовані два принципових технічних рішення: спрощено до рівня нейрона процесорний елемент однорідної структури і різко ускладнені зв'язки між елементами; програмування обчислювальної структури перенесено на зміну вагових зв'язків між процесорними елементами.

Загальне визначення нейрокомп'ютера може бути представлене таким чином.

Нейрокомп'ютер – це обчислювальна система з архітектурою апаратного і програмного забезпечення, адекватною виконанню алгоритмів, представлених у нейромережному логічному базисі.

6.8.1. Архітектурні особливості й апаратне забезпечення нейрокомп'ютерів

Ідея побудови автомата на основі порогових елементів, подібних до нейронів (нервових клітин), які здатні виконувати логічні функції, була сформульована більше як півстоліття тому Мак–Каллоком і Піттсом. Однак задача проектування систем на основі порогових елементів викликала великі труднощі і її рішення було знайдене лише 20 років по тому. Це було настільки складним, що практично виключало можливість синтезу автоматів, які складалися більш як з десятків нейронів.

Системи на основі порогових елементів отримали назву *штучних нейронних мереж (ШНМ)* [23]. Перші роботоздатні штучні нейронні системи (ШНС) були створені вже в кінці 50–х років минулого століття – перцептрон Ф. Розенблатта, система "Альфа" А.Г. Івахненко. Перші великі перцептрони на основі аналогової і цифрової техніки ("Адам–А" і"Адам–Д") за межами США були створені в 1969–71 рр. в одному з київських НДІ.

Схема перцептрона Розенблатта приведена на рис. 6.17.

Він уміщує три шари порогових елементів. Вхідні сигнали (стимули), діючи на рецептори (S-елементи), переводять їх у збуджений стан. S-елементи випадковим чином зв'язані з сукупністю асоціативних нейронів (A-елементів). Вихід A-елемента відрізняється від нуля тільки тоді, коли збуджено достатньо велике число зв'язаних з ним рецепторів. Реакції A-елементів поступають на входи ефекторів (R-елементів) через зв'язки, ваги котрих змінюються у процесі навчання. В ефекторах обчислюється постсинаптичний потенціал – врівноважена сума сигналів, які поступили. Як правило, в перцептроні для кожного запам'ятовуючого образу виділяється один ефектор, і рішення приймається по максимальному значенню постсинаптичного потенціалу.

Властива перцептрону Розенблатта неоднорідність структури (розділення на S-, A- і R-елементи) в більш пізніх моделях ШНС втрачається.

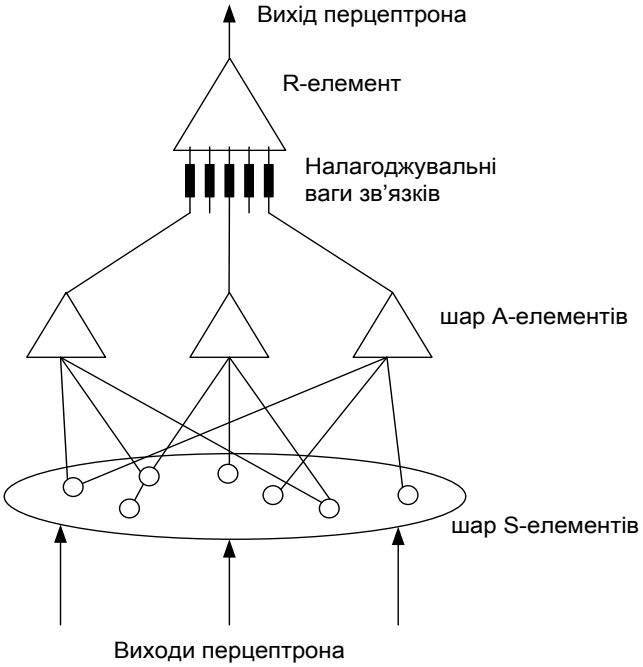


Рис. 6.17. Перцептрон Розенблатта

На рис. 6.18 приведена модель штучного нейрона. Штучний нейрон імітує в першому наближенні властивості біологічного нейрона.

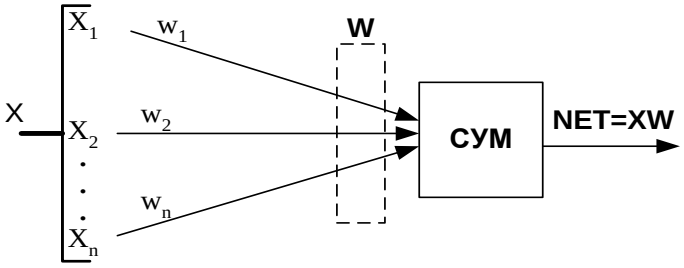


Рис. 6.18. Модель штучного нейрона

Множина вхідних сигналів x_1, x_2, …, x_n надходить на штучний нейрон. Дані вхідні сигнали, у сукупності позначені вектором X, відповідають сигналам, які приходять у синапси біологічного нейрона. Кожний сигнал збільшується на відповідну вагу w_i (i=1,2,…, n) і надходить у підсумовуючий блок СУМ (адаптивний суматор). Кожна вага відповідає "силі" одного синаптичного біологічного зв'язку. Множина ваг у сукупності утворює вектор ваг W.

Підсумовуючий блок, що відповідає тілу біологічного елемента, підсумовує зважені входи алгебраїчно, утворюючи вихід NET. У векторних позначеннях це може бути виражено таким чином: NET=XW. Сигнал NET у подальшому, як правило, перетворюється активаційною функцією F і дає вихідний нейронний сигнал OUT. Активаційна функція може бути звичайною лінійною функцією

OUT=K(NET),

де K – постійна граничної функції

OUT=1, якщо NET >T

OUT=0 в інших випадках,

де T– деяка постійна гранична величина.

На рис. 6.19 приведена структура штучного нейрона з активаційною функцією.

