

Розділ 3

Порядок виконання команд і програм в комп'ютері

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

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

3.1. Кодування та виконання команд в комп'ютері

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

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

Розглянемо як це відбувається в комп'ютері детальніше.

3.1.1. Кодування команди та програми

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

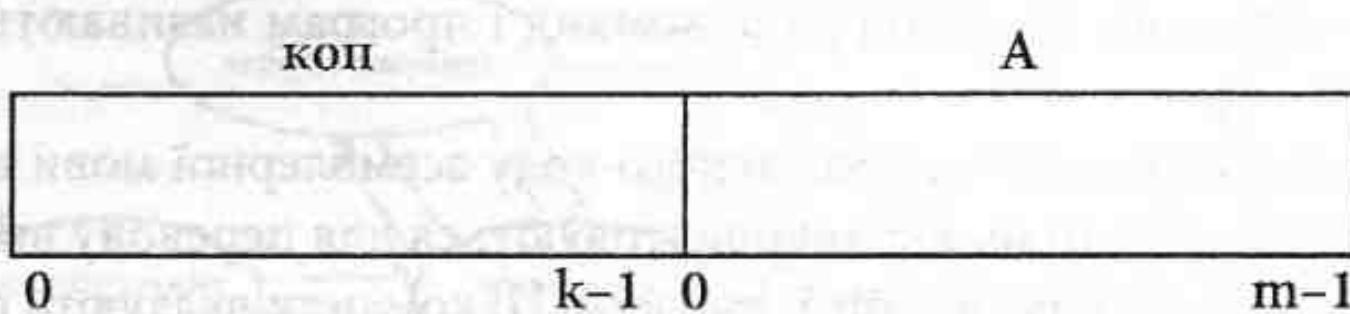


Рис. 3.1. Кодування команди

Поле коду операції (КОП) займає k розрядів. Ним може бути закодовано до $N = 2^k$ різних операцій. Кількість двійкових розрядів, які відводяться під код операції, вибирається таким чином, щоб ними можна було закодувати всі виконувані в даному комп'ютері операції. Якщо деякий комп'ютер може виконувати N_c різних операцій, то мінімальна розрядність поля коду операції k визначається наступним чином: $k = \lceil \log_2 N_c \rceil$, де вираз в дужках означає заокруглення до більшого цілого.

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

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

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

Використання вісімкового і шістнадцяткового кодування дозволяє скоротити записи і спростити роботу програміста. Як відомо 3 двійкових розряди (тріада) замінюються на 1 вісімковий, а 4 двійкових розряди (тетрада) – на 1 шістнадцятковий. Приклад: $(00001111111)_2 = (0377)_8 = (0FF)_{16}$

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

ADD – додати (add),

SUB – відняти (subtract),

MPLY – перемножити (multiply),

DIV – поділити (divide),

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

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

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

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

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

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

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

3.1.2. Порядок виконання команд

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

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

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

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



Рис. 3.2. Діаграма циклу виконання команди

3.1.3. Виконання команд на рівні регістрів процесора

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

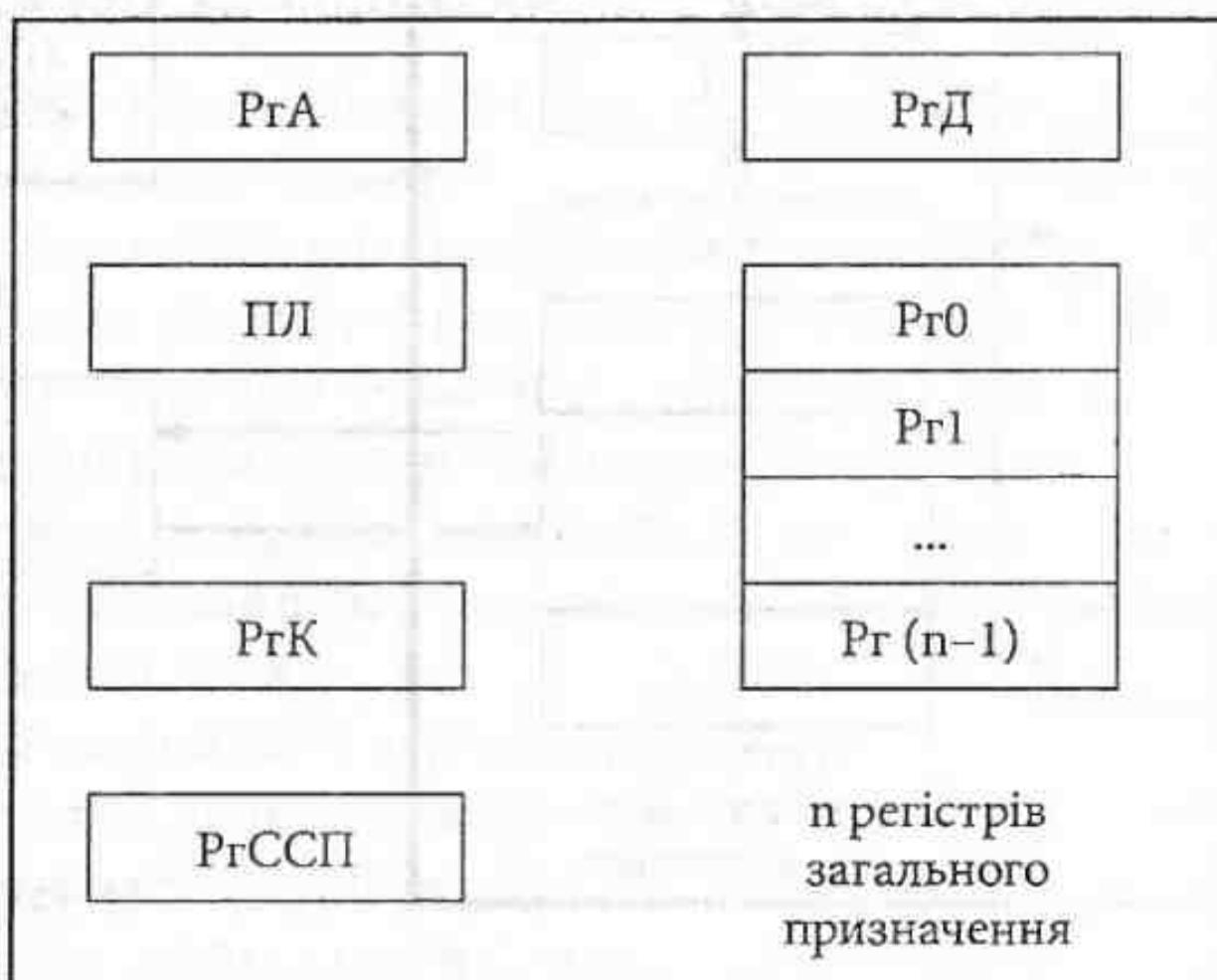


Рис. 3.3. Регістрова пам'ять процесора

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

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

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

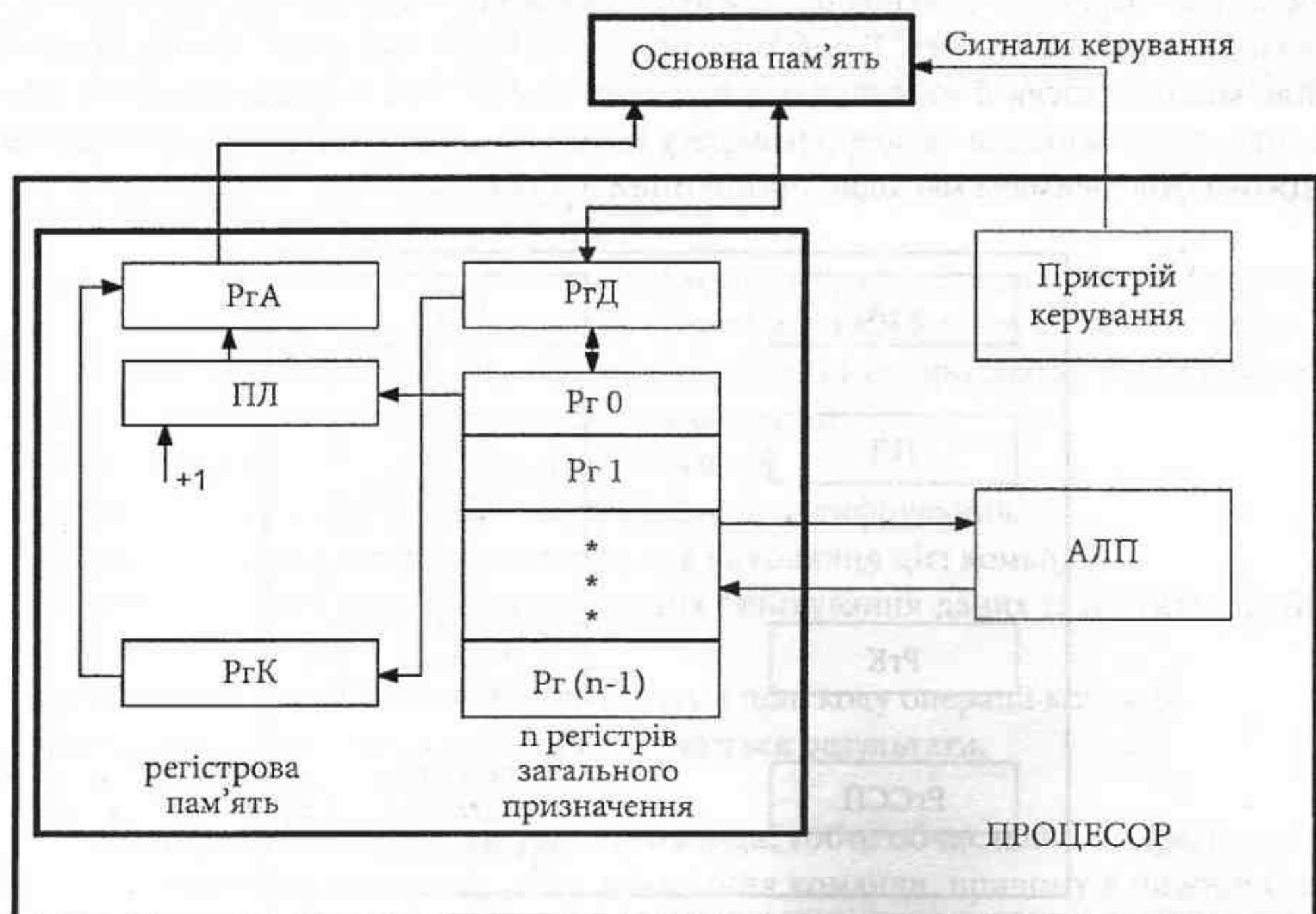


Рис. 3.4. Зв'язки між процесором і основною пам'яттю

Виходячи з наведеної вище інформації про вузли процесора та його зв'язки з основною пам'яттю, розглянемо детальніше виконання команд на прикладі виконання операції додавання слова, яке знаходитьться в пам'яті за адресою 9, з вмістом регістра Рг0 регістрової пам'яті процесора, коли результат операції засилается в пам'ять за адресою 9. Програма обчислень знаходиться також в пам'яті. Операцію можна записати наступним чином:

$$[\text{Комірка } 9] := [\text{Комірка } 9] + [\text{Рг0}].$$

Послідовність дій при виконанні цієї операції буде наступною:

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

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

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

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

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

6. Рг Д := [РгД] + [Рг0]. Виконання в АЛП операції [РгД] + [Рг0] і засилання результата в РгД.

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

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

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

3.2. Типи операцій та команд

3.2.1. Класифікація команд за типами операцій

Команди можуть бути класифіковані відповідно до ініційованих ними типів операцій.

Команди обробки даних ініціюють:

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

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

- операції переміщення даних вregistрах та стеках над скалярними та векторами;
- команди вибірки даних з пам'яті та запам'ятування даних в пам'яті;
- команди вибірки адрес з пам'яті та запам'ятування адрес в пам'яті;
- команди вибірки команд з пам'яті та запам'ятування команд в пам'яті.

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

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

Розглянемо деякі типи команд детальніше.

3.2.2. Команди обробки даних

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

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

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

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

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

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

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

Таблиця 3.1

№ пп	Код команди	Функції команд комп'ютера DLX
		Арифметичні та логічні
1	ADD, ADDI, ADDU, ADDUI	Додати, додати безпосереднє дане (immediate), з знаком та без знаку
2	SUB, SUBI, SUBU, SUBUI	Відняти, відняти безпосереднє дане, з знаком та без знаку
3	MULT, MULTU, DIV, DIVU	Перемножити та поділити, з знаком та без знаку
4	AND, ANDI	Додати, додати безпосереднє дане

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

3.2.3. Команди переміщення даних

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

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

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

Таблиця 3.2

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

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

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

Таблиця 3.3

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

3.2.4. Команди передачі керування

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

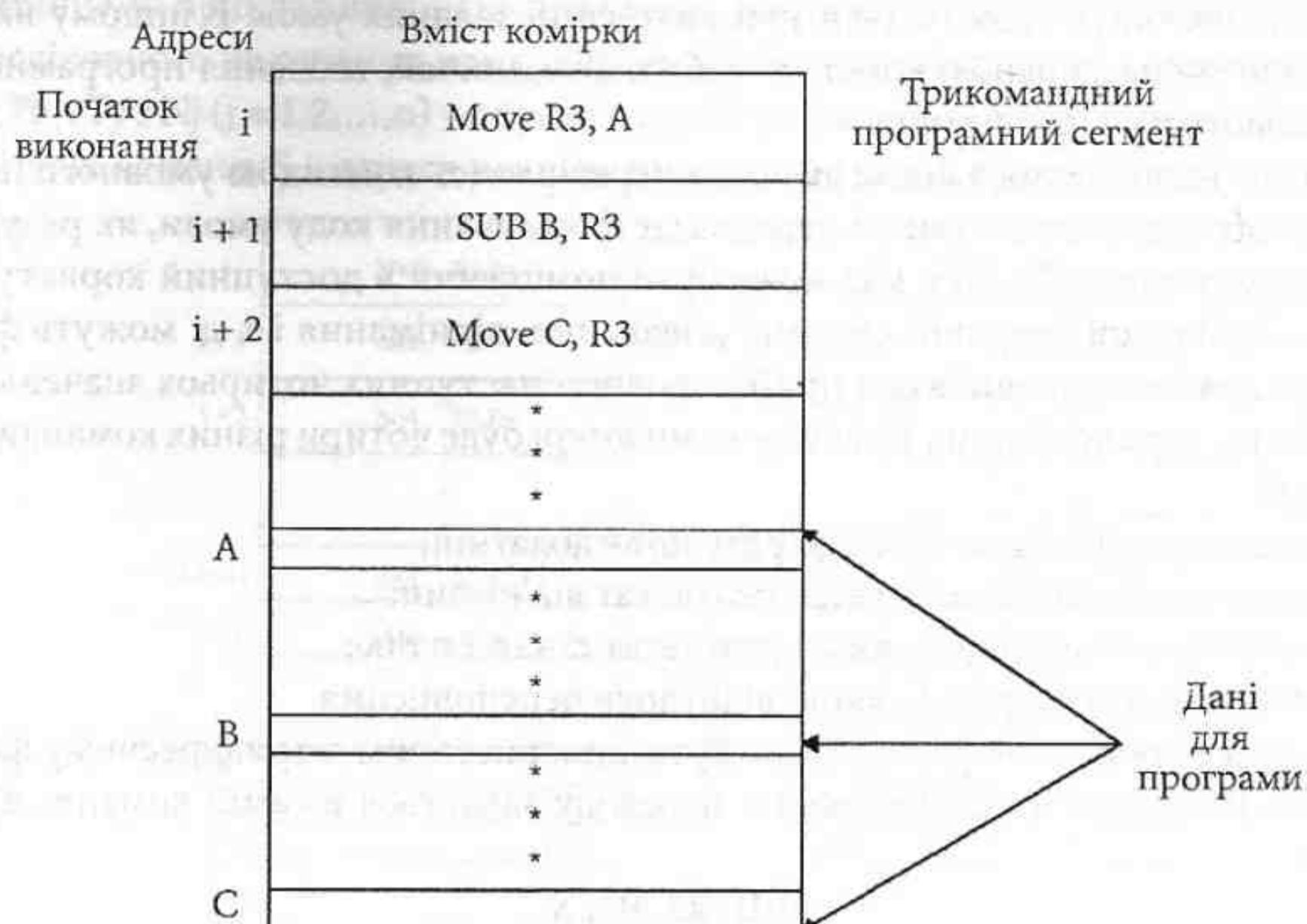


Рис. 3.5. Програма виконання операції $(C) := (A) - (B)$

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

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

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

Розглянемо організацію виконання основних команд передачі керування детальніше.

