

## Розділ 4

### *Процесор універсального комп'ютера*

Процесор – це центральний пристрій комп'ютера. Він виконує задані програмою петретворення інформації та здійснює керування всім обчислювальним процесом і взаємодією складових частин комп'ютерної системи.

Основною функцією процесора як центрального пристрою комп'ютера є виконання послідовності команд, які зберігаються в основній пам'яті. Послідовність операцій, необхідних для виконання однієї команди, називається командним циклом, який може бути поділений на дві основні фази: фазу вибірки та фазу виконання. Спочатку процесор вибирає команду з пам'яті в рамках фази вибірки. Фаза виконання включає декодування команди, вибірку з основної пам'яті вказаних в команді операндів, виконання операції, яка вказана в полі коду операції команди, а також запам'ятування результатів. Робота процесора в рамках командного циклу задається послідовністю мікрооперацій, які формує пристрій керування.

Крім виконання арифметичних та логічних операцій, процесор керує роботою інших вузлів комп'ютера. Зокрема, він здійснює керування введенням-виведенням інформації, наприклад пересиланням даних між пристроями введення-виведення та основною пам'яттю.

Основні елементи процесора – арифметико-логічний пристрій, пристрій керування і регістрова пам'ять або, як її ще називають, надоперативний запам'ятовуючий пристрій. До складу регістрової пам'яті, в свою чергу, входять, як було показано в розділі 3, наступні вузли – програмний лічильник, регістри: адреси, команди, даних, слова стану програми, а також регістровий файл, який складається з програмно доступних регістрів. Подібним чином і пристрій керування та арифметико-логічний пристрій поділяються на складові, які будуть розглянуті в наступних розділах.

Залежно від зв'язків між функціональними вузлами процесора і організації їх взаємодії розрізняють кілька структур процесора. Далі будуть розглянуті структури, які знайшли використання в сучасних комп'ютерах.

#### **4.1. Процесор комп'ютера із складною системою команд**

##### **4.1.1. Одношинна структура процесора**

Почнемо розгляд процесора з аналізу його структури та організації роботи. Однією з найпростіших структур процесора є одношинна структура. Одношинну структуру процесора і його зв'язки з іншими пристроями комп'ютера показано на рис. 4.1. Як бачимо, до складу процесора входять пристрій керування, арифметико-логічний пристрій АЛП з вхідним PgY та вихідним PgZ регістрами, і регістрова пам'ять, до складу якої входять регістр команд PgK, регістр даних PgD, регістр адреси PgA, програмний лічильник ПЛ, та регістрові

вий файл – стек програмно доступних регістрів Рг0, Рг1... Рг(n-1). Обмін інформацією між названими пристроями здійснюється через спільну внутрішню шину процесора. Зв'язок процесора з основною пам'яттю проводиться через регістри адрес РгА та даних РгД.