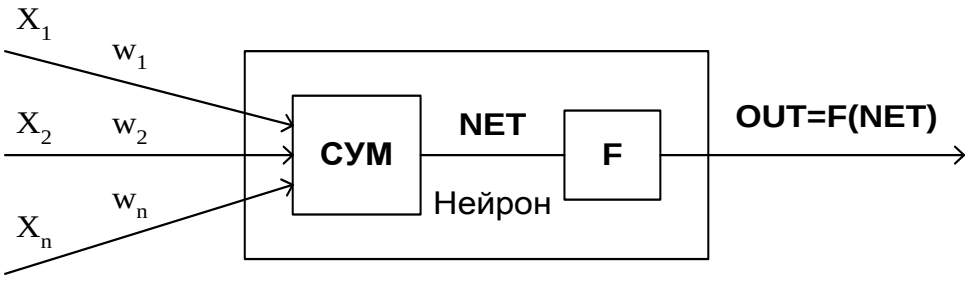


Рис. 6.19. Структура штучного нейрона з активаційною функцією

Блок F приймає сигнал NET і видає сигнал OUT. Якщо блок F звужує діапазон зміни величини NET так, що за будь-яких значень NET значення OUT належать деякому кінцевому інтервалу, то F називається стискаючою функцією. Як стискаюча функція часто використовується *логістична чи сигмоїдальна* (S-подібна) функція, що математично виражається так:

$F(x)= 1/(1+e^{-x})$.

Таким чином,

$OUT= 1/(1+e^{-NET})$.

За аналогією з електронними системами активаційну функцію можна вважати нелінійною підсилювальною характеристикою штучного нейрона. Коефіцієнт підсилення обчислюється як відношення збільшення величини OUT до його невеликого збільшення, що викликало величини NET.

Розглянута модель штучного нейрона ігнорує багато властивостей свого біологічного аналога. Наприклад, вона не бере до уваги затримки в часі, що впливають на динаміку системи. Вхідні сигнали відразу ж породжують вихідний сигнал.

Починаючи з перцептрона основна увага розробників ШНС приділялась розробці і вдосконаленню методів їх навчання. Через відсутність надійної теорії навчання дані розробки носили в основному евристичний характер і отримали назву нейропарадигм [23].

На сьогоднішній день кількість проданих в світі нейрокомп'ютерів обчислюється десятками, а можливо і сотнями тисяч. В основному це *нейрокомп'ютерні* програми для персональних комп'ютерів, які призначені для розв'язання задач апроксимації і прогнозування числових даних. Близько 5% нейрокомп'ютерів відносяться до пристроїв професійного рівня, котрі орієнтовані на застосування потужних робочих станцій і апаратних *нейроакселераторів.* Програмне забезпечення таких систем, як правило, вміщує *бібліотека нейропарадигм*, що дозволяє під час розв'язання задач використовувати різні типи нейронних мереж.

Типовим прикладом може бути система *Brain Maker фірми CSS* (США). Система може працювати на будь-якому комп'ютері, де встановлено Windows. Базова версія орієнтована на широке коло користувачів. Її застосування не потребує спеціальних знань. Для розширення можливостей системи служить набір додаткових програм Toolkit Option, які дозволяють прискорити процес навчання і покращити подання графічних даних.

6.8.3. Нейрокомп'ютерні мережі

Штучна нейронна мережа може розглядатися як направлений граф зі зваженими зв'язками, в якому штучні нейрони є вузлами. За архітектурою зв'язків ШНМ можуть бути згруповані у два класи (рис. 6.20): мережі прямого поширення, в яких графи не мають петель, і рекурсивні мережі, або мережі із зворотними зв'язками. У найбільш розповсюдженому сімействі мереж першого класу, багатошарових парцептронів, нейрони розташовані шарами і мають односпрямовані зв'язки між шарами.

Мережі прямого поширення є статичними в тому сенсі, що на заданий вхід вони виробляють одну сукупність вихідних значень, які не залежать від попереднього стану мережі.

Рекурентні мережі є динамічними мережами в тому плані, що в силу зворотних зв'язків у них модифікуються входи нейронів, що приводить до зміни стану мережі.

Рис. 6.20. Базові архітектури нейронних мереж

Навчання штучних нейронних мереж. Здатність до навчання є фундаментальною властивістю мозку людини. У контексті ШНМ процес навчання може розглядатися як настроювання архітектури мережі і ваг зв'язків для ефективного виконання спеціальної задачі. Нейронна мережа повинна налагодити ваги зв'язків за наявною навчальною вибіркою. Функціонування мережі поліпшується в міру ітеративного настроювання вагових коефіцієнтів.

Властивість мережі навчатися на прикладах робить їх більш привабливими. Алгоритм навчання означає процедуру, в якій використовуються правила навчання для настроювання ваг.

Існують три парадигми навчання:

– з "учителем";

– без "учителя" (самонавчання);

– змішана (комбінована).

У *першому випадку* нейронна мережа має в своєму розпорядженні правильні відповіді на кожний вхідний приклад. Ваги налагоджуються так, щоб мережа робила відповіді найбільш близькі до відомих правильних відповідей. Посилений варіант навчання з "учителем" припускає, що відома тільки критична оцінка правильності виходу нейронної мережі, але не самі правильні значення виходу.

Навчання *без "учителя"* не вимагає знання правильних відповідей на кожний приклад навчальної вибірки. В даному випадку розкривається внутрішня структура даних чи кореляції між зв'язками в системі даних, що дозволяє розподілити зв'язки за категоріями.

Під час *змішаного* навчання частина ваг визначається за допомогою навчання з "учителем", в той час як інша за допомогою самонавчання.

Відомі чотири основних типи правил навчання: *корекція помилково, машина Больцмана, правило Хебба і навчання методом змагання* [23].