3.2.4.1. Команди переходу

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

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

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

BRP X – перехід до комірки X, якщо результат додатній;

BRN X – перехід до комірки X, якщо результат від'ємний;

BRZ X – перехід до комірки X, якщо результат рівний нулю;

BRO X – перехід до комірки X, якщо відбулось переповнення.

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

BRE R3, R4, X

задає перехід до комірки X, якщо вміст регістра R3 дорівнює вмісту регістра R4.

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

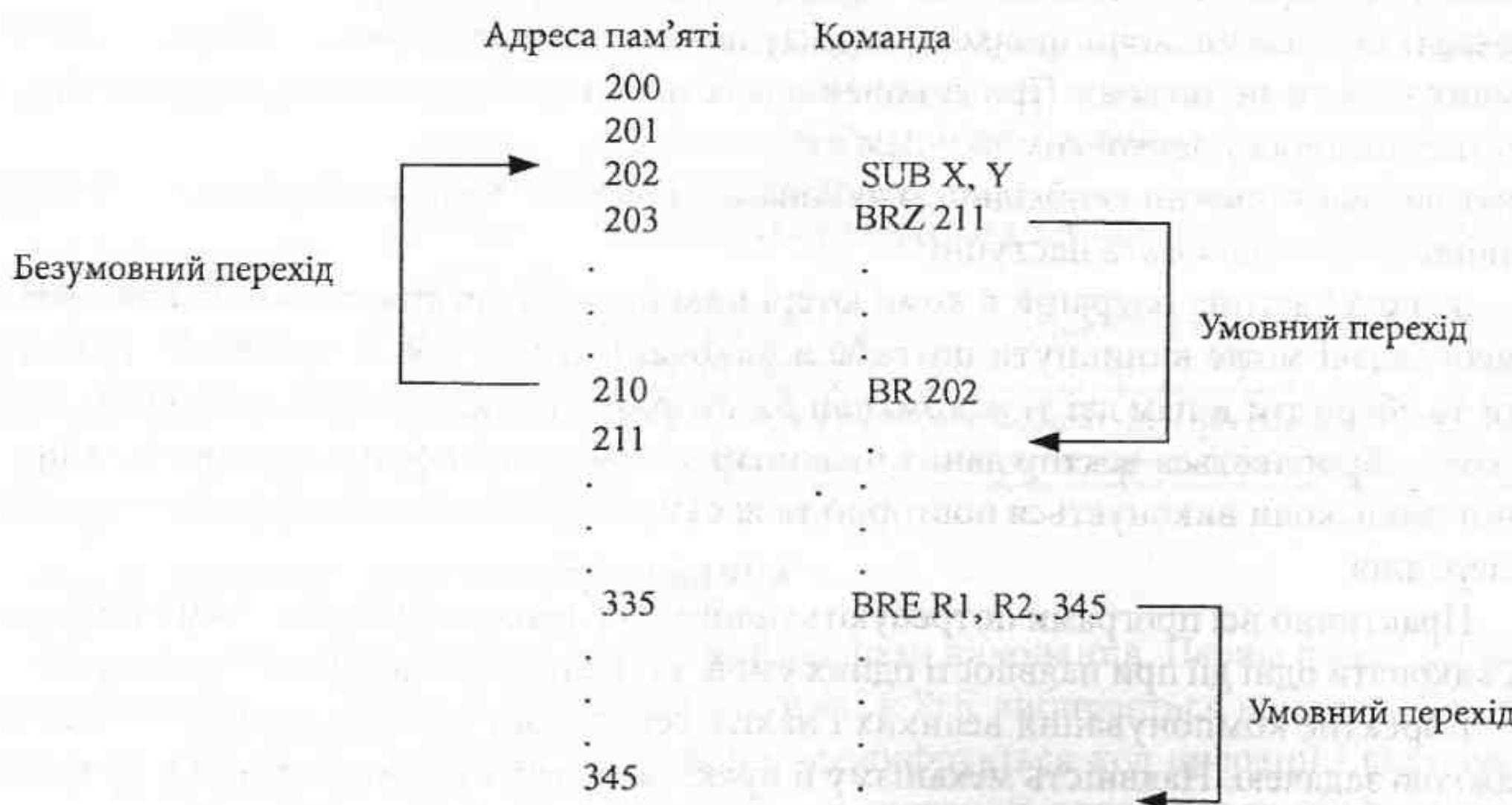


Рис. 3.6. Приклад операцій умовного та безумовного переходу

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

Розглянемо приклад виконання задачі додавання n чисел, яку можна виконати шляхом послідовного виконання команд (рис. 3.7a), або з використанням команд переходів (рис. 3.7b). Тут N_j ($j = 1, 2, \dots, n$) – адреса в пам'яті кожного із n чисел, i ($i=1, 2, \dots, n$) – адреси розміщення команд, S – адреса розміщення результатуючої суми.

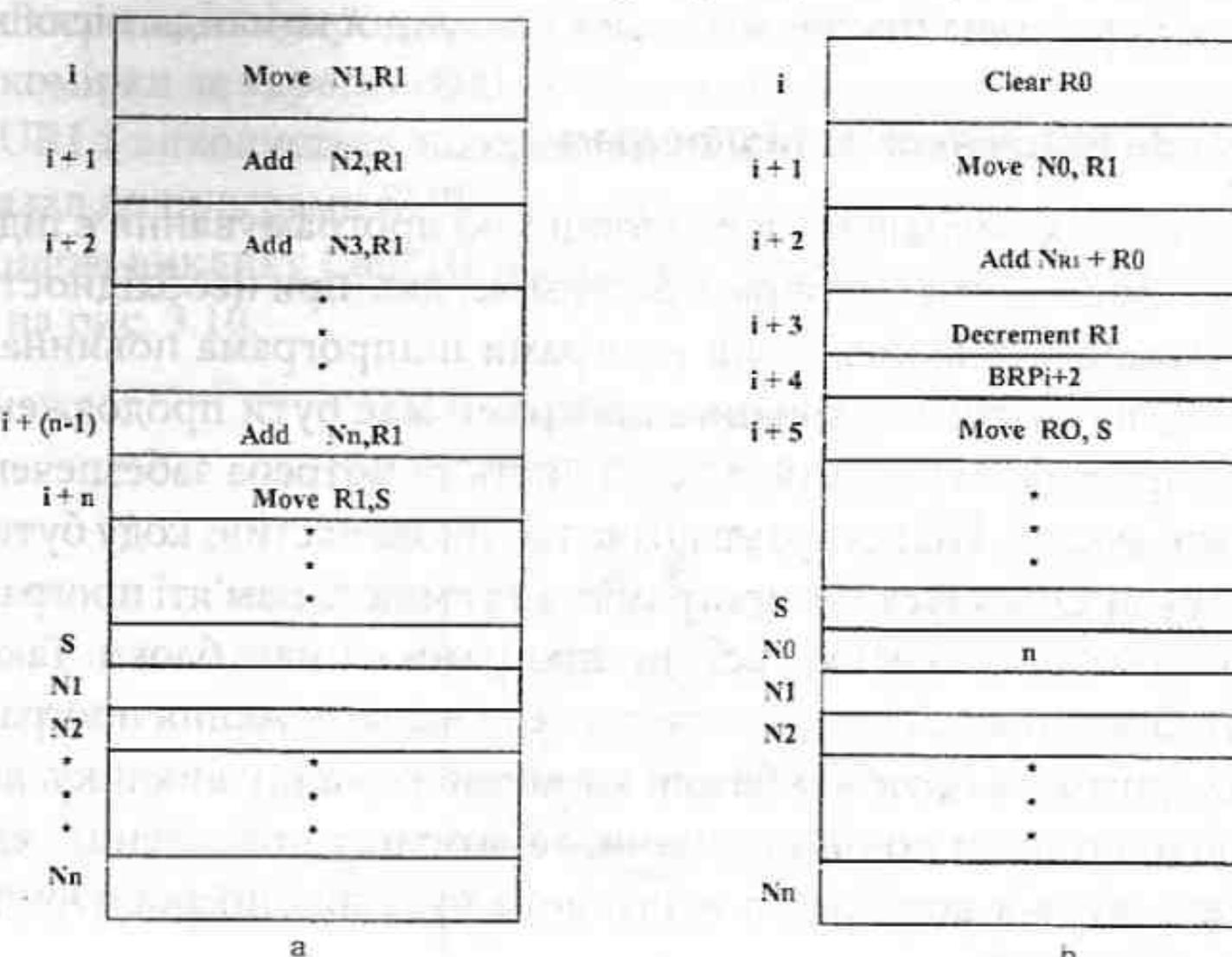


Рис. 3.7. Дві програми додавання n чисел шляхом послідовного виконання команд (a) та з використанням переходів (b)

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

3.2.4.2. Команди пропуску

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

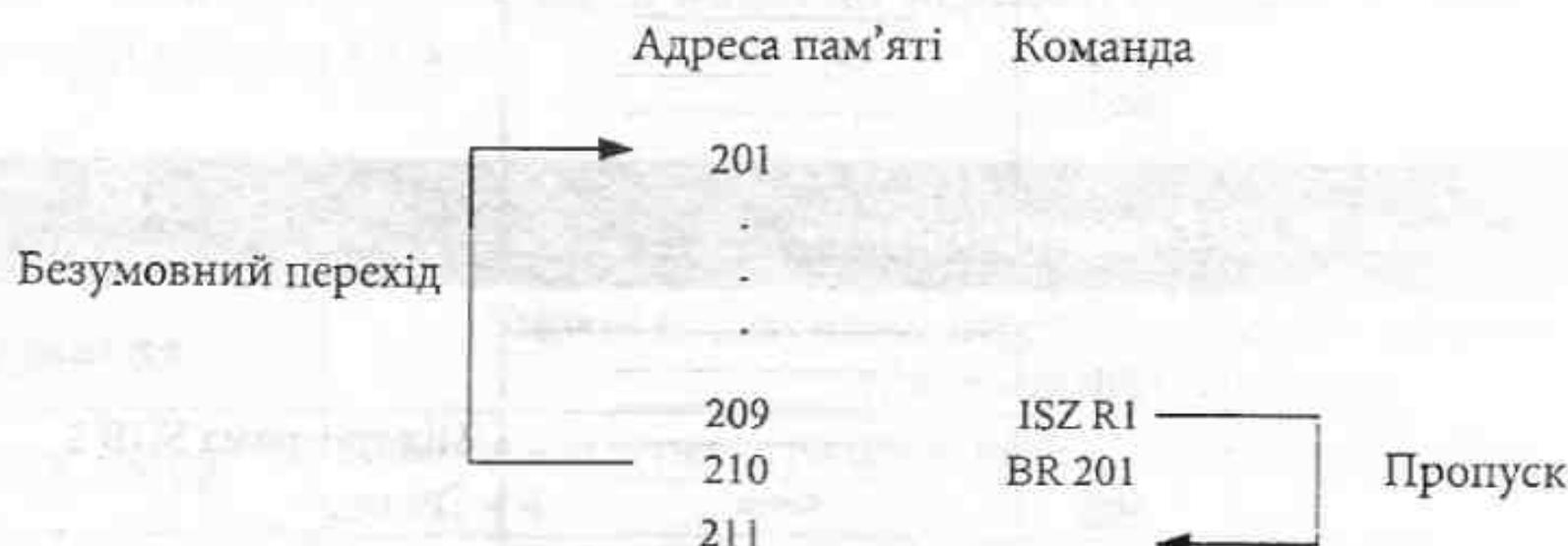


Рис. 3.8. Приклад використання команди пропуску

В цьому фрагменті використано команду пропуску для реалізації ітераційного циклу. В реєстрі R1 встановлюється від'ємне число, модуль якого рівний кількості необхідних ітерацій. В кінці циклу до вмісту реєстра R1 додається одиниця. Якщо отримане значення не рівне нулю, програма переходить на початок циклу. В іншому випадку переході пропускається і програма продовжується з наступної команди після кінця циклу.

3.2.4.3. Команди звернення до підпрограм

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

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

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

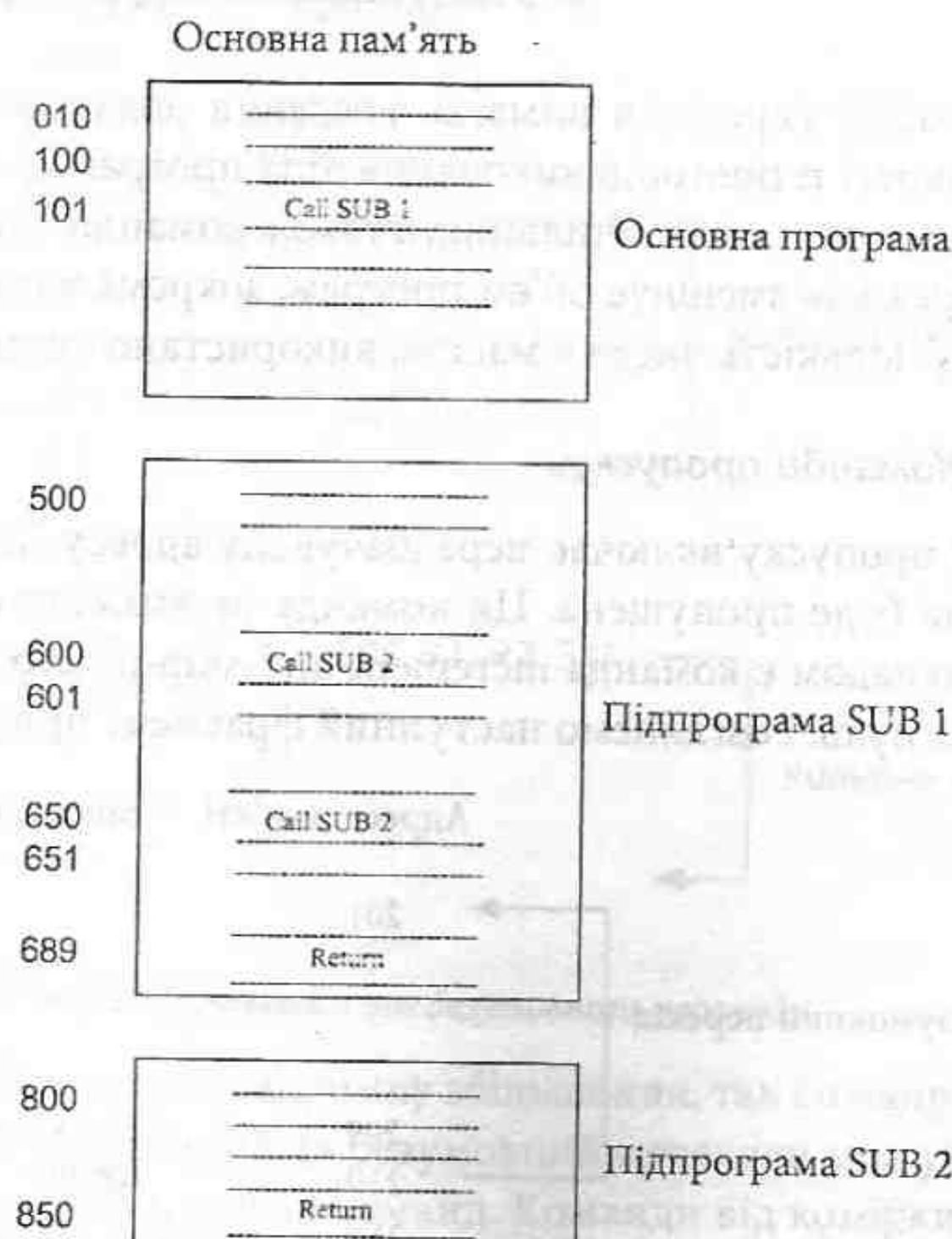


Рис. 3.9. Вкладені підпрограми

На цьому прикладі основна програма починається з комірки за адресою 010. Ця програма включає звернення до підпрограми SUB1, яка починається з комірки за адресою 500. Коли виконується ця команда виклику, процесор призупиняє виконання основної програми і починає виконання підпрограми SUB1 шляхом вибору наступної команди з комірки 500. Всередині підпрограми SUB1 є два звернення до підпрограми SUB2, яка починається з комірки за адресою 800. В обох цих випадках призупиняється виконання підпрограми SUB1 і виконується підпрограма SUB2. Команда Return вказує процесору повернутись назад до програми SUB1 та продовжити виконання команди, яка іде за відповідною командою виклику Call. Ці звернення між основною програмою та підпрограмами показані на рис. 3.10.