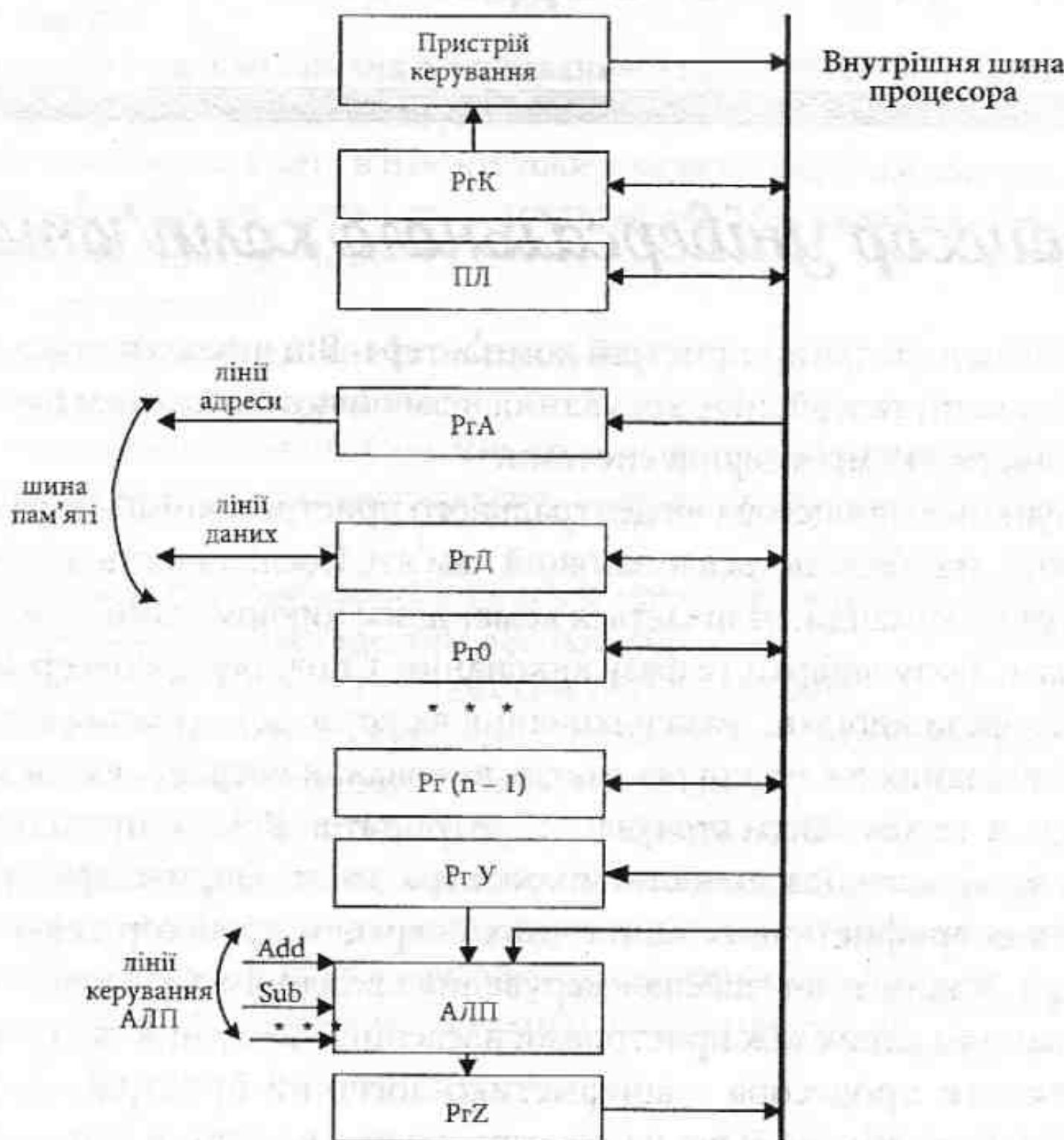


Рис. 4.1. Одношинна структура процесора

РгА зберігає адресу даного або команди при зверненні до основної пам'яті. РгК зберігає команду після її зчитування з основної пам'яті. РгД зберігає операнд при його запису або зчитуванні з основної пам'яті. ПЛ – програмний лічильник, який підраховує команди та зберігає адресу поточної команди.

Регістри загального призначення Рг<sub>0</sub>–Рг<sub>(n-1)</sub> регістрового файла є програмно доступними. Вони можуть використовуватися програмістом в якості адресних регістрів, індексних регістрів при виконанні операцій модифікації адрес або в якості регістрів для зберігання проміжних результатів обчислень. Більшість комп'ютерів мають в складі процесора тригери для зберігання бітів стану процесора, або як їх ще називають, прапорців. Кожен прапорець має спеціальне призначення. Частина прапорців вказує на результати арифметичних і логічних операцій: додатній результат (P), від'ємний результат (N), нульовий результат (Z), перенос (C), арифметичне переповнення (V) тощо.

Різні команди вказують процесору, коли встановити чи очистити ці тригери. Інша частина прапорців вказує режими захисту пам'яті. Існують також прапорці, які вказують пріоритети виконуваних програм. В деяких процесорах додаткові тригери слугують для зберігання кодів умов, формуючи регістр кодів умов. Взяті разом описані прапорці формують слово стану програми (ССП), а відповідні тригери – регістр ССП.

### 4.1.2. Основні операції процесора

Можна виділити дві фази виконання команди в процесорі: фаза вибірки та фаза виконання. Фаза вибірки передбачає вибірку вмісту комірки основної пам'яті, в якій зберігається команда, за значенням програмного лічильника ПЛ і засилання команди в РгК ( $\text{РгК} := [[\text{ПЛ}]]$ ), а також прирошення ПЛ на одиницю:  $[\text{ПЛ}] := [\text{ПЛ}] + 1$ . Фаза виконання команди передбачає її дешифрування та виконання операцій, вказаних в коді операції команди. Для того, щоб побачити, як обидві фази виконуються в процесорі, розглянемо його основні операції, до виконання яких задіяні представлені на рис. 4.1 елементи.