Правило корекції помилково. Під час навчання з "учителем" для кожного вхідного прикладу заданий бажаний вихід *d*. Реальний вихід мережі *y* може не збігатися з бажаним. Принцип корекції помилково під час навчання полягає у використанні сигналу *(d–y)* для модифікації ваг, що забезпечує поступове зменшення помилки. Навчання має місце тільки у випадку, коли перцептрон помиляється.

Навчання Больцмана являє собою стохастичне правило навчання, котре випливає з інформаційних теоретичних і термодинамічних принципів. Метою навчання Больцмана є таке настроювання вагових коефіцієнтів, за якого стани видимих нейронів задовольняють бажаний розподіл ймовірностей. Навчання Больцмана може розглядатися як спеціальний випадок корекції помилково, в якому під помилкою розуміється розбіжність кореляції станів у двох режимах.

Правило Хебба. Найстаршим навчальним правилом є постулат Хебба. Хебб спирався на такі нейрофізіологічні спостереження: якщо нейрони по обидва боки синапса активізуються одночасної регулярно, то сила синаптичного зв'язку зростає. Важливою особливістю цього правила є те, що зміна синаптичної ваги залежить тільки від активності нейронів, що зв'язані даним синапсом. Це істотно спрощує ланцюг навчання в реалізації VLSI.

Метод змагання. Під час навчання методом змагання вихідні нейрони змагаються між собою за активізацію. Дане явище відоме як правило "переможець бере все". Навчання за допомогою змагання дозволяє кластеризувати вхідні дані: подібні приклади групуються мережею відповідно до кореляцій і представляються одним елементом. Під час навчання модифікуються тільки ваги нейрона "переможця". Ефект даного правила досягається за рахунок такої зміни збереженого в мережі зразка (вектора ваг зв'язків, що переміг нейрона), за якого він стає трохи ближче до вхідного прикладу.

У порівнянні із звичайними комп'ютерами нейрокомп'ютери мають ряд переваг:

- висока швидкодія, котра пов'язана з тим, що алгоритми нейроінформатики мають високий ступінь паралельності;
- нейросистеми стійкі до завад;
- стійкі і надійні нейросистеми можуть створюватися з ненадійних елементів, які мають значний розкид параметрів.

Крім переваг нейросистеми мають і ряд недоліків:

- вони створюються спеціально для розв'язання конкретних задач, пов'язаних з нелінійною логікою і теорією самоорганізації. Розв'язання подібних задач на звичайних комп'ютерах можливе тільки чисельними методами.
- через свою унікальність дані пристрої достатньо дорогі.

Незважаючи на недоліки, нейрокомп'ютери можуть успішно використовуватися в різноманітних областях народного господарства:

- керування в режимі реального часу (літаками, ракетами, технологічними процесами беззупинного виробництва);
- розпізнавання об'єктів (букв і ієрогліфів, сигналів радара і сонара, відбитків пальців у криміналістиці, захворювань за симптомами у медицині, місцевостей під час пошуку корисних копалин і т.п.);
- прогнози: погоди, курсу акцій (та інших фінансових показників), політичних подій, поводження супротивників у військових конфліктах і в економічній конкуренції;
- оптимізація і пошук найкращих варіантів: під час конструювання технічних пристроїв, вибору економічної стратегії і т.п.

6.9. ЕФЕКТИВНІСТЬ ОБЧИСЛЮВАЛЬНИХ СИСТЕМ

6.9.1. Показники ефективності обчислювальних машин

Обчислювальну машину можна визначити множиною показників, що характеризують окремі її властивості. Виникає завдання введення міри для оцінки ступеня пристосованості ОМ до виконання покладених на неї функцій – міри ефективності.

Ефективність визначає ступінь відповідності ОМ своєму призначенню. Вона вимірюється або кількістю витрат, необхідних для отримання певного результату, або результатом, отриманим при певних витратах. Провести порівняльний аналіз ефективності декількох ОМ, ухвалити рішення на використання конкретної машини дозволяє критерій ефективності.

Критерій ефективності – це правило, що служить для порівняльної оцінки якості варіантів ОМ. Критерій ефективності можна назвати правилом переваги порівнюваних варіантів.

Будуються критерії ефективності на основі окремих показників ефективності (показників якості). Спосіб зв'язку між окремими показниками визначає вид критерію ефективності.

Як основні показники ОМ зазвичай розглядають: ємність пам'яті, швидкодію та продуктивність, вартість і надійність [25]. Зупинимося тільки на показниках швидкодії та продуктивності, що зазвичай представляють основний інтерес для користувачів.

Швидкодія. Доцільно розглядати два види швидкодії: номінальну і середню.

Номінальна швидкодія характеризує можливості ОМ під час виконання стандартної операції. Як стандартну зазвичай вибирають коротку операцію додавання. Якщо позначити через $\tau_{дод}$ час додавання, то номінальна швидкодія визначиться з виразу

$$V_{ном} = \frac{1}{\tau_{дод}} \left[\frac{оп}{с} \right]. \qquad (6.1)$$

Середню швидкодію характеризує швидкість обчислень при виконанні еталонного алгоритму або деякого класу алгоритмів. Величина середньої швидкодії залежить як від параметрів ОМ, так і від параметрів алгоритму і визначається співвідношенням

$$V_{сер} = \frac{N}{T_e}, \qquad (6.2)$$

де T_e – час виконання еталонного алгоритму;

N – кількість операцій, що містяться в еталонному алгоритмі.

Позначимо через n_i число операцій i-го типу; l – кількість типів операцій в ($i = 1, 2..., l$); τ_i – час виконання операції i-го типу.