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

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

Таблиця 3.4

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

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

3.2.5. Команди введення-виведення

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

3.2.6. Принципи формування системи команд комп’ютера

При розробці комп’ютера необхідно враховувати багато особливостей вибору системи команд. З одного боку система команд повинна бути функціонально повною, тобто комп’ютер повинен забезпечувати виконання всіх заданих функцій. З іншого боку система команд має бути ортогональною, тобто не повинна бути надлишковою. Для нового комп’ютера, який є розширенням відповідної серії, першочерговою є сумісність

програм, які виконуються на одному комп’ютері, повинні виконуватись і на іншому комп’ютері. Для збільшення терміну використання важливим є розширення адресного діапазону та однорідності адресного простору. І, нарешті, важливим є розмір та організація самого формату команди. Розглянемо ці особливості детальніше.

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

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

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

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

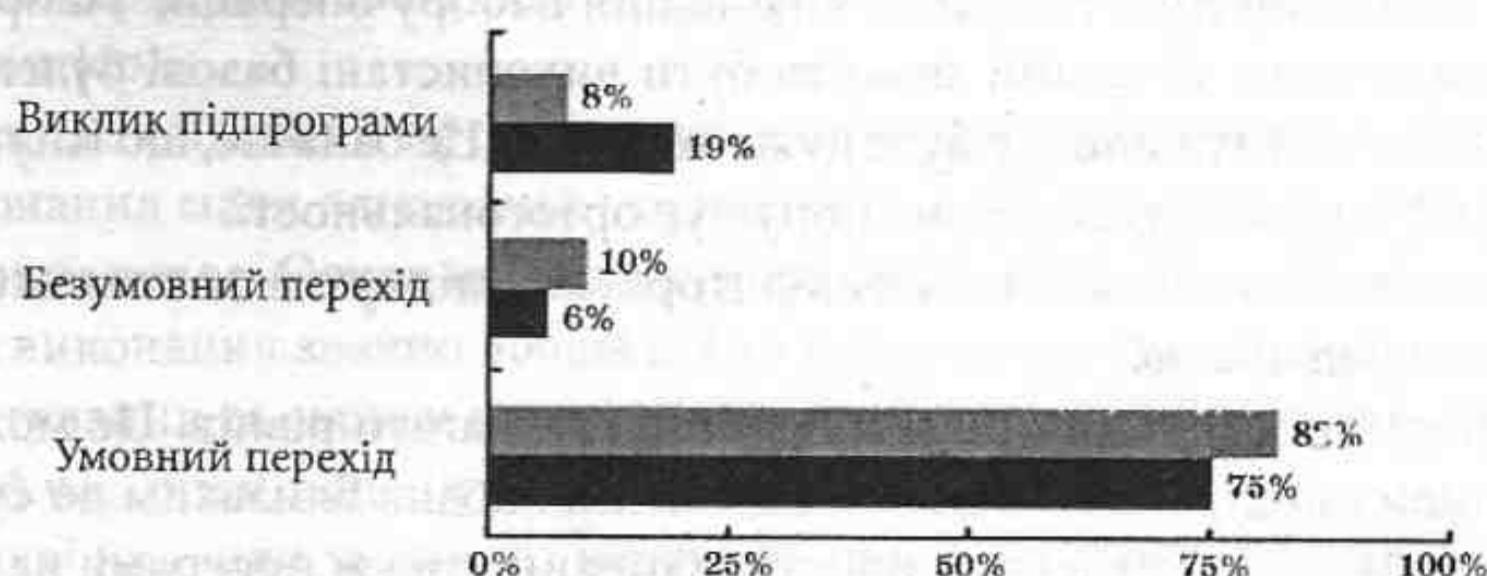


Рис. 3.11. Середня частота використання команд передачі керування

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

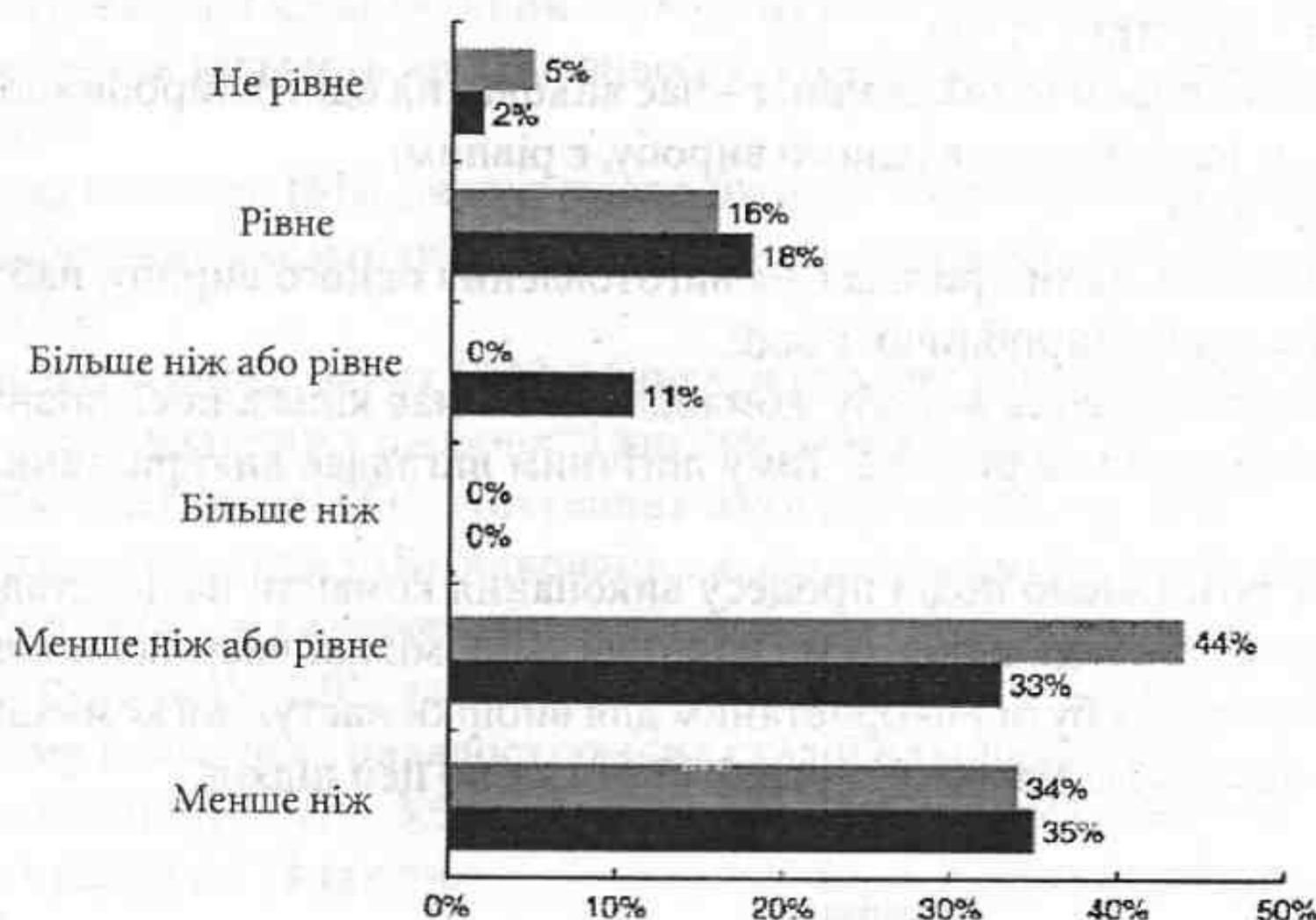


Рис. 3.12. Середня частота використання різних типів команд умовного переходу

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

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

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

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

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

3.2.7. Конвеєрне виконання команд

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

$$T = tm + t(n - 1) = t(m + n - 1),$$

де m – кількість виробничих стадій, t – час виконання однієї виробничої стадії, а час, який припадає на виготовлення одного виробу, є рівним:

$$T_b = t(m + n - 1)/n.$$

При $n \gg m$ час T_b , який припадає на виготовлення одного виробу, наближається до часу t виконання однієї виробничої стадії.

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

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

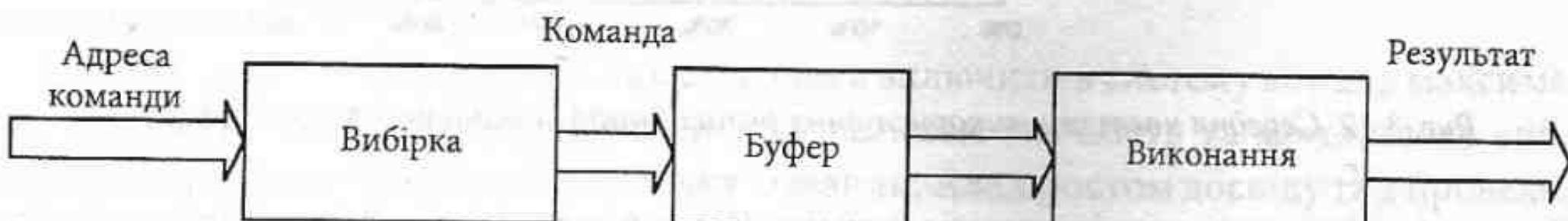


Рис. 3.13. Двоярусний конвеєр виконання команди

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

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

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

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

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

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

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

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

Таблиця 3.5

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

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

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

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

Декілька інших факторів обмежують ріст продуктивності за рахунок використання конвеєра:

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

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

Таблиця 3.6

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

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

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

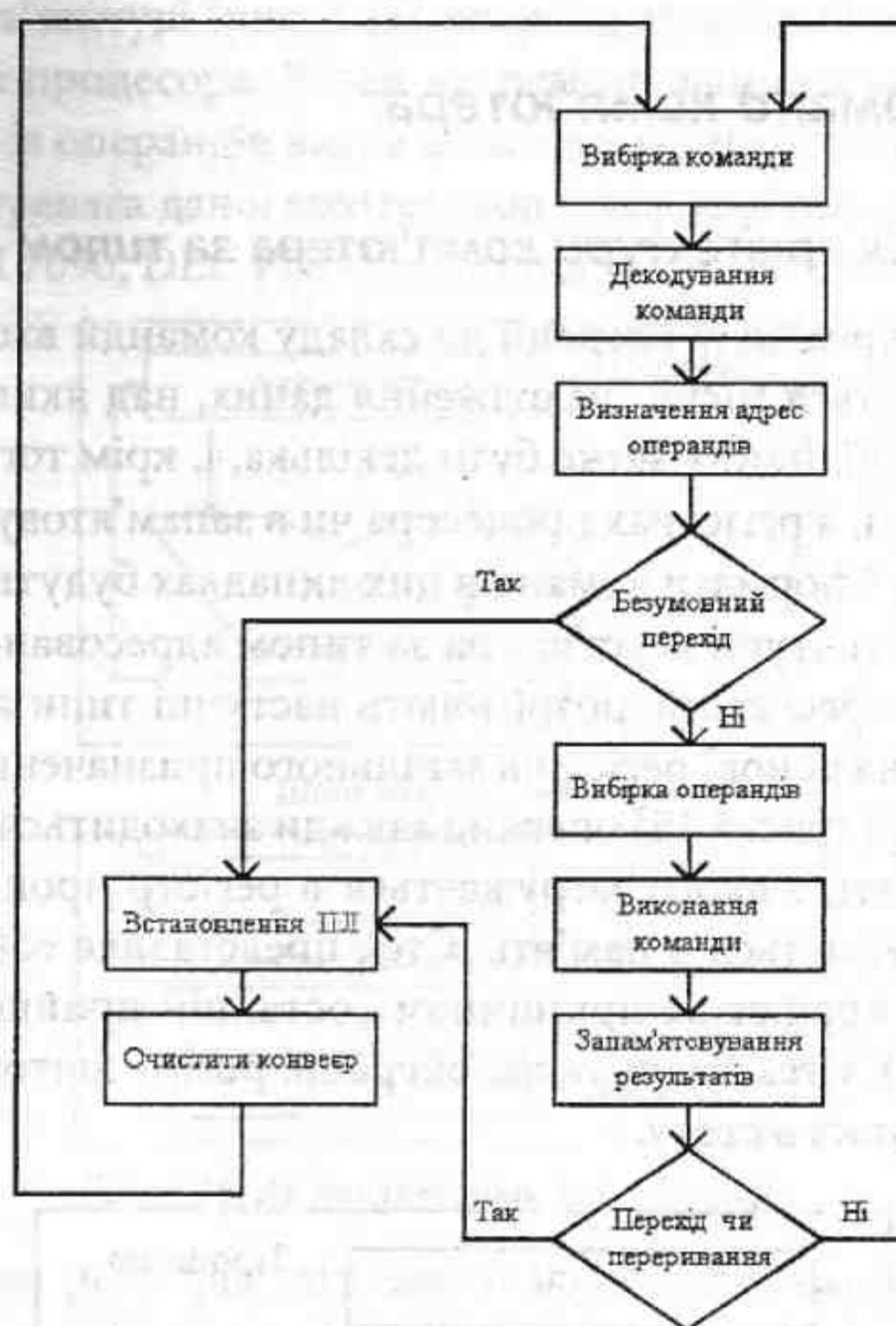


Рис. 3.14. Шестиярусний конвеєр команд

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

Таким чином, конфлікти конвеєра можуть бути трьох типів:

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

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

3.3. Формати команд комп'ютера

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

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

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

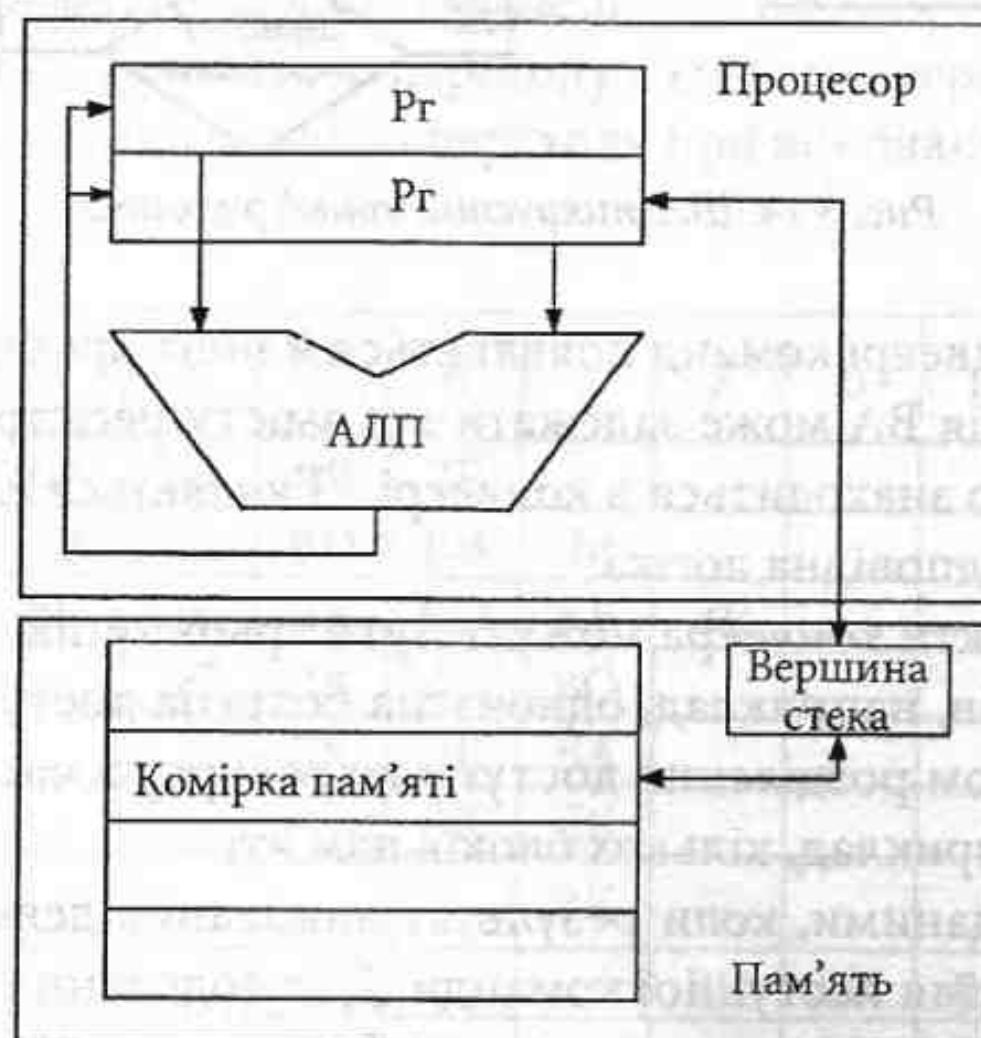


Рис. 3.15. Стекова архітектура

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

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

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



Рис. 3.16. Акумуляторна архітектура

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

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

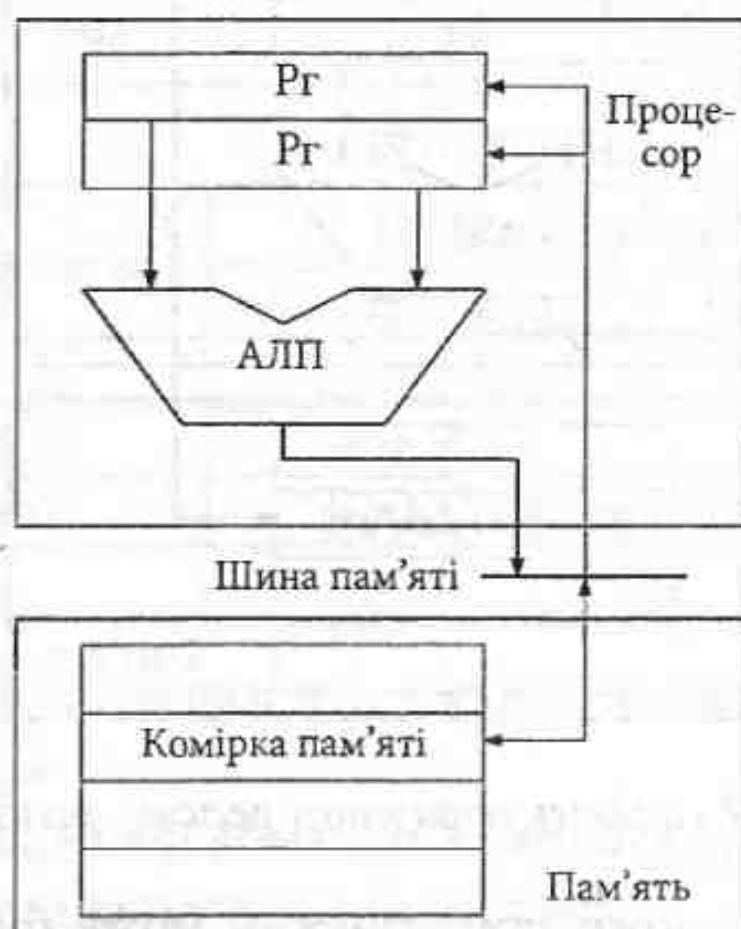


Рис. 3.17. Архітектура типу пам'ять-пам'ять

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



Рис. 3.18. Архітектура типу регістр-пам'ять

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



Рис. 3.19. Архітектура типу регістр-регистр

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

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

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

3.3.2. Порівняльний аналіз форматів команд

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

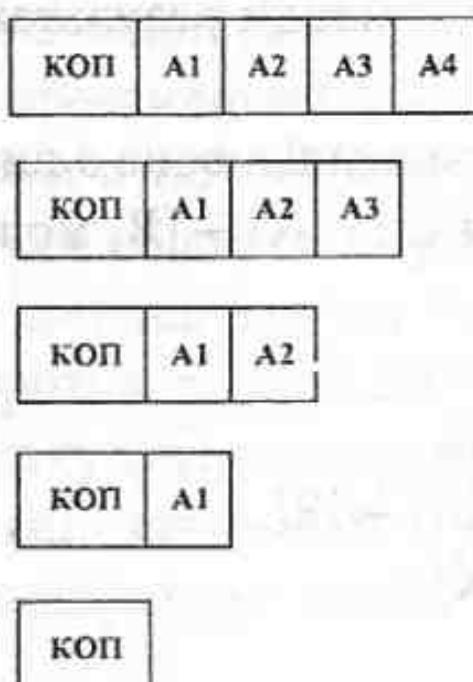


Рис. 3.20. Формати команд комп'ютерів

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

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

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

Таблиця 3.7

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

Тут А, В і С – адреси комірок пам'яті, в яких знаходяться числа А, В, С; R1, R2 і R3 – адреси регистрів процесора.

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

ADD A,B,C.

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

ADD A,B, яка означає $[A] := [A] + [B]$,

MOVE A,C, яка означає $[C] := [A]$,

або дві команди:

MOVE B,C, яка означає $[C] := [B]$,

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

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

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

LOAD A, яка означає $AKK := [A]$,

ADD B, яка означає $AKK := [AKK] + [B]$,

STORE C, яка означає $C := [AKK]$.

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

LOAD A,Ri, яка означає $Pr_i := [A]$,

ADD B,Ri, яка означає $Pr_i := [Pr_i] + [B]$,

STORE Ri,A, яка означає $[A] := [Pr_i]$.

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

ADD Ri,Rj, яка означає $Rj := Ri + Rj$.

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

Головні критерії вибору формату команд:

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

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

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

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

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

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

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

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

Приклади комп'ютерів, в яких використовуються описані архітектури:

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

3.4. Способи адресації operandів

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

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

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

та автодекрементну, сторінкову, стекову. Практично у всіх існуючих комп'ютерах використовується один або декілька з цих способів адресації. Тому в команду потрібно вводити спеціальні ознаки з тим, щоб пристрій керування міг розпізнати використаний спосіб. Це можуть бути додаткові розряди в команді, або для різних типів команд застосовуються різні способи адресації. Варіант формату команди з полями для розпізнавання типу адресації представлено на рис. 3.21, де TA1 – поле, яке вказує тип адресації, який використано в полі A1, а TA2 – поле, яке вказує тип адресації, який використано в полі A2.

КОП	TA1	A1	TA2	A2
-----	-----	----	-----	----

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

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

3.4.1. Безпосередня адресація

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

КОП	Операнд
-----	---------

Рис. 3.22. Безпосередня адресація

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

3.4.2. Пряма адресація

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

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

В другому випадку адреса R регістра регистрового файлу процесора із адресної частини АЧ команди поступає на адресні входи регистрового файла процесора (рис. 3.23 б), і у вказаній реєстр в режимі запису записується операнд із регістра даних РгД, а в режимі зчитування зчитується операнд в регістр даних РгД.

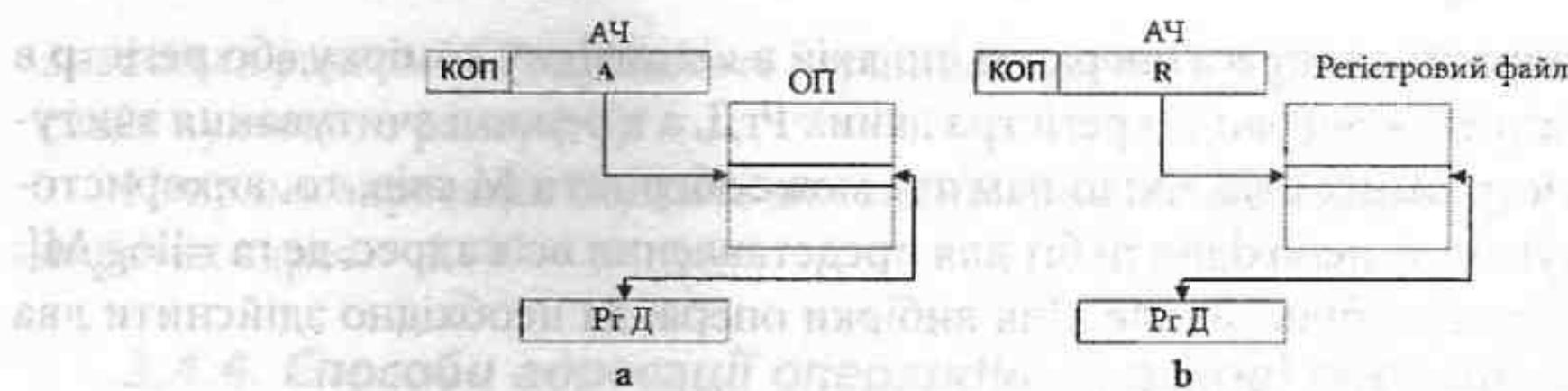


Рис. 3.23. Пряма адресація основної пам'яті а) та регістрового файлу процесора б)

Якщо основна пам'ять може зберігати M слів, то, використовуючи двійкове кодування, необхідно m біт для представлення адрес всіх комірок пам'яті, де $m = \lceil \log_2 M \rceil$. Значення в дужках означає більше ціле. Якщо регістровий файл має N регістрів, то, використовуючи двійкове кодування, необхідно n біт для представлення адрес всіх регістрів, де $n = \lceil \log_2 N \rceil$. Оскільки число регістрів значно менше кількості комірок пам'яті, то і розрядність адреси для їх адресації буде значно меншою, а відповідно значно меншою буде і розрядність команди в цілому. Нехай для прикладу кількість виконуваних в комп'ютері команд рівна 256, тобто розрядність коду операції рівна 8 бітів, ємність основної пам'яті рівна 1ГБ, тобто розрядність адреси рівна 30 бітів, а кількість регістрів регістрового файла процесора рівна 64, тобто розрядність адреси рівна 6 бітів. На рис. 3.24 показано формати двоадресних команд при прямій адресації основної пам'яті та регістрового файла процесора для наведеного прикладу.

8	30	30
КОП	A1	A2
8	6	6
КОП	R1	R2

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

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

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

КОП	ТП	Адреса
-----	----	--------

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

3.4.3. Непряма адресація

При непрямій адресації в адресному полі вказується місце розміщення адреси операнда, а виконавча адреса знаходиться наступним чином: $A = [A^1]$, де A^1 – адреса комірки пам'яті, в якій зберігається виконавча адреса. Адреса A із адресної частини АЧ команди поступає на адресні входи основної пам'яті ОП (рис. 3.26), з відповідної комірки осно-

вної пам'яті ОП вибирається адреса операнда, по якій в відповідну комірку або регистр в режимі запису записується операнд із регистра даних РгД, а в режимі зчитування зчитується операнд в регистр даних РгД. Якщо пам'ять може зберігати M слів, то, використовуючи двійкове кодування, необхідно m біт для представлення всіх адрес, де $m = \lceil \log_2 M \rceil$. Значення в дужках означає більше ціле. Для вибірки операнда необхідно здійснити два звернення до ОП.

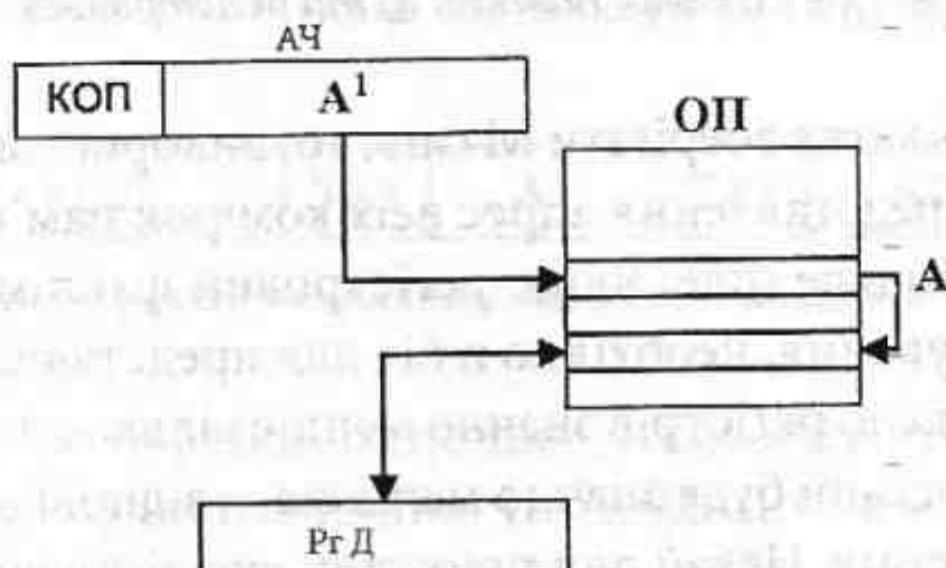


Рис. 3.26. Непряма адресація основної пам'яті, яка вимагає здійснення двох звернень

Для зберігання адрес операндів можна використати регістровий файл процесора (рис. 3.27). Якщо регістровий файл може зберігати N слів, то, використовуючи двійкове кодування, необхідно n біт для представлення непрямої адреси в адресній частині команди, де $n = \lceil \log_2 N \rceil$, а розрядність регістрів буде рівною $m = \lceil \log_2 M \rceil$. Значення в дужках означає більше ціле. Для вибірки операнда необхідно здійснити одне звернення до регістрової пам'яті і одне звернення до основної пам'яті ОП. Такий підхід дозволяє при малій розрядності адресної частини команди n звертатися до пам'яті великої ємності маючи велику розрядність регістрів m .