#### 4.1.2.1. Вибірка слова з пам'яті

Нехай адреса комірки основної пам'яті знаходиться в регістрі Рг1, а дані потрібно розмістити в регістрі Рг2.

Для вибірки із основної пам'яті необхідно виконати наступну послідовність операцій:

1. РгА := Рг1 (запис до регістра адреси РгА вмісту регістра Рг1).

2. Зчитування (виконання операції зчитування команди з комірки основної пам'яті до регістра РгД шляхом подання сигналу Read на вхід керування режимом роботи основної пам'яті та сигналу запису до регістра РгД).

3. Чекання на сигнал підтвердження зчитування.

4. Рг2 := РгД (запис до регістра Рг2 даного з регістра РгД).

Пункт 3 виконується при асинхронному принципі обміну між процесором і основною пам'яттю, коли потрібно чекати на сигнал підтвердження зчитування. При синхронному принципі обміну чекати на сигнал підтвердження зчитування не потрібно, оскільки до подання сигналу запису в регістр РгД передбачається гарантована наявність даного на його вході.

#### 4.1.2.2. Запам'ятування слова в пам'яті

Нехай слово, яке запам'ятується в основній пам'яті, знаходиться в регістрі Рг2, а адреса – в регістрі Рг1. Тоді послідовність операцій буде наступною:

1. РгА := Рг1 (запис до регістра адреси РгА вмісту регістра Рг1).

2. РгД := Рг2 (запис до регістра даних РгД вмісту регістра Рг2), запис (виконання операції запису слова з регістра РгД до комірки основної пам'яті шляхом подання сигналу Write на вхід керування режимом роботи основної пам'яті).

3. Чекання на сигнал підтвердження запису (при асинхронному принципі обміну між процесором і основною пам'яттю).

#### 4.1.2.3. Обмін даними між регістрами

Символьне зображення вхідних і вихідних елементів регістрів процесора показане на рис. 4.2 у вигляді ключів, які пропускають або не пропускають інформацію з входу на вихід, залежно від значення сигналів керування на їх входах. Тут сигнал керування входом  $i$ -го регістра Ргі позначено як Ргі in, а сигнал керування виходом  $i$ -го регістра Ргі позначено як Ргі out. В конкретній схемі регістра це можуть бути, наприклад, вхід запису числа до регістра та його триставільний вихід.