Час виконання еталонного алгоритму розраховується за формулою

$$T_e = \sum_{i=1}^{l} \tau_i n_i . \qquad (6.3)$$

Підставивши (6.3) у вираз для $V_{сер}$, отримаємо

$$V_{сер} = \frac{N}{\sum_{i=1}^{l} \tau_i n_i} . \qquad (6.4)$$

Розділимо чисельник і знаменник в (6.4) на N:

$$V_{сер} = \frac{1}{\sum_{i=1}^{l} \frac{n_i}{N} \tau_i} . \qquad (6.5)$$

Позначивши частоту появи операції i-го типу в (6.5) через $q_i = \frac{n_i}{N}$, запишемо остаточну формулу для розрахунку середньої швидкодії

$$V_{сер} = \frac{1}{\sum_{i=1}^{l} q_i \tau_i} \left[\frac{оп}{с} \right]. \qquad (6.6)$$

У виразі (6.6) вектор $\{\tau_1, \tau_2, ..., \tau_l\}$ характеризує систему команд ОМ, а вектор $\{q_1, q_2, ..., q_l\}$, що називається частотним вектором операцій, характеризує алгоритм.

Очевидно, що для ефективної реалізації алгоритму необхідно прагнути до збільшення $V_{сер}$. Якщо $V_{ном}$ головним чином відштовхується від швидкодії елементної бази, то $V_{сер}$ дуже сильно залежить від оптимальності вибору команд ОМ.

Формула (6.6) дозволяє визначити середню швидкодію машини під час реалізації одного алгоритму. Розглянемо більш загальний випадок, коли повний алгоритм складається з декількох окремих, періодично повторюваних алгоритмів. Середня швидкодія під час рішення повної задачі розраховується за формулою

$$V_{сер}^{n} = \frac{1}{\sum_{j=1}^{m}\sum_{i=1}^{l}\beta_j q_{ji}\tau_i}\left[\frac{оп}{с}\right], \qquad (6.7)$$

де m – кількість окремих алгоритмів;

β_j – частота появи операцій j-го окремого алгоритму в повному алгоритмі;

q_{ij} – частота операцій i-го типу в j-му окремому алгоритмі.

Позначимо через N_j і T_j – кількість операцій і період повторення j-го окремого алгоритму; $T_{\max} = \max_j (T_1, ..., T_j, ..., T_m)$ – період повторення повного алгоритму; $a_j = T_{max}/T_j$ – циклічність включення j-го окремого алгоритму в повному алгоритмі.

Тоді за час T_{max} в ОМ буде виконано $N_{\max} = \sum_{j=1}^{m}\alpha_j N_j$ операцій, а частоту появи операцій j-го окремого алгоритму в повному алгоритмі можна визначити з виразу

$$\beta_j = \frac{\alpha_j N_j}{N_{\max}}. \qquad (6.8)$$

Для розрахунку за формулами (6.7, 6.8) необхідно знати параметри ОМ, представлені вектором $\{\tau_1, \tau_2, ..., \tau_l\}$, параметри кожного j-го окремого алгоритму – вектор $\{q_{j1}, q_{j2}, ..., q_{jl}\}$ і параметри повного алгоритму – вектор $\{\beta_1, \beta_2, ..., \beta_m\}$.

Продуктивність ОМ оцінюється кількістю еталонних алгоритмів, що виконуються в одиницю часу:

$$P = \frac{1}{T_e}\left[\frac{задач}{с}\right]. \qquad (6.9)$$

Продуктивність під час виконання повного алгоритму оцінюється за формулою

$$P_n = \frac{1}{\sum_{j=1}^{m}\sum_{i=1}^{l}\alpha_j N_j q_{ij}\tau_i}\left[\frac{задач}{с}\right]. \qquad (6.10)$$

Продуктивність є більш універсальним показником, ніж середня швидкодія, оскільки в явному вигляді залежить від порядку проходження завдань через ОМ.

На практиці зазвичай використовують простіші вирази [16]. Для оцінки часу виконання програми з динамічною кількістю команд N використовують вираз

$$T = \frac{N \cdot K}{F}\ , \qquad (6.11)$$

де K – середня кількість тактів, що витрачаються на вибірку і виконання однієї команди;

F – тактова частота процесора.

Для оцінки продуктивності використовують *пропускну здатність* процесора – кількість команд, що виконуються за одну секунду. У разі послідовного виконання команд пропускна здатність P_s визначається за формулою

$$P_s = \frac{F}{K}\ . \qquad (6.12)$$

Для *конвеєрного* процесора пропускна здатність сама по собі ще не є свідоцтвом хорошої продуктивності. Основні показники конвеєрного процесора були розглянуті в розділі 4.5.2. Реальною мірою продуктивності комп'ютера є загальний час виконання програми. В загальному випадку *n*-ступінчастий конвеєр потенційно підвищує продуктивність в *n* разів. Виходить, що чим більше значення *n*, тим вище продуктивність процесора. Проте реальна продуктивність конвеєрного процесора нижча. Кожного разу, коли відбувається зупинка конвеєра, потік команд, що обробляються процесором, різко скорочується. Таким чином, продуктивність конвеєра багато в чому залежить від таких чинників, як накладні витрати переходів і промахів у разі звернення до кеш-пам'яті.

Скорочення накладних витрат можна добитися за рахунок введення вторинного кеша, який розміщується між первинним кешем, інтегрованим у мікросхему процесора, і основною пам'яттю. Крім того, в комп'ютерах використовується оптимізуючий компілятор, який по можливості віддаляє залежні один від одного команди, поміщаючи між ними інші. А також, якщо в поцесорі існує

черга команд, промах під час вибірки команди може мати значно менші негативні наслідки, оскільки процесор продовжує виконання команд, що знаходяться в черзі.

Для підвищення продуктивності конвеєрного процесора здавалося б доцільним розділяти процес виконання команд на якомога більшу кількість ступенів. Проте чим більше ступенів, тим вище вірогідність зупинок конвеєра. Це пов'язано з тим, що більшість команд виконуються паралельно, в зв'язку з чим навіть значно віддалені одна від одної команди, між якими є залежності, можуть викликати зупинки конвеєра, що, у свою чергу, приводить до зростання накладних витрат переходів. Через перераховані причини підвищення продуктивності за рахунок збільшення кількості ступенів виявляється не таким уже істотним.