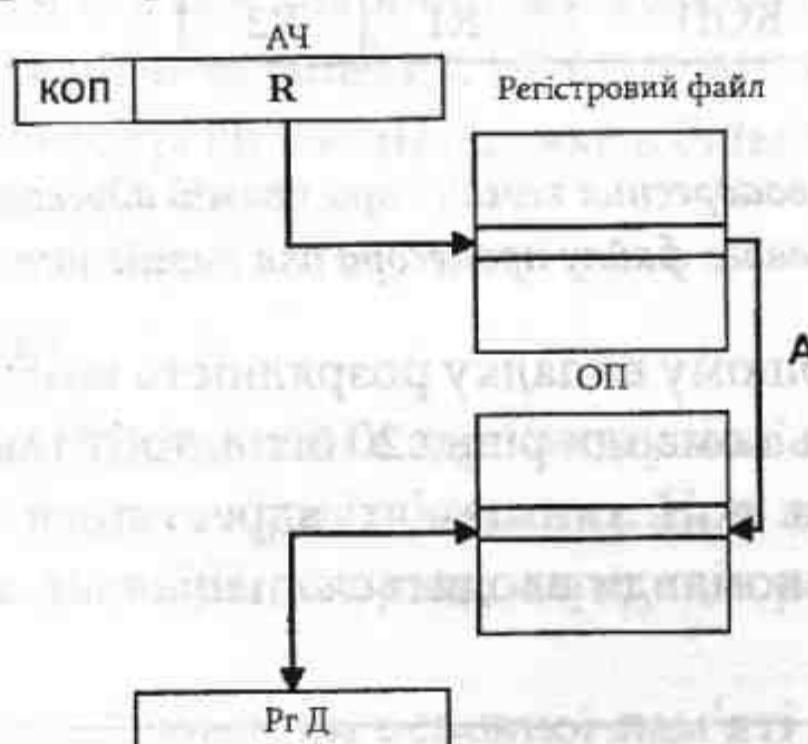


Рис. 3.27. Непряма адресація основної пам'яті з використанням регістрового файлу процесора

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

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

зивається рангом r_i . Розрізняють перший, другий і т. д. ранги. Пряма адресація – це адресація нульового рангу (r_0).

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

3.4.4. Способи адресації операндів на основі операції зміщення

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

3.4.4.1. Відносна адресація

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

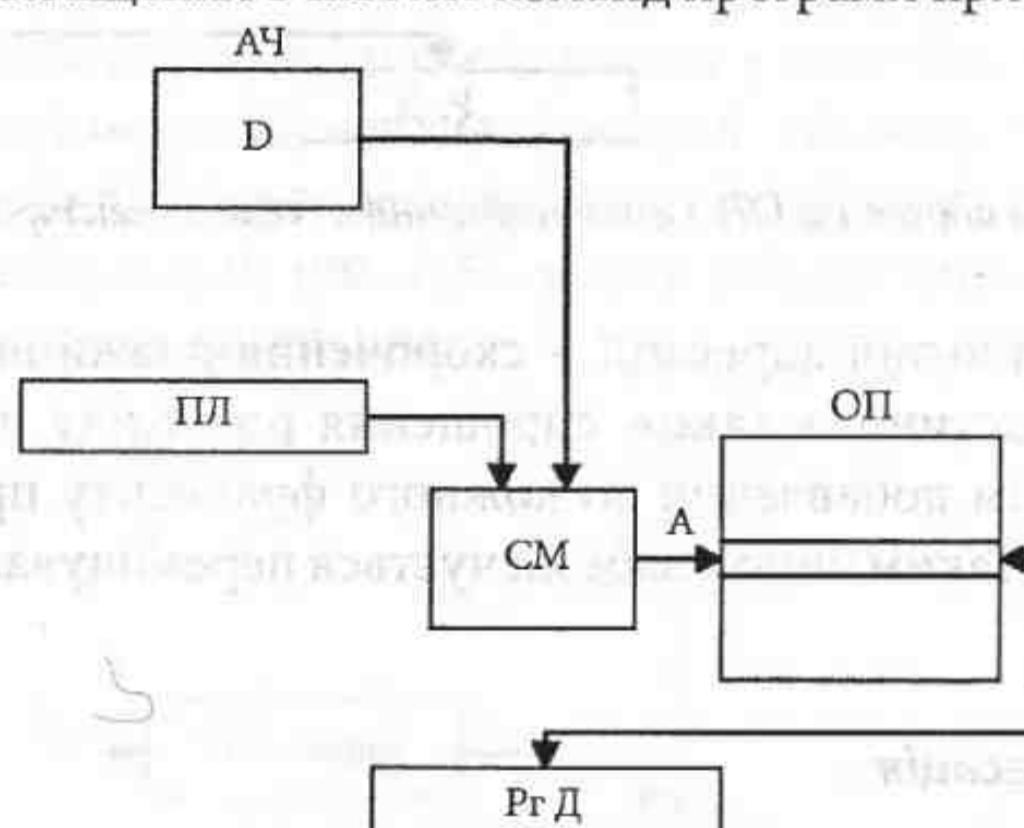


Рис. 3.28. Адресація основної пам'яті з використанням відносної адресації

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

3.4.4.2. Базова адресація

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

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

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

Якщо основна пам'ять може зберігати M слів, регістровий файл процесора може зберігати N слів, а сегмент має розмір L слів, то, використовуючи двійкове кодування, поле B буде займати n біт, де $n = \lceil \log_2 N \rceil$, поле D буде займати l біт, де $l = \lceil \log_2 L \rceil$, а розрядність регістрів буде рівною $m = \lceil \log_2 M \rceil$. Значення в дужках означає більше ціле.