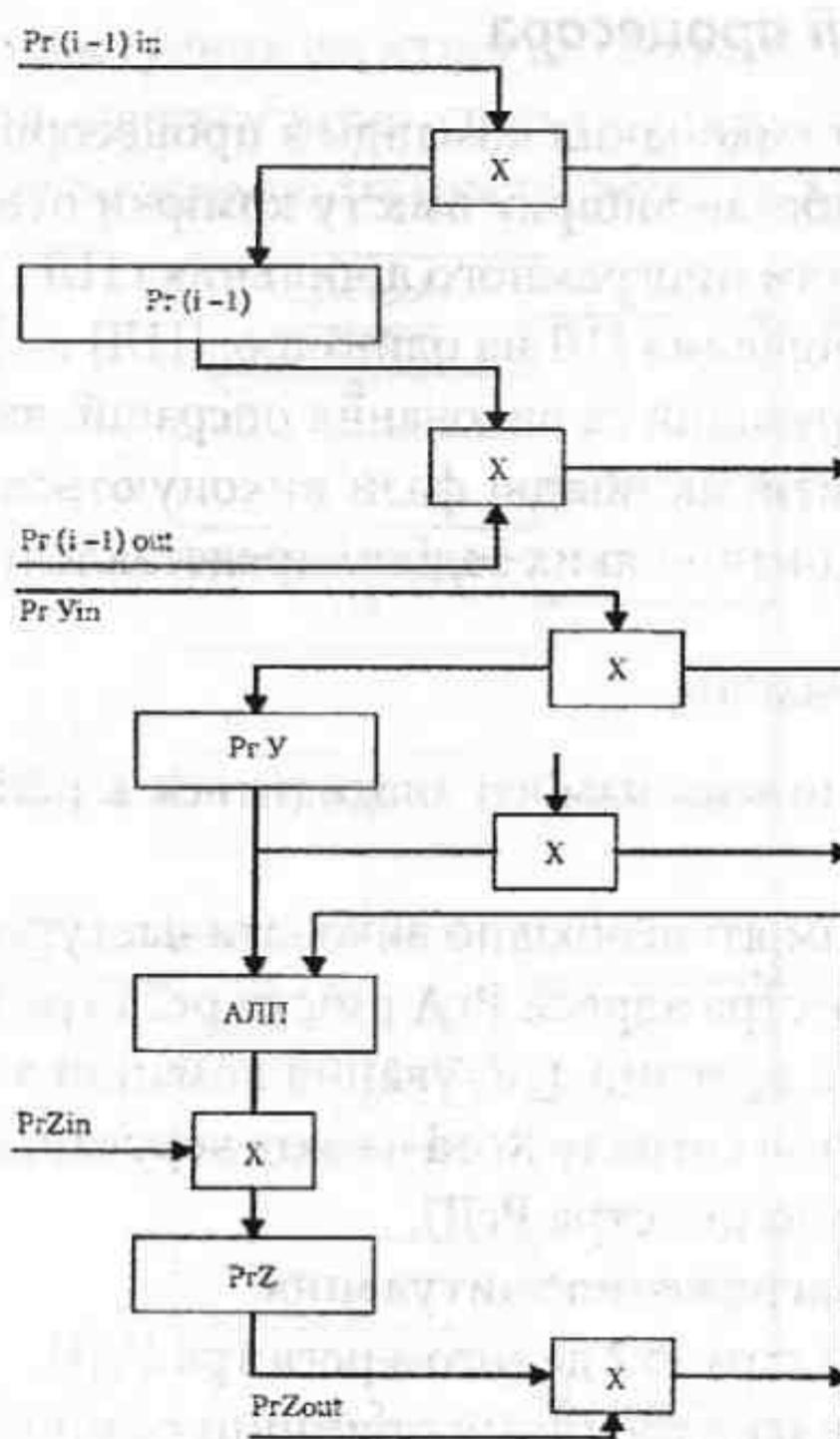


Рис. 4.2. Фрагмент схеми процесора з вхідними та вихідними елементами регістрів

Подання 1 на вхід керування регістра  $Pr_{iout}$  з'єднує вихід регістра  $Pr_i$  з шиною, а подання 1 на вхід керування регістра  $Pr_{iin}$  записує число з шини в регістр  $Pr_i$ . Подаючи на регістри вказані сигнали, можна переписувати числа з одного регістра в інший наприклад, для перезапису числа із регістра  $Pr_3$  до регістра  $Pr_5$  необхідно подати наступні сигнали:  $Pr_{3out}, Pr_{5in}$ .

#### 4.1.2.4. Виконання арифметичних і логічних операцій

Арифметико-логічний пристрій (АЛП) процесора призначений для виконання операцій обробки даних. Тип виконуваної операції вказується кодом на вході керування АЛП. В АЛП, зокрема, виконуються такі операції: зсув – зміщення кодів, які зберігаються в регістрах регістрового файлу, вліво або вправо на задане число розрядів; додавання до слова 1 або -1 – операція рахунку; дешифрування – перетворення двійкових кодів у сигнали (однорядний код); шифрування – перетворення однорядного коду в двійковий; порівняння – визначення відношення старшинства двох чисел або їх рівності; порозрядне доповнення – формування оберненого коду; порозрядні логічні множення і додавання двох чисел; порозрядне додавання двох чисел по модулю; додавання двох чисел. Звичайно, цей перелік може бути розширений.

Розглянемо виконання операції додавання двох чисел з регістрів  $Pr_1$  і  $Pr_2$  з записом результату в регістр  $Pr_3$  на одношинній структурі процесора, представлений на рис. 4.1:

- 1)  $Pr_{1out}, Pr_{Yin}$  (запис до вхідного регістра АЛП  $Pr_Y$  вмісту регістра  $Pr_1$ ).
- 2)  $Pr_{2out}, Add, Pr_{Zin}$  (подання числа з регістра  $Pr_2$  на внутрішню шину процесора, звідки воно поступає на другий вхід АЛП, виконання в АЛП операції додавання чисел з регістра  $Pr_Y$  та з шини і запам'ятовування результату в регістрі  $Pr_Z$ ).

3) PrZout, PrZin (запис до реєстра PrZ вмісту реєстра PrZ).