Ще один важливий чинник, що впливає на продуктивність, – внутрішні затримки під час виконання процесором базових операцій. Особливої уваги заслуговує затримка в АЛП. У багатьох процесорах тактова частота підбирається так, щоб операція складання в АЛП здійснювалася за один такт. Решта операцій розділяється на кроки, що займають той же час, що і операція складання. При цьому АЛП може мати конвеєрну організацію.

У багатьох конвеєрних процесорах використовується від чотирьох до шести ступенів. У ряді процесорів процедура виконання команди ділиться на менші кроки, використовується більша кількість ступенів конвеєра і вища тактова частота. Наприклад, у процесорі UltraSPARC II застосовується 9-ступінчастий конвеєр, а в процесорі Intel Pentium Pro – 12-ступінчастий, Intel Pentium 4 містить 20-ступінчастий конвеєр і функціонує на тактовій частоті від 1,3 до 1,5 ГГц. Для прискорення роботи на кожному такті виконуються два ступені конвеєра.

6.9.2. Продуктивність мультипроцесорних систем

Через особливості паралельних обчислень для оцінки їх ефективності використовують специфічну систему показників.

Число процесорів багатопроцесорної системи, що паралельно беруть участь у виконанні програми в кожен момент часу *t*, визначають поняттям *ступінь паралелізму D(t)* (DOP, Degree Of Parallelism). Графічне зображення параметра *D(t)* як функції часу називають *профілем паралелізму програми*.

Типовий профіль паралелізму для алгоритму декомпозиції (divide-and-conquer algorithm) показаний на рис. 6.21.

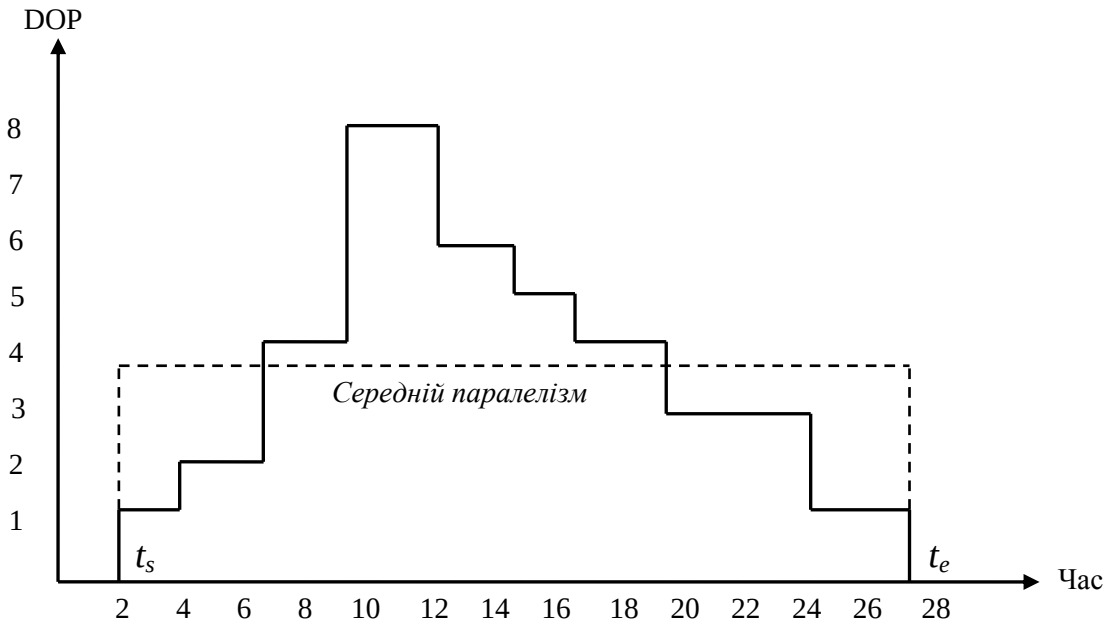


Рис. 6.21. Профіль паралелізму

Зміни в рівні завантаження процесорів за час спостереження залежать від багатьох чинників (алгоритму, доступних ресурсів, ступеня оптимізації, що забезпечується компілятором, і т.д.).

Надалі виходитимемо з наступних припущень: система складається з n гомогенних процесорів; максимальний паралелізм у профілі рівний m і, в ідеальному випадку, $n >> m$. Продуктивність Е одиничного процесора системи виражається в одиницях швидкості обчислень (кількість операцій в одиницю часу) і не враховує витрат, пов'язаних із зверненням до пам'яті і пересилкою даних. Якщо за спостережуваний період завантажені i процесорів, то $D = i$.

Загальний об'єм обчислювальної роботи W (команд або обчислень), виконаної починаючи із стартового моменту t_s до моменту завершення t_e, пропорційний площі під кривою профілю паралелізму:

Середній паралелізм A визначається як

$$A = \frac{1}{t_e - t_s} \int_{t_s}^{t_e} D(t)\, dt \ . \qquad (6.13)$$

У дискретній формі це можна записати так:

$$A = \frac{\sum_{i=1}^{m} i \cdot t_i}{\sum_{i=1}^{m} t_i} \ . \qquad (6.14)$$

Прискорення (speedup) або, точніше, середнє прискорення за рахунок паралельного виконання програми – це відношення часу, потрібного для

виконання якнайкращого з послідовних алгоритмів на одному процесорі, і часу паралельного обчисленню на *n* процесорах.

Без урахування комунікаційних витрат прискорення *S(n)* визначається як

$$S(n) = \frac{T(1)}{T(n)} . \qquad (6.15)$$

Як правило, прискорення задовольняє умову $S(n) \le n$.

Ефективність (efficiency) *n*-процесорної системи – це прискорення на один процесор, що визначається виразом

$$E(n) = \frac{S(n)}{n} = \frac{T(1)}{n \cdot T(n)} . \qquad (6.16)$$

Ефективність зазвичай відповідає умові $1/n \le E(n) \le n$.