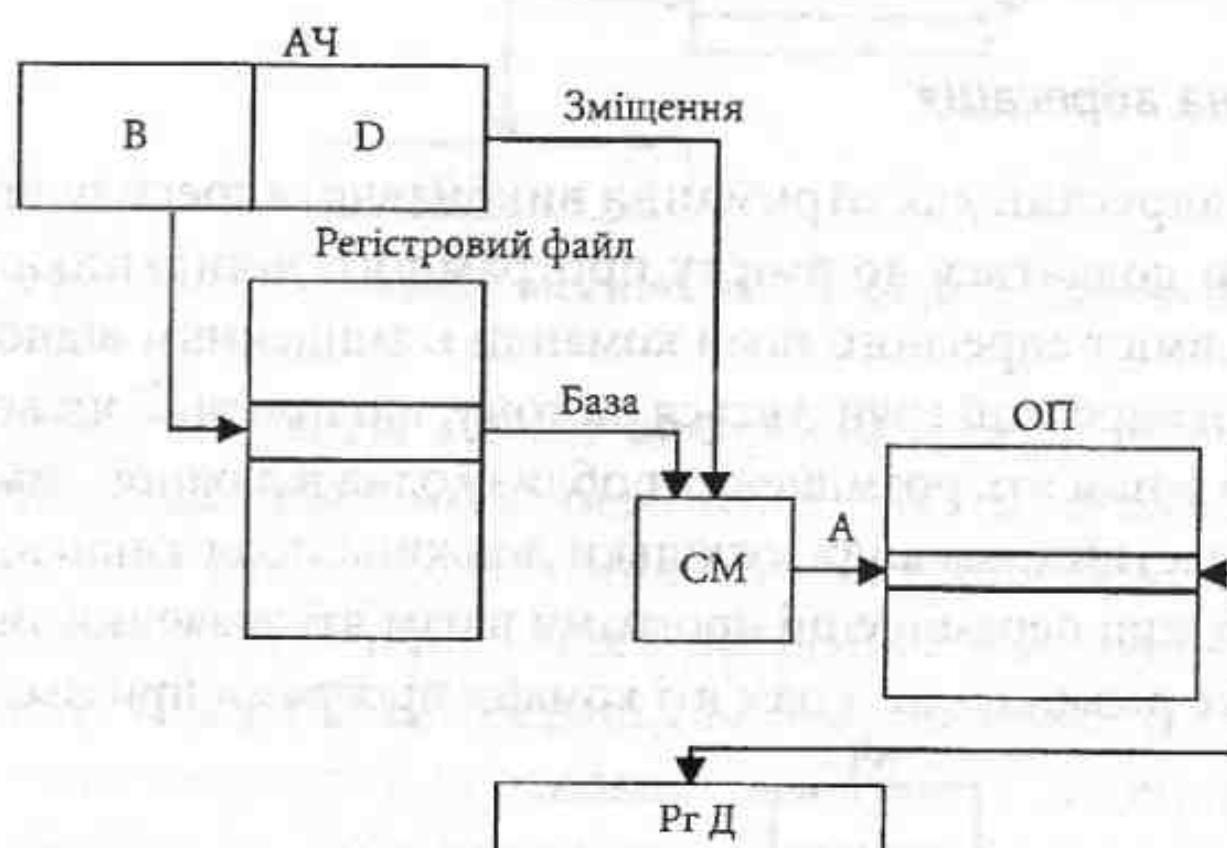


Рис. 3.29. Базова адресація ОП з використанням бази із регістрової пам'яті

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

3.4.4.3. Індексна адресація

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

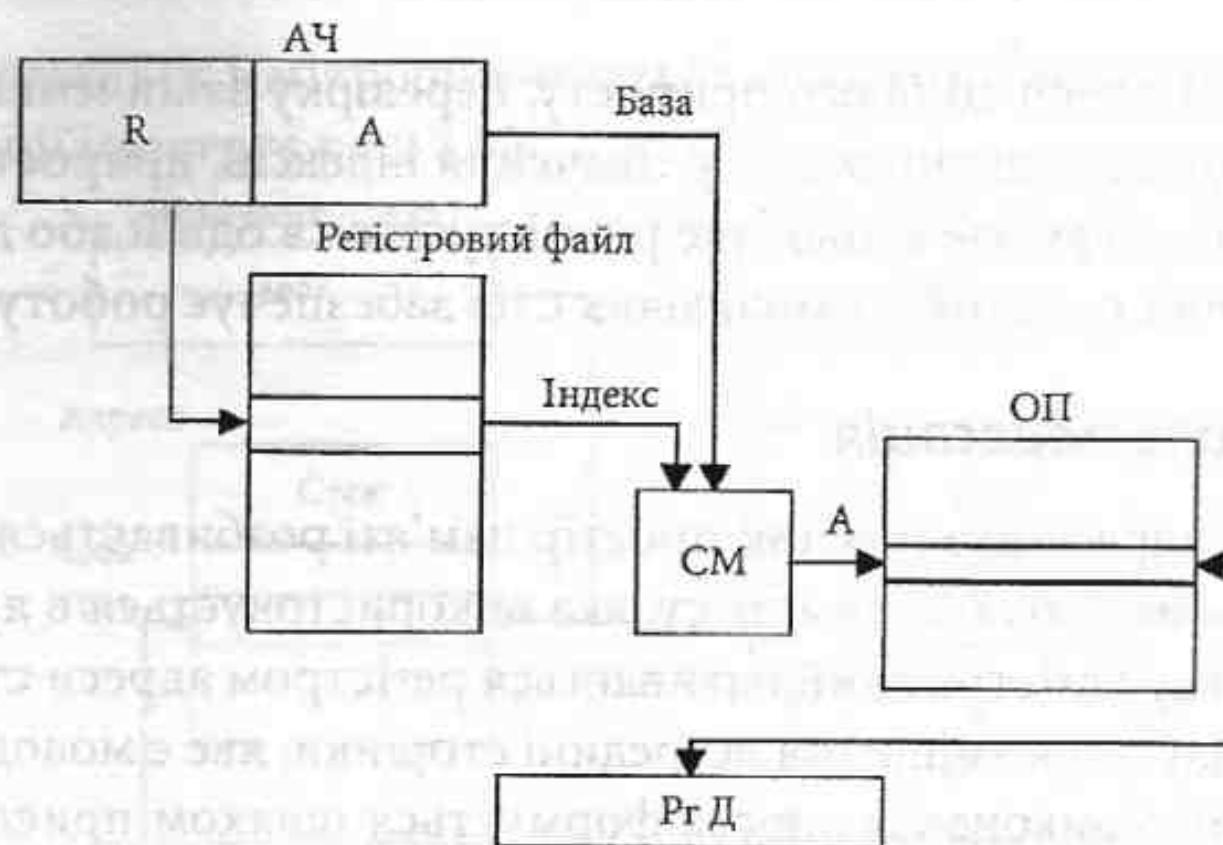


Рис. 3.30. Формування адреси при використанні індексної адресації

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

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



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

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

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

3.4.5. Сторінкова адресація

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

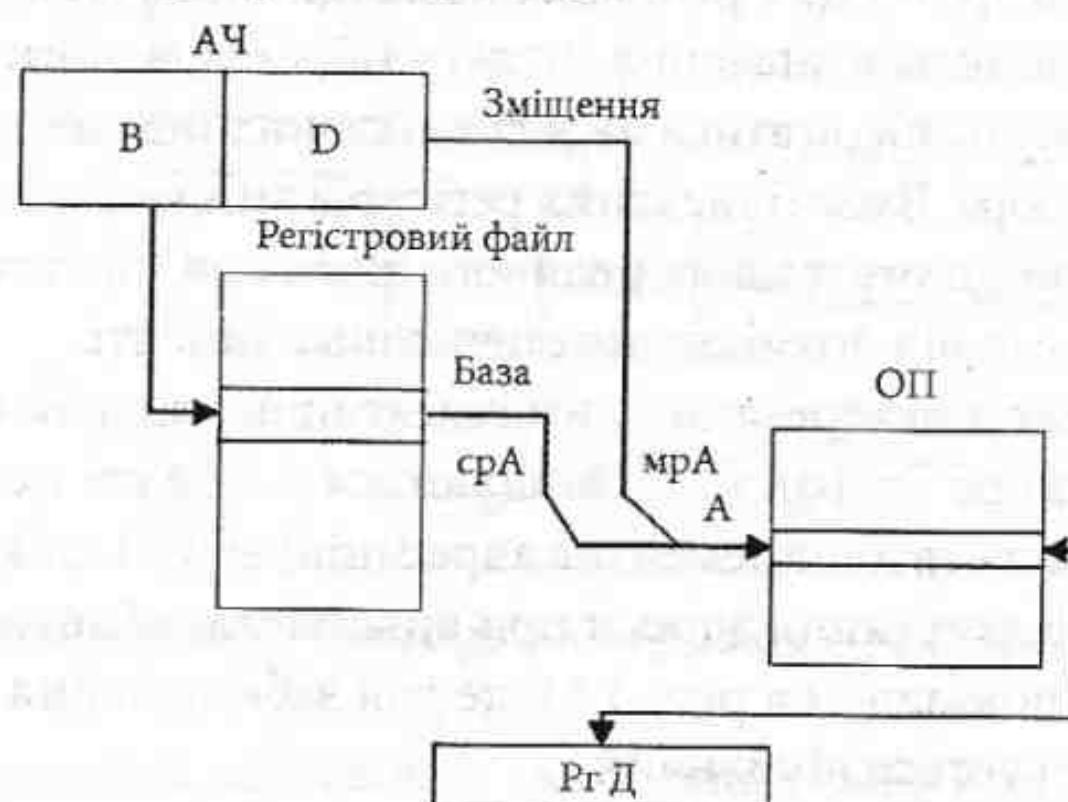


Рис. 3.32. Формування адреси при використанні сторінкової адресації

3.4.6. Неявна адресація

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

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

3.4.7. Стекова адресація

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

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

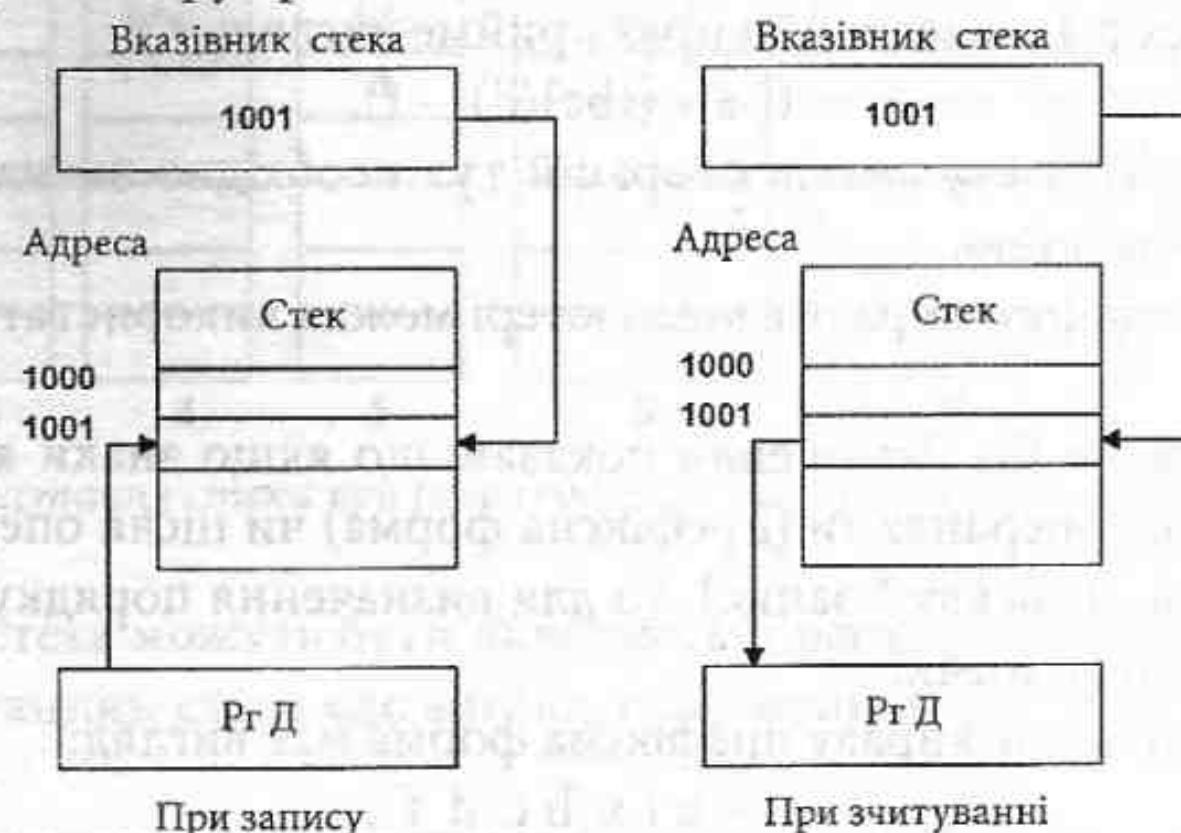


Рис. 3.33. Стекова адресація

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



Рис. 3.34. Виконання двомісної операції з використанням стека

3.4.8. Використання стекової адресації

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

$$a + bc/d - f$$

пріоритети можуть бути наступні: $x, /, +, -$.

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

$$((a + ((bc)/d)) - f).$$

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

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

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

Так, для наведеного вище виразу префіксна форма має вигляд:

$$- + a / x b c d f ,$$

а постфіксна форма має вигляд:

$$a b c x d / + f - .$$

Ця форма – обернений польський запис.

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

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

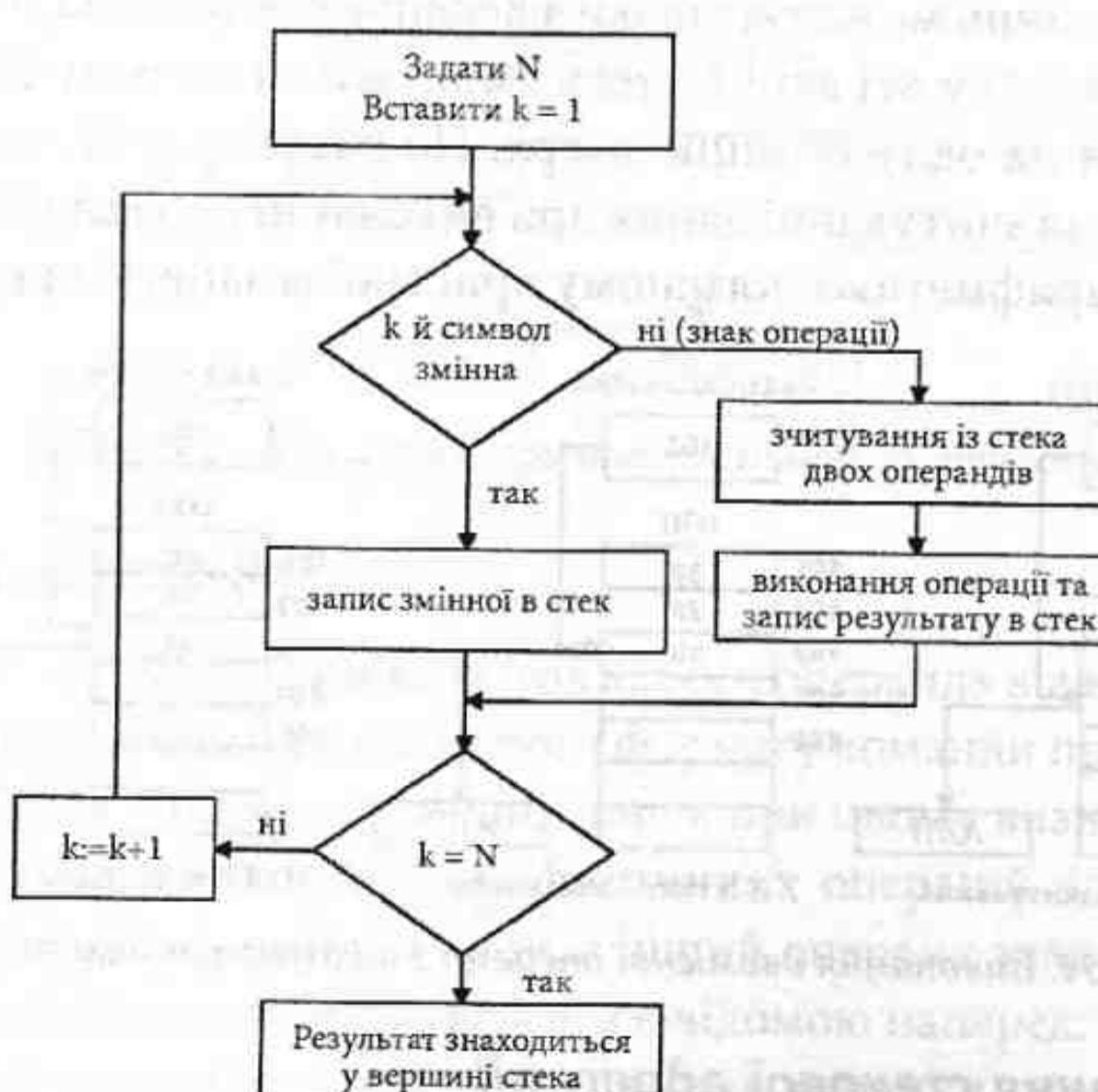


Рис. 3.35. Блок-схема обчислення на стеку виразу, представленого в постфіксній формі

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

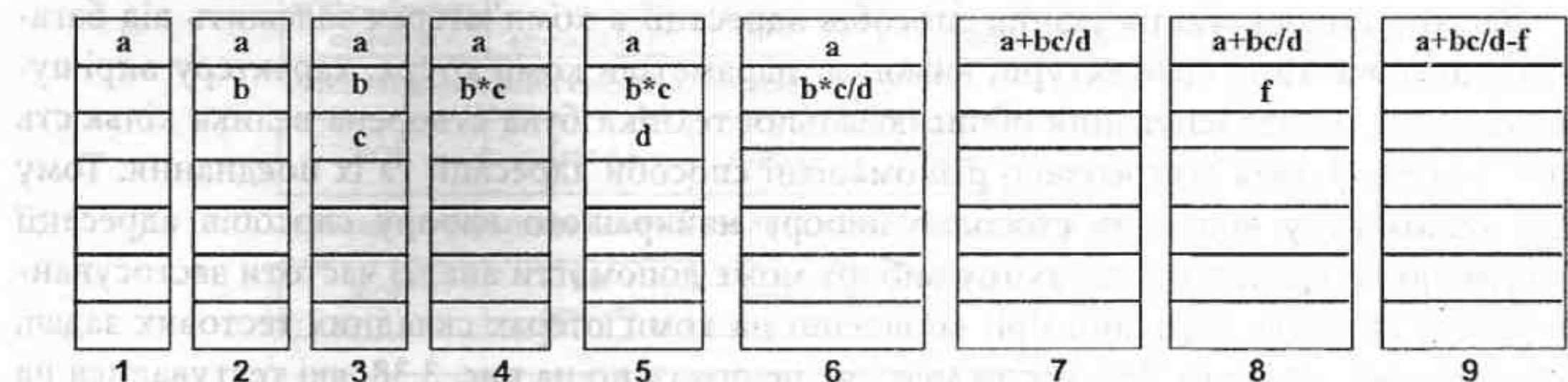


Рис. 3.36. Використання стека при розрахунку по формулі в оберненому польському запису

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

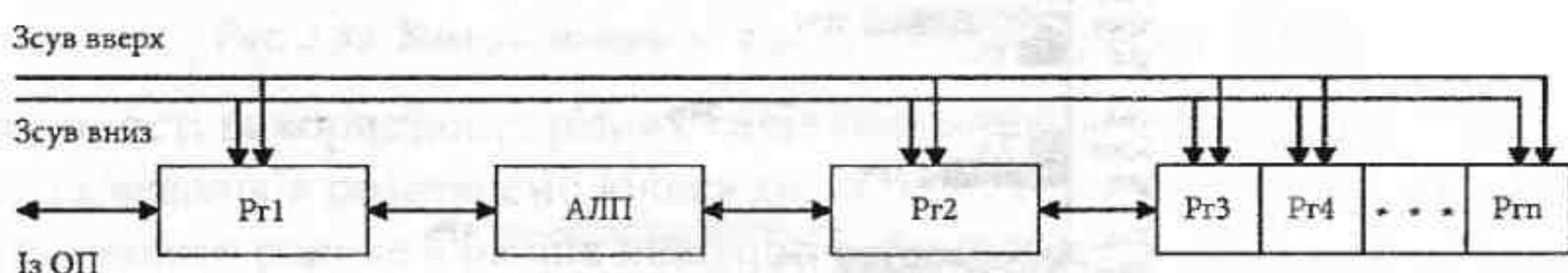


Рис. 3.37. Схема обчислень з використанням регістрового стека

Алгоритм функціонування такий же, як вище описано функціонування на базі основної пам'яті ОП. Операція виконується над вмістом Pr1 і результат розміщується в Pr1, а вміст нижніх регістрів зміщується на один крок вверх.

3.4.9. Вибір способів адресації операндів

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

Таблиця 3.8

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

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

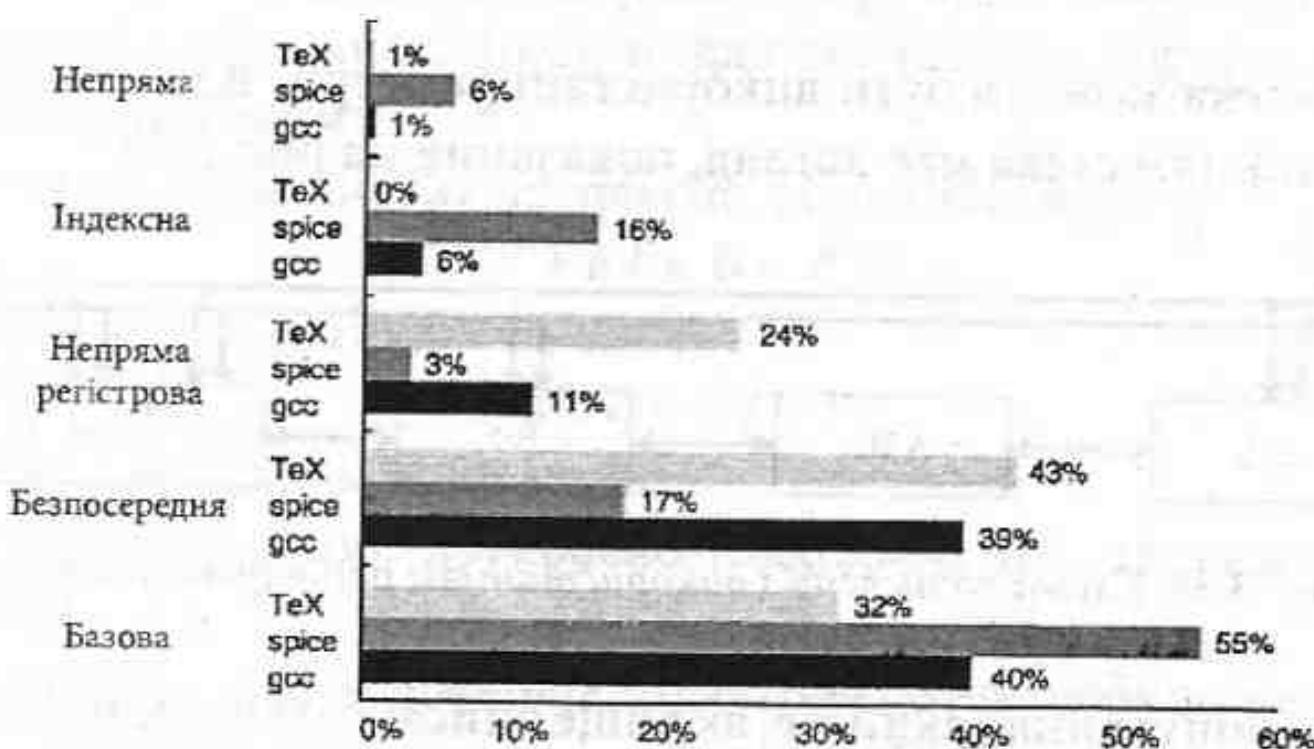


Рис. 3.38. Частота використання способів адресації при виконанні програм TeX, spice та gcc на комп'ютері VAX фірми DEC

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

3.5. Приклади форматів команд

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

Формати команд комп’ютерів різних типів детально розглянуті в літературі та в технічній документації на ці комп’ютери. На рис. 3.39 показано три групи узагальнених форматів команд, які використовувалися та використовуються зараз в комп’ютерах: а – змінний, б – фіксований та с – гіbridний формати.