Подібним чином виконуються інші вище перераховані операції. Необхідно відзначити, що сигнали Prjout та Prjin, де i та j – номери реєстрів, мають бути рознесеними в часі для забезпечення коректного перезапису інформації з одного реєстра до іншого з врахуванням часу спрацювання їх вхідних та вихідних схем, ємності провідників шини та затримки в комбінаційних схемах АЛП. Цей час визначає такт роботи процесора.

#### 4.1.3. Багатошинна структура процесора

Наведена на рис. 4.1 одношинна структура процесора є достатньо простою. Тому вона широко використовується при побудові процесорів реальних комп'ютерів. Зокрема, за такою схемою побудовано більшість мікроконтролерів, від яких не вимагається висока швидкодія. Якщо ж така вимога існує, то застосовується багатошинна організація процесора, в якій завдяки наявності багатьох шин забезпечується можливість паралельного обміну інформації між функціональними вузлами процесора і, тим самим, суттєво підвищується швидкість опрацювання інформації. Як приклад на рис. 4.3 показано можливий варіант двошинної структури процесора.

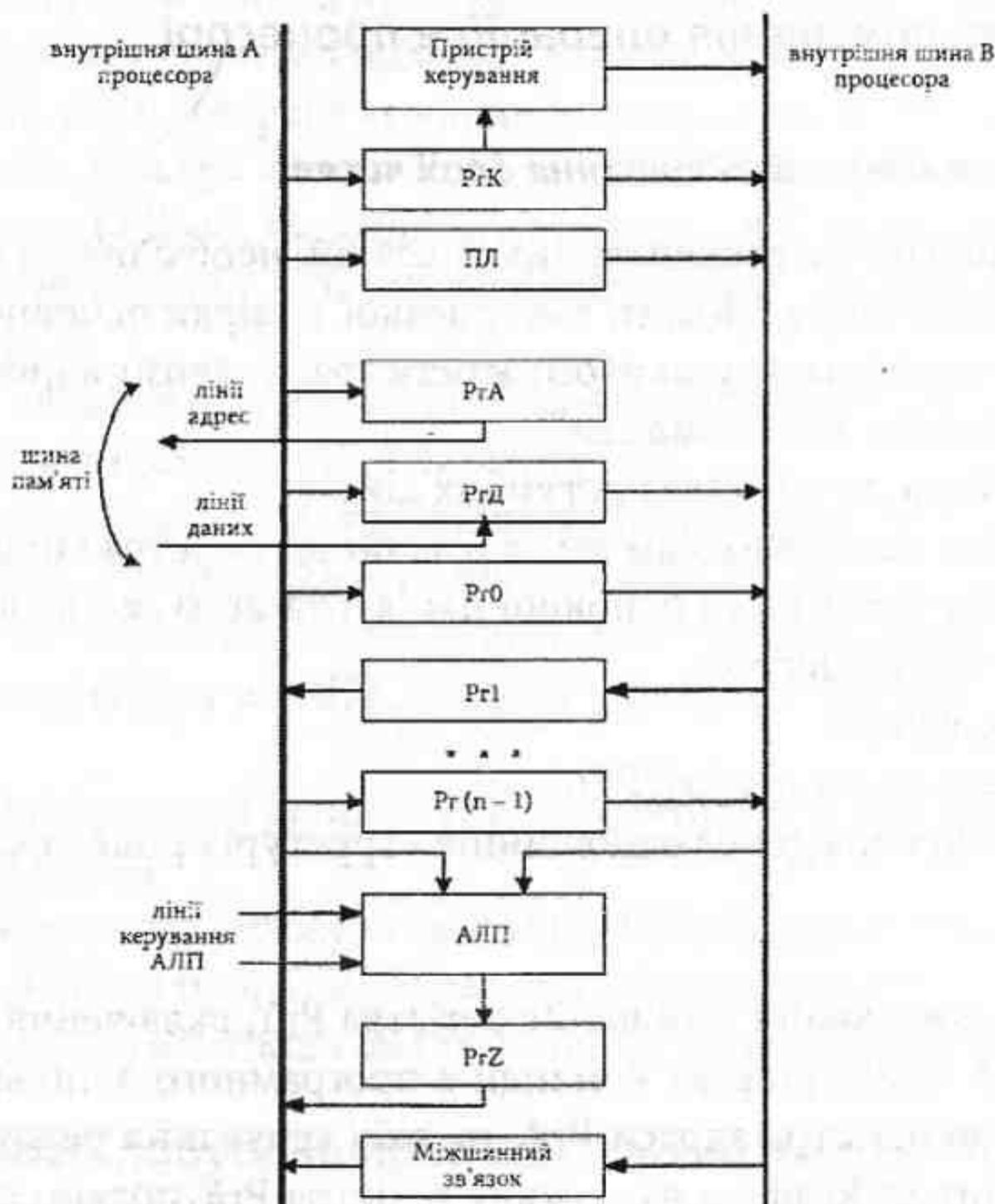


Рис. 4.3. Двошинна структура процесора

Тут входи та виходи реєстрів процесора приєднані до різних шин, що дозволяє одночасно здійснити обмін між двома парами реєстрів, а не між однією, як це було в одношинній структурі процесора. Тим самим в два рази зростає кількість переданої одночасно інформації. Міст зв'язку між шинами, який може бути відчинений або засунений, призначений для обміну інформацією між шинами.

Розглянемо виконання на цій структурі процесора операції  $\text{Pr}3 := \text{Pr}0 + \text{Pr}1$ :

1)  $\text{Pr}0\text{out}, \text{Pr}1\text{out}, \text{Add}, \text{Pr}Z\text{in}$  (подання на шину A числа з регістра  $\text{Pr}1$  та на шину B числа з регістра  $\text{Pr}0$ , виконання в АЛП операції додавання та запис результату до регістра  $\text{Pr}Z$ ).

2)  $\text{Pr}Z\text{out}, \text{Pr}3\text{in}$  (запис числа з регістра  $\text{Pr}Z$  до регістра  $\text{Pr}3$  через міст M зв'язку між шинами).

Як видно, для виконання операції додавання тут потрібно лише два такти, тоді як в одношинній структурі процесора було потрібно три такти.