Залежно від кількості процесорів у системі розрізняють три можливі варіанти прискорень.

Власне прискорення визначається шляхом реалізації паралельного алгоритму на одному процесорі.

Якщо прискорення, досягнуте на *n* процесорах, дорівнює *n*, то говорять, що алгоритм показує *лінійне прискорення*.

У виняткових ситуаціях прискорення *S(n)* може бути більше, ніж *n*. *У* цих випадках іноді застосовують термін *суперлінійне прискорення*.

До чинників, що обмежують прискорення, слід віднести:

- *Програмні витрати*. Навіть якщо послідовні і паралельні алгоритми виконують одні і ті ж обчислення, паралельним алгоритмам властиві додаткові програмні витрати – додаткові індексні обчислення, що неминуче виникають через декомпозицію даних і розподіл їх по процесорах; різні види облікових операцій, потрібні в паралельних алгоритмах, але відсутні в алгоритмах послідовних.
- *Витрати через дисбаланс завантаження процесорів.* Між точками синхронізації кожен з процесорів повинен бути завантажений однаковим об'ємом роботи, інакше частина процесорів чекатиме, поки останні завершать свої операції. Ця ситуація відома як дисбаланс завантаження. Таким чином, прискорення обмежується найбільш повільним з процесорів.
- *Комунікаційні витрати.* Якщо прийняти, що обмін інформацією та обчислення можуть перекриватися, то будь-які комунікації між процесорами знижують прискорення. В плані комунікаційних витрат важливий рівень гранулярності, що визначає об'єм обчислювальної роботи, яка виконується між комунікаційними фазами алгоритму. Для зменшення комунікаційних витрат вигідніше, щоб обчислювальні гранули були достатньо великими і частка комунікацій була менша.

6.9.3. Закон Амдала

В ідеальному випадку можна вважати, що обчислювальна ситема з *n* процесорів могла б прискорити обчислення в *n* разів. Реально досягти такого показника з ряду причин не вдається. Головна з цих причин полягає в неможливості повного розпаралелювання жодного із завдань. Як правило, в кожній програмі є фрагмент коду, який принципово повинен виконуватися послідовно і лише одним з процесорів. Це може бути частина програми, яка відповідає за запуск задачі і розподіл розпаралеленого по процесорах коду, або фрагмент програми, що забезпечує операції вводу/виводу. Можна привести і інші приклади, але головне полягає в тому, що про повне розпаралелювання задачі говорити не доводиться. Відомі проблеми виникають і з тією частиною задачі, яка піддається розпаралелюванню. Тут ідеальним був би варіант, коли паралельні гілки програми постійно завантажували б усі процесори системи, причому так, щоб навантаження на кожен процесор було однакове. На жаль, обидві ці умови на практиці важко реалізовуються. Таким чином, орієнтуючись на паралельну ОС, необхідно чітко усвідомлювати, що добитися прямо пропорційного числу процесорів збільшення продуктивності не вдасться, і, природно, встає питання про те, на яке реальне прискорення можна розраховувати. Відповідь на це питання в якійсь мірі дає закон Амдала.

Джин Амдал (Gene Amdahl) – один з розробників всесвітньо відомої системи IBM 360. У своїй роботі, опублікованій у 1967 році, він запропонував формулу, яка відображає залежність прискорення обчислень, що досягається на багатопроцесорній ОС, від числа процесорів і співвідношення між послідовною і паралельною частинами програми. Показником скорочення часу обчислень служить така метрика, як «*прискорення*». Нагадаємо, що прискорення S – це відношення часу T_s, що витрачається на проведення обчислень на однопроцесорній ОС (у варіанті якнайкращого послідовного алгоритму), до часу T_p, розв'язання тієї ж задачі на паралельній системі (у разі використання якнайкращого паралельного алгоритму):

$$S = \frac{T_s}{T_p} . \qquad (6.17)$$

Обмеження щодо алгоритмів розв'язання задачі зроблені, щоб підкреслити той факт, що для послідовного і паралельного вирішення кращими можуть бути різні реалізації, а оцінюючи прискорення, необхідно виходити саме з якнайкращих алгоритмів.

Проблема розглядалася Амдалом у наступній постановці. Перш за все, об'єм вирішуваної задачі із зміною числа процесорів, що беруть участь в її розв'язанні, залишається незмінним. Програмний код вирішуваної задачі складається з двох частин: послідовної і такої, що розпаралелює. Позначимо частку операцій, які повинні виконуватися послідовно одним з процесорів,

через f, де $0 \le f \le 1$ (тут частку розуміють не по числу рядків коду, а по числу реально виконуваних операцій). Звідси частка, що приходиться на розпаралелювану частину програми, складе $1 - f$. Крайні випадки в значеннях f відповідають повністю паралельним ($f = 0$) і повністю послідовним ($f = 1$) програмам. Розпаралелювана частина програми рівномірно розподіляється по всіх процесорах.

З урахуванням приведеного формулювання маємо:

$$T_p = f \cdot T_s + \frac{(1-f)\cdot T_s}{n}. \qquad (6.18)$$

У результаті отримуємо формулу Амдала, що виражає прискорення, яке може бути досягнуте на системі з n процесорів:

$$S = \frac{T_s}{T_p} = \frac{n}{1+(n-1)\cdot f}. \qquad (6.19)$$

Формула виражає просту і таку, що володіє великою спільністю залежність.

Якщо спрямувати число процесорів до нескінченності, то в границі отримуємо:

$$\lim_{n\to\infty} S = \frac{1}{f}. \qquad (6.20)$$

Це означає, що якщо в програмі 10% послідовних операцій (тобто $f = 0{,}1$), то, скільки б процесорів не використовувалося, прискорення роботи програми більш ніж вдесятеро ніяк не отримати, але і 10 – це теоретична верхня оцінка найкращого випадку, коли ніяких інших негативних чинників немає. Слід зазначити, що розпаралелювання веде до певних витрат, яких немає у разі послідовного виконання програми. Як приклад таких витрат можна згадати додаткові операції, пов'язані з розподілом програм по процесорах, обмін інформацією між процесорами і так далі.