КОП та кількість операндів	Тип адресації 1	Адреса 1	...	Тип адресації k	Адреса k
----------------------------	-----------------	----------	-----	-----------------	----------

а) Змінний формат (комп'ютери VAX, Intel 80x86 та інші)

КОП	Адреса 1	Адреса 2	Адреса 3
-----	----------	----------	----------

б) Фіксований формат (комп'ютери Alpha, ARM, MIPS, SPARC та інші)

КОП	Тип адресації	Адреса
-----	---------------	--------

КОП	Тип адресації 1	Тип адресації 2	Адреса 1
-----	-----------------	-----------------	----------

КОП	Тип адресації	Адреса 1	Адреса 2
-----	---------------	----------	----------

с) Гібридний формат (комп'ютери IBM 360/70, MIPS16, Thumb, TMS320C64x та інші)

Рис. 3.39. Використовувані в комп'ютерах формати команд

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

3.5.1. Формати команд комп'ютерної системи IBM 370

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

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

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

Формат RR	КОП	R1	R2	R-регистр	
	8	4	4		
Формат RRE	КОП			R1	R2
	16			8	4
Формат RX	КОП	R1	X2	B2	D2
	8	4	4	4	12
Формат RS	КОП	R1	R3	B2	D2
	8	4	4	4	12
Формат SI	КОП	Const	B1	D1	
	8	8	4	12	
Формат S	КОП			D2	
	16			4	12
Формат SS(OL)	КОП	L	B1	D1	B2
	8	8	4	12	4
Формат SS(TL)	КОП	L1	I2	B1	D1
	8	4	4	4	12
Формат SS(RS)	КОП	R1	R3	B1	D1
	8	4	4	4	12
Формат SSE	КОП			D1	B2
	16			4	12

Х – індекс
В – регістр бази
D – зміщення
S – пам'ять
I – поле безпосередньо з даним

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

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

Команда регистр-індексована пам'ять RX. За цією командою перший операнд знаходиться в реєстрі, а другий операнд обчислюється шляхом додавання 12-розрядного зміщення D2 до вмісту реєстра бази B2 та індексного реєстра X2. Обидва реєстри належать до реєстрів загального призначення.

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

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

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

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

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

3.5.2. Формати команд комп'ютера Cyber-70

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

1)	<table border="1"> <tr> <td>КОП</td><td>R1</td><td>R2</td><td>R3</td></tr> <tr> <td>6</td><td>3</td><td>3</td><td>3</td></tr> </table>	КОП	R1	R2	R3	6	3	3	3	2)	<table border="1"> <tr> <td>КОП</td><td>R1</td><td>N</td></tr> <tr> <td>6</td><td>3</td><td>6</td></tr> </table>	КОП	R1	N	6	3	6
КОП	R1	R2	R3														
6	3	3	3														
КОП	R1	N															
6	3	6															
3)	<table border="1"> <tr> <td>КОП</td><td>R1</td><td>R2</td><td>A</td></tr> <tr> <td>6</td><td>3</td><td>3</td><td>18</td></tr> </table>	КОП	R1	R2	A	6	3	3	18								
КОП	R1	R2	A														
6	3	3	18														
4)	<table border="1"> <tr> <td>КОП</td><td>R1</td><td>A</td><td></td></tr> <tr> <td>9</td><td>3</td><td>18</td><td></td></tr> </table>	КОП	R1	A		9	3	18									
КОП	R1	A															
9	3	18															
5)	<table border="1"> <tr> <td>КОП</td><td>L1</td><td>A1</td><td>L2</td><td>C1</td><td>C2</td><td>A2</td></tr> <tr> <td>9</td><td>3</td><td>18</td><td>4</td><td>4</td><td>4</td><td>18</td></tr> </table>	КОП	L1	A1	L2	C1	C2	A2	9	3	18	4	4	4	18		
КОП	L1	A1	L2	C1	C2	A2											
9	3	18	4	4	4	18											

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

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

3.5.3. Формати команд сучасного комп'ютера

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

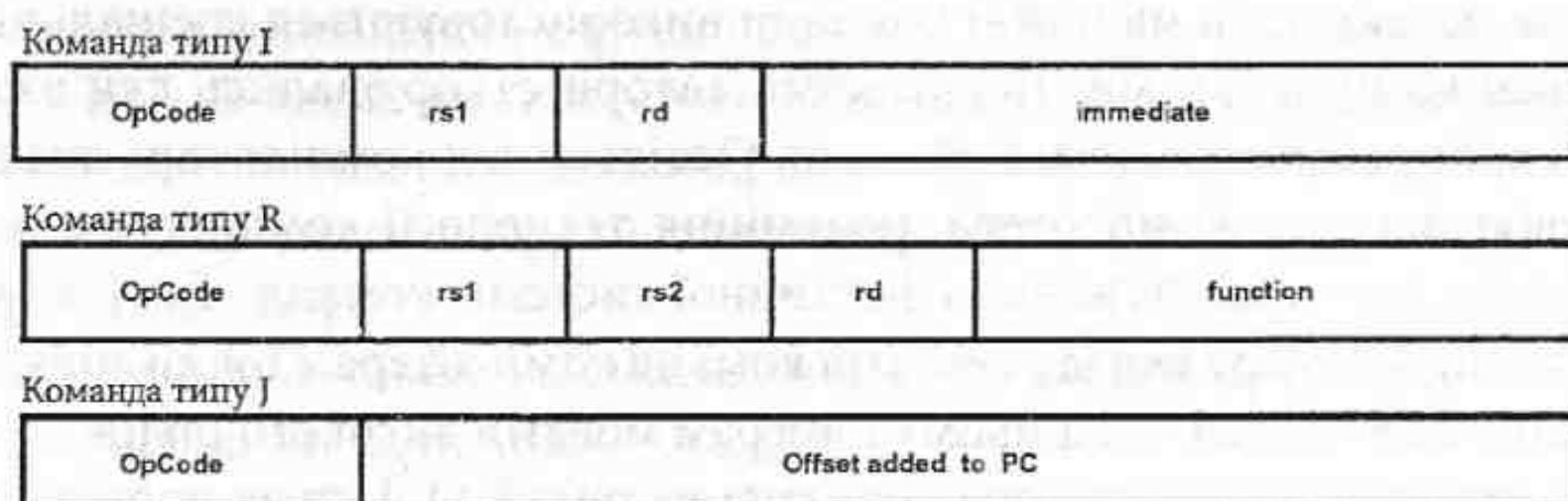


Рис. 3.42. Формати команд комп'ютера DLX

Дамо коротке пояснення щодо наведених на рисунку форматів команд.

- Команда типу I опрацьовує безпосередній операнд (Immediate).

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

- Команда типу J є командою безумовного переходу (Jump).

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

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

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

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

- Function – це поле, що визначає функцію, яка розширює на $2^{11} - 1 = 2047$ комбінацій обмежену кількість дозволених кодів операцій.

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

Особливості форматів команд комп'ютера DLX:

- Довжина усіх форматів – 32 біти.
- Реалізовано тип архітектури регістр-регістр.

- Реалізовано фіксовану систему поділу форматів на поля.
- Усі команди з погляду їхньої обробки поділено на три групи: операції АЛП, операції зчитування/запису, операції керування виконанням програми.

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

3.6. Вплив технології компілювання на систему команд комп'ютера

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

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

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

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

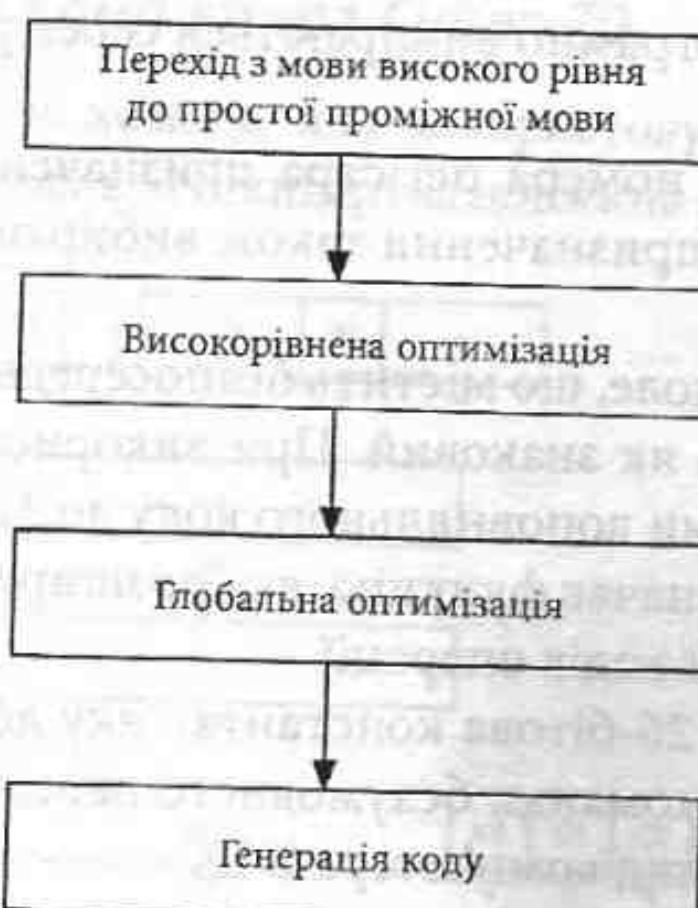


Рис. 3.43. Структура компілятора

3.7. Архітектура системи команд комп'ютера

3.7.1. Класифікація архітектури комп'ютера за складом системи команд

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

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

- комп'ютери із складною (комплексною) системою команд (КССК, Complex Instruction Set Computers – CISC);
- комп'ютери з простою (спрощеною) системою команд (КПСК, Reduced Instruction Set Computers – RISC);
- комп'ютери з доповненою системою команд (КДСК, Supplemented Instruction Set Computers – SISC). В таких комп'ютерах складна або проста система команд доповнюється командами, орієнтованими на конкретну область використання. До таких комп'ютерів, зокрема, відносяться програмовані процесори обробки сигналів (Programmable Digital Signal Processors);
- комп'ютери з орієнтованою (спеціалізованою) системою команд (КОСК, Application Specific Instruction Set Computers – ASISC).

Далі ці архітектури будуть розглянуті детально.

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

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

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

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

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

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

Серед інших особливостей архітектури КПСК слід зазначити наявність досить великого реєстрового файла (у типовій архітектурі КПСК реалізуються 32 або більша кількість реєстрів у порівнянні з 8–16 реєстрами в архітектурі КССК), що дозволяє більшому об'єму даних зберігатися в реєстрах на кристалі процесора більший час і спрощує роботу компілятора при розподілі реєстрів під змінні. Для обробки, як правило, використовуються триадресні команди, що, крім спрощення дешифрування, дає можливість зберігати більшу кількість змінних в реєстрах без їх подальшого перезавантаження.

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

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

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

3.7.3. Особливості архітектури комп'ютера з простою системою команд

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

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

- Система команд має містити мінімальну кількість спрощених команд, що статистично переважають у програмах
- Команди обробки даних реалізуються лише у формі «регистр регистр». Обміни з пам'яттю даних (гарвардська архітектура), з метою модифікації змінних у пам'яті виконуються лише за допомогою команд читання/запису (архітектура load/store).
- Програми, що модифікують власні коди (раніше це було розповсюджено у різних комп'ютерах, зокрема PDP-11/VAX-11) є забороненими. Виходить, що згідно з новою концепцією команди обробки не можуть адресувати нічого, за винятком регістрів процесора
- Дешифрування команд із спрощеними форматами має виконуватися лише апаратно, аби збільшити швидкодію
- У системі команд відносно небагато операцій та режимів адресування операндів (способів адресації).
 - Високий рівень конвеєризації виконання команд
 - Велика кількість регістрів
 - Застосовується багато рівнів ієархії пам'яті
 - Склад системи команд має задовольняти вимоги «зручної» компіляції операторів мов високого рівня

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

- Аналіз області використання з метою визначення найпоширеніших операцій та формування на основі цього списку команд (наприклад, обробки сигналів або створення документа).
- Оптимізація структури процесора, що проектується, з метою забезпечення найшвидшого виконання обраних команд
- Додавання до отриманого списку інших команд, якщо вони не ускладнюють процесора
- Застосування цих же принципів у розробленні інших пристрій комп'ютерної системи.
- Перенесення більшої частини дій з апаратури на програмну частину (компілятор).

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

3.7.4. Архітектура комп'ютера з доповненою системою команд

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

- розділенню шин даних і команд із забезпеченням інформаційного обміну між ними, тобто використанню гарвардської архітектури;
- широкому використанню конвеєрного принципу обробки даних;

- використанню, крім традиційного АЛП, спеціалізованого операційного пристрою, який структурно орієнтований на виконання найуживаніших операцій
- використанню широкорозрядних блоків оперативної та постійної пам'яті великої ємності з можливістю її секціювання із забезпеченням незалежного доступу до секцій;
- використанню взамін адресного реєстра спеціального процесора формування адрес та його структурній орієнтації на роботу з масивами даних та на виконувані алгоритми адресації пам'яті
- введенню в пристрой вибірки команд стеків індексних реєстрів, реєстрів загального призначення, реєстрів попередньої вибірки та інших апаратних засобів для маніпуляцій з командами і даними;
- введенню в систему команд спеціальних команд для виконання найуживаніших алгоритмів;
- використанню швидкодіючих послідовних і паралельних інтерфейсів, які забезпечують створення багатопроцесорних систем
- короткому командному циклу

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

Типова структура комп'ютера з доповненою системою команд наведена на рис. 3.44. Видно, що тут використовується гарвардська архітектура, яка передбачає розподіл пам'яті на пам'ять даних і пам'ять програм, що дає змогу суміщати в часі вибірку і виконання команд. Такий розподіл шин дає змогу створити конвеєр виконання команд і підвищити продуктивність КДСК. В багатьох комп'ютерах з доповненою системою команд, зокрема сім'ї TMS320, використовується модифікована гарвардська архітектура, яка дозволяє обмін інформацією між пам'яттю даних і пам'яттю команд створенням зв'язку між шиною даних і шиною команд (міст зв'язку шин на рис. 3.44).



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

3.7.5. Комп'ютери зі спеціалізованою системою команд

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

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

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

Зазвичай КОСК мають малий набір команд, який включає:

- набір стандартних арифметичних команд, команди роботи з пам'яттю та команди керування програмним потоком, які необхідні для конкретного використання;
- декілька спеціальних команд, наприклад, виконання операцій множення–накопичення даних, цифрової фільтрації, декодування за алгоритмом Вітербі і т. д.

Таким чином, критичні за швидкодією команди можуть бути виконані за мінімальну кількість машинних циклів (можливо за 1 простий цикл) без зберігання у пам'яті проміжних результатів.

Охарактеризувати КОСК можна за допомогою шести основних параметрів. Цими параметрами є: формат даних, формат команд, структура пам'яті, структура регістрової пам'яті, кодування операцій, а також структурні особливості процесора. КОСК зазвичай орієнтовані на обробку даних з фіксованою комою. Розрядна сітка функціональних блоків, шин та пам'яті вибирається, виходячи із потреб задачі. Вони базуються на архітектурі типу регістр-регістр. Регістри регістрового файла процесора є програмно доступними. В КОСК можуть використовуватись два базових типи кодування операцій: мікрокодування та макрокодування.