Зрозуміло, що із збільшенням кількості шин швидкість обміну зростає. Так, в тришинній структурі процесора операція додавання двох чисел буде виконана за один такт.

Збільшення кількості шин ускладнює структуру процесора та збільшує кількість необхідного обладнання на його побудову. Адже кожна додаткова шина – це велика кількість додаткових провідників, які займають багато місця на кристалі. Однак потреба підвищення продуктивності змушує розробників застосовувати саме багатошинну структуру процесора. Більшість процесорів сучасних високопродуктивних комп’ютерів є багатошинними.

#### **4.1.4. Приклади виконання операцій в процесорі**

##### **4.1.4.1. Виконання операції додавання двох чисел**

Об’єднаємо послідовність елементарних операцій, необхідних для виконання однієї команди. Розглянемо команду “Додати вміст деякої комірки основної пам’яті до вмісту регістра  $\text{Pr}1$  із записом результату до цього ж регістра, причому адреса комірки основної пам’яті задана в адресному полі команди”.

Виконання цієї команди вимагає наступних дій:

1. Вибірки команди з основної пам’яті та її запис до регістра команди  $\text{Pr}K$ .
2. Вибірки першого операнда з основної пам’яті та його запис до одного з регістрів надоперативної пам’яті процесора.
3. Виконання додавання.
4. Засилання результату в регістр  $\text{Pr}1$ .

Програма виконання цих дій на одношинній структурі процесора (рис. 4.1) буде мати наступний вигляд:

Фаза вибірки :

1) ПЛout,  $\text{Pr}A\text{in}$ , зчитування, очищенння регістра  $\text{Pr}Y$ , включення переносу та операції додавання в АЛП,  $\text{Pr}Z\text{in}$  (адреса команди з програмного лічильника ПЛ подана на шину та записалась до регістра адреси  $\text{Pr}A$ , на вхід керування режимом роботи основної пам’яті подано сигнал Read, на вхід скиду регістра  $\text{Pr}Y$  подано сигнал Reset, в АЛП виконалась операція додавання 1 до вмісту програмного лічильника ПЛ та її результат записався в регістр  $\text{Pr}Z$ ).

2)  $\text{Pr}Z\text{out}, \text{PLin}$ , чекання підтвердження сигналу зчитування (результат додавання з регістра  $\text{Pr}Z$  записався в програмний лічильник ПЛ, команда з основної пам’яті записалась в регістр  $\text{Pr}D$ ).

3)  $\text{Pr}D\text{out}, \text{Pr}K\text{in}$  (команда з регістра даних  $\text{Pr}D$  записалась до регістра команди  $\text{Pr}K$ ).