6.9.4. Закон Густафсона

Відому частку оптимізму в оцінку, що дається законом Амдала, вносять дослідження, проведені Джоном Густафсоном з NASA Ames Research [25]. Вирішуючи на обчислювальній системі з 1024 процесорів три великі завдання, для яких частка послідовного коду f лежала в межах від 0,4 до 0,8%, він набув значень прискорення в порівнянні з однопроцесорним варіантом, рівні відповідно 1021, 1020 і 1016. Згідно з законом Амдала для даного числа процесорів і діапазону f, прискорення не повинне було перевищити величини порядку 201. Намагаючись пояснити це явище, Густафсон прийшов до висновку, що причина криється в початковій передумові, що лежить в основі закону Амдала: збільшення числа процесорів не супроводжується збільшенням

об'єму вирішуваного завдання. Реальна ж поведінка користувачів істотно відрізняється від такого уявлення. Зазвичай, отримуючи в своє розпорядження могутнішу систему, користувач не прагне скоротити час обчислень, а, зберігаючи його практично незмінним, старається пропорційно потужності ОС збільшити об'єм вирішуваного завдання. І тут виявляється, що нарощування загального об'єму програми стосується головним чином частини програми, що розпаралелює. Це веде до скорочення значення *f*. Прикладом може служити вирішення диференціального рівняння в окремих похідних. Якщо частка послідовного коду складає 10% для 1000 вузлових точок, то для 100 000 точок частка послідовного коду знизиться до 0,1%. Тобто, залишаючись практично незмінною, послідовна частина в загальному об'ємі збільшеної програми має вже меншу питому вагу.

Було відмічено, що в першому наближенні об'єм роботи, яка може бути проведена паралельно, зростає лінійно із зростанням числа процесорів у системі. Для того щоб оцінити можливість прискорення обчислень, коли об'єм останніх збільшується із зростанням кількості процесорів у системі (коли постійний загальний час обчислень), Густафсон рекомендує використовувати вираз, запропонований Е. Барсисом (E. Barsis):

$$S = \frac{T_s}{T_p} = \frac{f \cdot T_s + n \cdot (1 - f) \cdot T_s}{f \cdot T_s + (1 - f) \cdot T_s} = n + (1 - n) \cdot f \ . \qquad (6.21)$$

Даний вираз відомий як *закон масштабованого прискорення* або *закон Густафсона* (іноді його називають також законом Густафсона-Барсиса). На закінчення відзначимо, що закон Густафсона не протирічить закону Амдала. Відмінність полягає лише у формі утилізації додаткової потужності ОС, що виникає у разі збільшення числа процесорів.

КОНТРОЛЬНІ ПИТАННЯ

1. Дайте характеристику рівням паралелізму.
2. Який принцип покладений в основу класифікації архітектур комп'ютерних систем за Флінном?
3. Укажить переваги і недоліки схеми класифікації Флінна.
4. У чому полягають принципи побудови процесора з рознесеною архітектурою?
5. Дайте характеристику продуктивності SIMD-систем як функції їх типу та кількості процесорів.

6. У чому особливість структури векторного процесора?
7. Охарактеризуйте особливості векторних команд.
8. У чому особливість побудови векторно-конвеєрних ЕОМ?
9. Поясніть відмінність між конвеєрними і векторно-конвеєрними обчислювальними системами.
10. Охарактеризуйте структуру матричної комп'ютерної системи.
11. У чому особливість побудови комп'ютерних систем з систолічною структурою?
12. Наведіть класифікацію комп'ютерних систем з систолічною структурою.
13. У чому полягають принципи VLIW-архітектури?
14. Дайте характеристику продуктивності MIMD-систем як функції їх типу та кількості процесорів.
15. Сформулюйте основні характеристики SMP-системи.
16. За якою ознакою обчислювальну систему відносять до архітектури з масовою паралельною обробкою?
17. Яка особливість побудови кластерних обчислювальних систем?
18. Які задачі в кластерній обчислювальній системі покладаються на спеціалізоване програмне забезпечення?
19. У чому особливість побудови потокових комп’ютерних систем?
20. Охарактеризуйте основні показники ефективності обчислювальних систем.
21. Як визначається продуктивність мультипроцесорних систем?
22. Поясніть сутність закону Амдала і наведіть приклади, які пояснюють його обмеження.
23. Яку проблему закону Амдала вирішує закон Густафсона? Сформулюйте області застосування цих законів.

ПІСЛЯМОВА

Шановні читачі, вивчивши матеріал даного посібника, ви стали озброєними базовими знаннями в даній предметній області і, звичайно, ви відкриті новим знанням, тому, що у нас попереду. Це дуже важливо, адже темпи розвитку в цій області знань надзвичайно високі. Спеціаліст з обчислювальних машин і систем повинен бути готовий до навчання впродовж всього професійного життя. Прогрес в області нових комп'ютерних рішень рухається надзвичайно стрімко, тільки встигай відстежувати його тенденції. Спеціалісти прогнозують, що в найближчі роки мікропроцесори Intel будуть працювати на тактовій частоті 10 тисяч МГц. При цьому число транзисторів на кожному такому процесорі досягне мільярда, а обчислювальна потужність 100 мільярдів операцій у секунду. Комп'ютер із процесором такої потужності буде коштувати, як і раніше, біля півтори тисячі доларів і розміщатися на столі. Вже є обнадійливі результати по вакуумній ультрафіолетовій літографії, що дозволить значно зменшити розміри транзисторів. А в лабораторних умовах вже випробувані "балістичні" транзистори, час переключення яких порядку фемтосекунди.