При мікрокодуванні всі команди виконуються за один машинний цикл. Кожна команда вибирається з пам'яті програм, визначаються всі тракти даних і звертання до пам'яті, які будуть виконані протягом даного машинного циклу. При макрокодуванні деякі команди можуть виконуватися протягом багатьох циклів. Кожна команда визначає всі тракти даних і звертання до пам'яті, які відповідають специфічній трансформації даних, навіть якщо ці дії мають місце в різних машинних циклах. У випадку мікрокодування легше організується конвеєр команд, оскільки машинний код доступний програмісту, а при макрокодуванні ці дії виконує блок керування, тому можлива поява ефектів впливу затримки конвеєра на продуктивність. Цей ефект може бути усунений шляхом введення додаткової апаратури яка його передбачає, але відповідно вимагає ускладнення розробки.

Можливі два варіанти формату команд – ортогональний та кодований. Ортогональний формат складається з фіксованих незалежних керуючих полів. У випадку кодованого формату інтерпретація бітів команди як керуючих полів залежить від значення бітів формату. У випадку ортогонального формату команди розряди всередині керуючого поля можуть бути також закодовані для зменшення його розрядності.

Зазвичай КОСК мають 16- або 32-розрядний кодований формат команди. Прикладна програма розміщується в пам'яті кристала. Формат команди вибирають узгодженим з розміром паралельного порту даних та стандартних блоків пам'яті, які використовуються для зберігання прикладної програми.

КОСК мають ефективну структуру основної та регістрової пам'яті, які забезпечують високу швидкість обміну між різними блоками тракту даних і між трактом даних та пам'яттю. Такі комп'ютери мають один або два блоки пам'яті даних. Для збільшення продуктивності роботи таких комп'ютерів при їх побудові використовується гарвардська архітектура (розділення пам'яті даних та пам'яті програм).

Більшість КОСК використовують гетерогенну структуру регістрової пам'яті, що дозволяє зменшити кількість розрядів команди, збільшуєчи швидкість обміну даними. Це означає, що спеціально організовані реєстри та регістрові файли прямо під'єднані до спеціальних портів функціональних блоків. Функціональні блоки можуть вибирати операнди або записувати результати тільки в конкретні реєстри. Між елементами комп'ютера є тільки ті зв'язки, які є корисними для ефективної реалізації заданого алгоритму.

Архітектура КОСК зазвичай має ряд особливостей, які відрізняють її від інших архітектур і не були відображені в попередніх розділах. Далі наведено перелік особливостей існуючих КОСК:

- Як зазначалось вище, система команд КОСК включає спеціалізовані команди, які дозволяють виконувати критичні ділянки заданих алгоритмів за мінімальну кількість машинних циклів та без засилання в пам'ять проміжних результатів. Структура трактів даних таких процесорів спеціально проектується для конкретних алгоритмів.
- Пам'ять даних зазвичай працює з одним або двома процесорами формування адрес, які підтримують безпосередній, прямий та непрямий способи адресації. Також підтримуються спеціальні адресні операції, такі як обчислення модуля при реалізації циркулярних буферів при виконанні фільтрації та підрахунок з інверсним розповсюдженням переносу при виконанні швидкого перетворення Фур'є.
- КОСК обробки мовних, відео- та аудіосигналів часто включають блоки бітових маніпуляцій. Більше того, ці комп'ютери підтримують кілька типів даних з фіксованою комою (наприклад: 2 різних типи 16-розрядних даних двох доповняльних кодів з різною інтерпретацією бінарних значень, 32-розрядний тип двох доповняльних кодів у акумуляторі та тип 8-розрядного цілого в процесорах формування адрес). Перетворення типів даних підтримуються апаратурою тракту даних.
- Більшість КОСК підтримують стандартні команди керування, такі як умовні розгалуження, що базуються на значеннях розрядів в реєстрі кодів умов. При цьому в них враховано можливості появи затримок при виконанні операцій переходу, які пов'язані з наявністю конвеєра, реалізовано можливість виконання арифметичних команд та команд пересилання при наявності певної умови, що дозволяє реалізувати умовні ал-

горитми без необхідності умовного зчитування програмного лічильника, а також можливість виконання деяких арифметичних команд резидентно, коли поведінка команди залежить від значення розряду в керуючому реєстрі, який може бути записаний іншими операціями.

3.8. Короткий зміст розділу

В даному розділі були розглянуті основні елементи архітектури комп’ютера, які включають організацію пам’яті, формати і типи команд, способи адресації. Показано, як кодуються та виконуються команди в комп’ютері. Проведена класифікація команд відповідно до ініційованих ними типів операцій та детально розглянуті команди обробки даних, переміщення даних, передачі керування, введення виведення. Багато рішень має бути прийнято, коли проектується система команд. Наявність значної кількості команд в системі команд комп’ютера призводить до збільшення довжини команди, до збільшення часу вибірки та декодування. Команди фіксованої довжини легше декодувати, але втрачається гнучкість. Реалізація одних команд на основі інших є компромісом між потребою у великих системах команд і бажанні мати короткі команди.

Конвеєризація виконання команд – один із прикладів паралелізму на рівні команд. Це загальна але складна технологія, яка може підвищити продуктивність виконання послідовності команд. Проте рівень паралелізму може бути обмежений конфліктами в конвеєрі.

Є три типи архітектур комп’ютера за типом адресованої пам’яті: стекова, акумуляторна, та на основі реєстрів загального призначення. Кожна має свої переваги і недоліки, які потрібно розглядати в контексті застосування запропонованої архітектури.

Досягнення в технології пам’яті, привівши до великих її ємностей, викликали потребу в альтернативних способах адресації. Були введені різні способи адресації, включаючи безпосередню, пряму, непряму, базову, індексну, сторінкову і стекову. Наявність цих різних способів забезпечує гнучкість і зручність для програміста без заміни фундаментальних операцій процесора.

Розгляд архітектури комп’ютера на рівні системи команд встановлює межу між апаратурою та програмним забезпеченням і дозволяє побачити комп’ютер на рівні, який видимий програмісту, що працює на мові асемблера, або розробнику компіляторів. За складом системи команд комп’ютери можуть бути поділені на наступні типи: комп’ютери із складною, з простою, з доповненою та спеціалізованою системою команд. В розділі щі архітектури розглянуті детально.

3.9. Література для подальшого читання

Класифікація різних типів пам’яті залежно від способу доступу до даних наведена в [5]. В роботі [6] було запропоновано новий тип пам’яті, яка за способом доступу до даних належить до пам’яті з впорядкованим доступом. Принципи побудови та роботи пам’яті з довільним доступом детально розглянуті в літературі [8, 9]. Асоціативна пам’ять описана в [1].

Порядок виконання команд та формат команди описані в роботах [2, 3]. Системи команд, адресація та формати команди розкриті детально в майже кожній книзі, яка стосується архітектури комп’ютерної системи. Книги [21, 22] найповніше охоплюють ці питання.

ня. Багато книг, як наприклад [10, 11, 15, 18], присвячені архітектурі процесора x86 фірми Intel. Для тих, хто зацікавлений у вивченні серії 68000, пропонується література [20, 31].

Належне обговорення конвеєрної обробки команд надає [24]. В [16] подано цікавий короткий огляд конфліктів у конвеєрі. Історія конвеєрної обробки розкрита в [23]. Щоб одержати знання обмежень і проблем з конвеєрною обробкою, варто прочитати [30].

Подивіться [11] для довершеного і зручного введення в сім'ю процесорів Intel. В [14] зроблено опис безадресної машини фірми Burroughs. [28] дає достатнє уявлення про серію IBM 360. В [12] наведено деталі про систему VAX, в якій об'єднана двоадресна архітектура з відпрацьованою системою команд. В [25] надано короткий огляд архітектури SPARC. В [19] зроблено цікавий огляд віртуальної машини Java.

Порівняння архітектур комп'ютерів із складною та спрощеною системою команд зроблено в [21, 22]. Тут же запропоновано архітектуру комп'ютера із спрощеною системою команд DLX. Архітектура комп'ютерів з доповненою системою команд розглянута в [4].

3.10. Література до розділу 3

1. Искусственный интеллект: В 3-х книгах. Кн. 3. Программные и аппаратные средства: Справочник / Под ред. В. Н. Захарова, В. Ф. Хорошевского. – М.: Радио и связь, 1990. – 191 с
2. Каган Б М. Электронные вычислительные машины и системы. – М.: Энергия, 1979. – 528 с
3. Каган Б М., Каневский М М. Цифровые вычислительные машины и системы. – М.: Энергия, 1974. – 680 с
4. Мельник А О. Программировані процесори обробки сигналів. – Львів: Вид-тво Національного університету “Львівська політехніка”, 2000. – 55 с
5. Угрюмов Е П. Цифровая схемотехника. – СПб.: БХВ – Санкт Петерберг, 2000. – 528 с
6. Мельник А О. Принципи побудови буферної сортувальної пам'яті. Вісник Державного університету “Львівська політехніка”. – “Комп'ютерна інженерія та інформаційні технології”, № 307, 1996. – С. 65–71.
7. Справочник по цифровой вычислительной технике. Б Н. Малиновский и др – К.: Техніка, 1980. – 320 с
8. Шигин А Г., Дерюгин А А. Цифровые вычислительные машины (память ЦВМ). – М.: Энергия, 1975. – 536 с
9. Метлицький Б А., Каверзnev В В. Системы параллельной памяти. Теория, проектирование, применение. Под. ред. В И. Тихонова. – Л., 1989.
10. Abel, Peter. *IBM PC Assembly Language and Programming*, 5th ed., Upper Saddle River, NJ: Prentice Hall, 2001.
11. Brey, B. *Intel Microprocessors 8086/8088, 80186/80188, 80286, 80386, 80486 Pentium, and Pentium Pro Processor, Pentium II, Pentium III, and Pentium IV: Architecture, Programming, and Interfacing*, 6th ed., Englewood Cliffs, NJ: Prentice Hall, 2003.
12. Brunner, R.A. *VAX Architecture Reference Manual*, 2nd ed., Herndon, VA: Digital Press, 1991.
13. Burks, Alice, & Burks, Arthur. *The First Electronic Computer: The Atanasoff Story*. Ann Arbor, MI: University of Michigan Press, 1988.
14. Hauck, E. A., & Dent, B. A. “Burroughs B6500/B7500 Stack Mechanism”, *Proceedings of AFIPS SJCC* (1968), Vol. 32, pp. 245–251.
15. Jones, William. *Assembly Language Programming for the IBM PC Family*, 3rd ed., El Granada, CA: Scott/Jones Publishing, 2001.
16. Kaeli, D., & Emma, P. “Branch History Table Prediction of Moving Target Branches Due to Subroutine Returns”. *Proceedings of the 18th Annual International Symposium on Computer Architecture*, May 1991.

17. Lindholm, Tim, & Yellin, Frank. *The Java Virtual Machine Specification*. Online at java.sun.com/docs/books/vmspec/html/VMSpecTOC.cod.html.
18. Messmer, H. *The Indispensable PC Hardware Book*. Reading, MA: Addison-Wesley, 1993.
19. Meyer, J., & Downing, T. *Java Virtual Machine*. Sebastopol, CA: O'Reilly & Associates, 1991.
20. Miller, M. A. *The 6800 Family, Architecture Programming and Applications*, 2nd ed., Columbus, OH: Charles E. Merrill, 1992.
21. D. Patterson, J. Hennessy. Computer Architecture. A Quantitative Approach. Morgan Kaufmann Publishers, Inc. 1996.
22. Patterson, D. A., & Hennessy, J. L. *Computer Organization and Design, The Hardware/Software Interface*, 2nd ed , San Mateo, CA: Morgan Kaufmann, 1997.
23. Rau, B. Ramakrishna, & Fisher, Joseph A. "Instruction-Level Parallel Processing: History, Overview and Perspective". *Journal of Supercomputing* 7 (1), Jan. 1993, pp. 9-50.
24. Sohi, G. "Instruction Issue Logic for High-Performance Interruptible, Multiple Functional Unit, Pipelined Computers". *IEEE Transactions on Computers*, March 1990.
25. SPARC International, Inc., *The SPARC Architecture Manual: Version 9*, Upper Saddle River, NJ: Prentice Hall, 1994.
26. Stallings, W. *Computer Organization and Architecture*, 5th ed., New York, NY: Macmillan Publishing Company, 2000.
27. Stern, Nancy. *From ENIAC to UNIVAC: An Appraisal of the Eckert-Mauchly Computers*. Herndon, VA: Digital Press, 1981.
28. Struble, G. W. *Assembler Language Programming: The IBM System/360 and 370*, 2nd ed., Reading, MA: Addison-Wesley, 1975.
29. Tanenbaum, Andrew. *Structured Computer Organization*, 4th ed., Upper Saddle River, NJ: Prentice Hall, 1999.
30. Wall, David W. *Limits of Instruction-Level Parallelism*. DEC-WRL Research Report 93/6, Nov. 1993.
31. Wray, W. C., & Greenfield, J. D. *Using Microprocessors and Microcomputers, the Motorola Family*. Englewood Cliffs, NJ: Prentice Hall, 1994.

3.11. Питання до розділу 3

1. Як організовується зв'язок між процесором і основною пам'яттю?
2. Який порядок виконання команд в комп'ютері?
3. Як кодуються команди в комп'ютері?
4. Що таке асемблерна мова і для чого використовується асемблер?
5. Як класифікуються команди за типами операцій?
6. Назвіть команди обробки даних
7. Назвіть базові операції зсуву
8. Назвіть команди переміщення даних
9. Поясніть принципи організації послідовного виконання команд і розгалуження
10. Назвіть команди передачі керування
11. Назвіть команди переходу
12. Назвіть команди пропуску
13. Назвіть команди звертання до підпрограм
14. Поясніть принципи конвеєрного виконання команд
15. Якою є продуктивність 4-ярусного конвеєра з тактом 20 нс при виконанні 100 команд?
16. Назвіть можливі конфлікти, які можуть сповільнити конвеєр
17. Які використовуються формати команд при роботі з основною пам'яттю?
18. Які формати команд використовуються при роботі з реєстрами процесора?

19. Які головні критерії вибору формату команд?
20. Поясніть різницю між акумуляторною архітектурою, стековою архітектурою та архітектурою на основі регистрів загального призначення
21. Поясніть різницю між архітектурами системи команд типу register register, register memory і memory register
22. Які переваги та недоліки команд з фіксованим та зі змінним форматом? Який формат є вживаним в сучасних комп'ютерах і чому?
23. Яким чином знаходяться дані в пам'яті коли в команді відсутня адресна частина?
24. Яка програма має більше команд: та, що складається з безадресних команд, одноадресних команд, чи з двоадресних команд? Чому?
25. Що таке спосіб адресації?
26. Які є способи адресації пам'яті? Їх призначення?
27. Як організовується стекова пам'ять?
28. Поясніть порядок організації обчислень при використанні стекової адресації
29. Наведіть приклади використання інфіксної, префіксної та постфіксної форм запису арифметичних виразів
30. Наведіть приклади безпосередньої, прямої, непрямої, відносної та базової адресацій
31. Чим відрізняється індексна адресація від базової?
32. Чому необхідна велика кількість різних способів адресації?
33. Які формати команд використовуються в системі IBM 370? Їх відмінності
34. Які формати команд використовуються в машині Cyber-70?
35. Які формати команд використовуються в комп'ютері DLX?
36. Дайте класифікацію архітектур комп'ютера за складом системи команд
37. Яка різниця між комп'ютерами із складною та простою системами команд?
38. Які особливості має комп'ютер з доповненою системою команд?
39. Які переваги має комп'ютер з орієнтованою системою команд?