Фаза виконання:

4) (Поле адреси PrK)out, PrAin, зчитування (адреса з поля адреси регістра команди PrK переписується до регістра адреси PrA, на вхід керування режимом роботи основної пам'яті подано сигнал Read).

5) PrIout, PrYin, чекання підтвердження сигналу зчитування (число з регістра PrI записалось до регістра PrY, а число з основної пам'яті записалось до регістра PrD).

6) PrDoout, Add, PrZin (число з регістра PrD поступило на шину та додалося в АЛП до числа з регістра PrY, а результат записався в регістр PrZ).

7) PrZout, PrIin, End (число з регістра PrZ переписалось в регістр PrI, кінець виконання операції).

Тут Add – код операції додавання, який поступає на вхід АЛП та вказує йому тип виконуваної операції.

#### **4.1.4.2. Виконання операції переходу**

Адреса переходу зазвичай одержується шляхом додавання значення X, яке знаходиться в адресному полі команди, до біжучого значення ПЛ.

Програма має наступний вигляд при виконанні безумовного переходу:

1) ПЛout, PrAin, зчитування, очищення регістра Y, включення переносу та операції додавання в АЛП, PrZin (адреса команди з програмного лічильника ПЛ подана на шину та записалась до регістра адреси PrA, на вхід керування режимом роботи основної пам'яті подано сигнал Read, на вхід скиду регістра PrY подано сигнал Reset, в АЛП виконалась операція додавання 1 до вмісту програмного лічильника ПЛ та її результат записався в регістр PrZ).

2) PrZout, ПЛin, чекання підтвердження сигналу зчитування (результат додавання з регістра PrZ записався в програмний лічильник ПЛ).

3) PrDoout, PrKin (команда з регістра даних PrD записалась до регістра команди PrK).

4) ПЛout, PrYin (число з програмного лічильника ПЛ записалось до регістра PrY).

5) (Поле адреси PrK)out, ADD, PrZin (число з поля адреси регістра команди PrK поступило на шину та додалося в АЛП до числа з регістра PrY, а результат записався в регістр PrZ).

6) PrZout, ПЛin, End (число з регістра PrZ переписалось в програмний лічильник ПЛ, кінець виконання операції).

При виконанні умовного переходу по значенню ПЛ = 0 крок 4 заміниться на наступний:

7) ПЛout, PrYin, if ПЛ=0 then End (запис вмісту програмного лічильника до регістра PrY і його аналіз на рівність 0. Якщо це так – кінець виконання операції).

#### **4.1.5. Особливості побудови процесора комп'ютера із складною системою команд**

Вище розглядалася структура та організація роботи процесора комп'ютера, який із середини вісімдесятих років минулого століття був віднесений до комп'ютерів з складною системою команд КССК (CISC). Аналізуючи його роботу і розглянуті в розділі 2 формат команди та склад системи команд названого комп'ютера, можна прийти до висновку, що для процесора комп'ютера із складною системою команд характерні наступні особливості:

- виконання команди за багато тактів, оскільки для цього потрібно здійснити багаторазові операції звернення до основної пам'яті та до програмно-доступних реєстрів процесора;
- орієнтація АЛП на виконання великої кількості операцій, що пов'язано з розширенім складом системи команд;
- складна система розпізнавання команди, що пов'язано з великою кількістю методів адресації та великою кількістю форматів команд різної розрядності;
- програмне дешифрування команд з метою зменшення затрат обладнання;
- складна організація конвеєризації виконання команд, що пов'язано, в першу чергу, з різноманітністю їх виконання;
- орієнтація структури на виконання команд типу реєстр-пам'ять та пам'ять-пам'ять.

Вказані особливості стримують побудову високопродуктивних комп'ютерів на основі процесора розглянутого типу. Вони були враховані при створенні процесорів комп'ютерів із простою системою команд.