Двадцять перше сторіччя несе масу нововведень в області елементної бази обчислювальних машин і обчислювальних систем, а разом з її змінами переміниться архітектура і організація обчислювальних засобів. Ось короткий список тих новацій, які вже стоять на порозі: голографічна, твердотільна і протонна пам'ять; схеми на базі молекулярних ключів; оптичні, квантові і нано-комп'ютери; електронний цифровий папір; пластмасові дисплеї; нейроінформатика, біоінформатика. І це ще далеко не все. Словом, дорога в комп'ютерне майбутнє відкрита, а інформаційна революція тільки починається. Будьте готові до змін, і все у вас вийде.

Попереду довгий і цікавий шлях пізнань. Удачі вам, шановні читачі, на цьому шляху!

СПИСОК ЛІТЕРАТУРИ

1. Архитектура компьютерных систем и сетей: учеб. пособие / Т.П. Барановская, В.И. Лойко, М.И. Семенов,А.И. Трубилин; Под ред. В.И. Лойко. - М.: Финансы и статистика, 2003. – 256 с.

2. Валецька Т.М. Комп'ютерні мережі. Апаратні засоби: навчальний посібник.-К.: Центр навчальної літератури, 2007. – 208 с.

3. Вычислительные системы, сети и телекоммуникации / В. Л. Бройдо. – СПб.: Питер, 2004. – 703 с.

4. Гук М.Ю. Аппаратные интерфейсы ПК. Энциклопедия. – СПб.: Питер, 2002. – 528 с.

5. Гук М.Ю. Аппаратные средства IBM PC. Энциклопедия. – СПб.:Питер, 2006. – 1072 с.

6. Гук М., Юров В. Процессоры Pentium III, Athlon и другие – СПб: Издательство Питер, 2000. – 480 с.

7. Каган Б.М. Электронные вычислительные машины и системы. –М.: Энергоатомиздат, 1991. – 592 с.

8. Корнеев В.В., Киселёв А.В. Современные микропроцессоры. – 3-е изд. – СПб.: БХВ – Петербург, 2003. – 442 с.

9. Ларионов А.М. Периферийные устройства в вычислительных системах. -М.: Высшая школа, 1991. – 335 с.

10. Локазюк В. М. Мікропроцесори та мікро-ЕОМ у виробничих системах: посібник. – К.: Видавничий центр « Академія », 2002. – 368 с.

11. Мелехин В.Ф. Вычислительные машины, системы и сети: учебник для студ. высш. учеб. заведений /В.Ф. Мелехин, Е.Г. Павловский. – 2-е изд., стер. – М.: Издательский центр «Академия», 2007. – 560 с.

12. Мураховский В. И. Железо ПК. Новые возможности. – СПб.: Питер, 2005. – 592 с.

13. Мюллер Скотт. Модернизация и ремонт ПК. – 15-е изд.: Пер. с англ. – М.: Издательский дом « Вильямс », 2004. – 1344 с.

14. Олифер В.Г., Олифер Н.А. Компьютерные сети. Принципы, технологии, протоколы: учебник для вузов. 3 – е изд. – СПб.: Питер, 2007. – 960 с.

15. Организация ЭВМ. –5-е изд. /К. Хамахер, З. Вранешич, С. Заки. – СПб.: Питер; Киев: Издательская группа BHV, 2003. – 848 с.

16. Поворознюк А.И. Архитектура компьютеров. Архитектура микропроцессорного ядра и системных устройств: учеб. пособие. Ч.1. – Харьков: Торнадо, 2004. – 355 с.

17. Рябенький В.М., Жуйков В.Я., Гулий В.Д. Цифрова схемотехніка: навч. посібник. – Львів: «Новий світ – 2000», 2009. – 736 с.

18. Стивен Бигелоу. Устройство и ремонт персонального компьютера. Аппаратная платформа и основные компоненты. – 2-е изд.: пер. с англ. – М.:ООО « Бином – Пресс », 2005. – 976 с.

19. Схемотехника електронних систем: в 3 кн. Кн.3. Микропроцессоры и микроконтроллеры: учебник /В.И. Бойко, А.М. Гуржий, В.Я. Жуйков и др. – СПб.:БХВ - Петербург, 2004. – 455 с.

20. Тарарака В.Д. Основы вычислительной техники и программирование. Ч.2. Основы вычислительной техники: учебное пособие. – Житомир: ЖВУРЭ ПВО, 1992. – 500 с.

21. Тарарака В.Д. Обчислювальна техніка. Ч.І. Основи побудови ЕОМ: навчальний посібник. – Житомир: ЖВІРЕ, 2003. – 348 с.

22. Тарарака В.Д. Обчислювальна техніка. Ч. ІІ. Апаратні засоби персональних комп'ютерів: навчальний посібник. - Житомир ЖВІРЕ, 2004. – 308 с.

23. Уоссермен Ф. Нейрокомпьютерная техника: Теория и практика. – М.: Мир, 1992. – 184 с.

24. Хмелевский И.В., Битюцкий В.П. Организация ЭВМ и систем: учебное пособие. – Екатеринбург: ГОУ ВПО УГТУ-УПИ, 2005. – 98 с.

25. Цилькер Б.Я., Орлов С.А. Организация ЭВМ и систем: ученик для вузов. - СПб.: Питер, 2004. – 668 с.

26. Чистяков В.Д. Анатомия ПК. Все о компьютерном железе. – М.: НТ Пресс, 2007. – 160 с.

27. Э. Таненбаум. Архитектура компьютера. – СПб.: Питер, 2007. – 848 с.

Навчальне видання

Тараракa Валерій Дмитрович

АРХІТЕКТУРА КОМП'ЮТЕРНИХ СИСТЕМ

Навчальний посібник

Відповідальний редактор **Ю.А. Подчашинський**
Комп'ютерний набір та верстка **В.Д. Тараракa**
Макетування **В.Д. Тараракa**

Гарнітура Times New Roman