## 4.2. Процесор комп'ютера з простою системою команд

### 4.2.1. Вимоги до процесора комп'ютера з простою системою команд

При розгляді системи команд комп'ютера ми ознайомилися з архітектурою комп'ютерів із простою системою команд КПСК (RISC). Виходячи з основних принципів реалізації цих комп'ютерів, можна виділити наступні вимоги, яких необхідно притримуватися при побудові їх процесора:

- Довільна комп'ютерна команда, незалежно від її типу, має виконуватися за один такт (чи однотактовий цикл).
- Пристрій керування та арифметико-логічний пристрій процесора мають орієнтуватися на виконання мінімальної кількості спрощених команд, що статистично переважають у програмах, причому в системі команд відносно небагато операцій та режимів адресації операндів (способів адресації).
- Команди обробки даних мають реалізуватися лише у формі “реєстр-реєстр”. Обміни з основною пам'яттю виконуються лише за допомогою команд завантаження/запису (архітектура load/store).
- Дешифрування команд із спрощеними форматами має виконуватися лише апаратно, аби збільшити швидкодію.
- Необхідно забезпечити високий рівень конвеєризації виконання команд.
- Регістрова пам'ять має включати велику кількість програмно-доступних реєстрів.

При цьому необхідно проводити оптимізацію структури процесора, що проектується, з метою забезпечення найшвидшого виконання обраних команд та передбачити можливість додавання до отриманого списку інших команд, якщо вони не ускладнюють процесора.

### 4.2.2. Базові принципи побудови процесора комп'ютера з простою системою команд

Для ілюстрації базових принципів побудови процесора комп'ютера з простою системою команд ми використаємо архітектуру комп'ютера, яка була запропонована для навчальних цілей Джоном Хеннессі та Дейвідом Паттерсоном і отримала назву DLX. Ця

архітектура узагальнює особливості архітектур наступних сучасних комп'ютерів: AMD 29000, DEC3100, HP850, IBM801, Intel860, MIPS M/120A, MIPS M/1000, M88000, RISC1, SGI 4D/60, SPARCstation-1, SUN-4/110, SUN-4/260.

Регістровий файл процесора комп'ютера DLX вміщує 32 регістри загального призначення (R0...R31) для зберігання цілих чисел, 32 регістри (F0...31) для зберігання даних з рухомою комою. Набір команд цього комп'ютера включає типові арифметичні й логічні операції, операції з фіксованою та рухомою комою, операції пересилання даних, операції керування потоком команд і системні операції. У арифметичних командах використовується триадресний формат, типовий для комп'ютерів з архітектурою КПСК, а для звернення до пам'яті використовуються операції завантаження і запису вмісту регістрів у пам'ять.

Основою проектування структури процесора комп'ютера з простою системою команд є часова діаграма виконання команд з найбільшою складністю, до числа яких належить, зокрема, команда завантаження слова. Розглянемо цикл виконання команди вибірки з основної пам'яті (завантаження) слова LW R5, 16(R26). В комп'ютері DLX командний цикл поділений на п'ять фаз. Тому для виконання вказаної команди потрібно виконати наступні фази:

- вибрati зазначену команду з основної пам'яті (перша фаза виконання команди із назвою IF (Instruction Fetch));
- декодувати команду та вибрati операнди (друга фаза виконання команди із назвою ID (Instruction Detecting));
- виконати команду, тобто обрахувати виконавчу адресу операнда  $16 + [R26]$  (третя фаза виконання команди із назвою EX (Execution));
- вибрati операнд із основної пам'яті (четверта фаза виконання команди із назвою MEM (Memory));
- переслати вибраний з основної пам'яті операнд до регістра R5 регістрового файла (п'ята фаза виконання команди із назвою WB (Write Back)).

Використані назви фаз дещо узагальнюють притаманну лише команді LW семантику кожної окремої фази. Це коректно, бо нашою метою є наближення до такої послідовності фаз, яка задовольняє вимогам будь-якої команди із заданої до реалізації множини з метою досягнення найбільшої швидкодії. Інші команди не завжди вимагають реалізації усього переліченого набору фаз, тому що мають меншу часову складність.

На рис. 4.4 наведена часова діаграма виконання п'ятифазової команди завантаження LW. Залежно від структури процесора ця команда може бути виконаною за різний час, який буде складатися з суми проміжків часу, необхідних для виконанняожної фази. Розглянемо підхід до побудови процесора з тим, щоб задоволити вимогу, згідно з якою довільна комп'ютерна команда, незалежно від її типу, має виконуватися за один такт (чи однотактовий цикл), яка ставиться до процесорів комп'ютера з простою системою команд.



Рис. 4.4. Часова діаграма виконання п'ятифазової команди завантаження LW

Представимо алгоритм виконання команди у вигляді потокового графа, кожна з вершин якого позначає оператор відповідної фази виконання команди (рис. 4.5а).

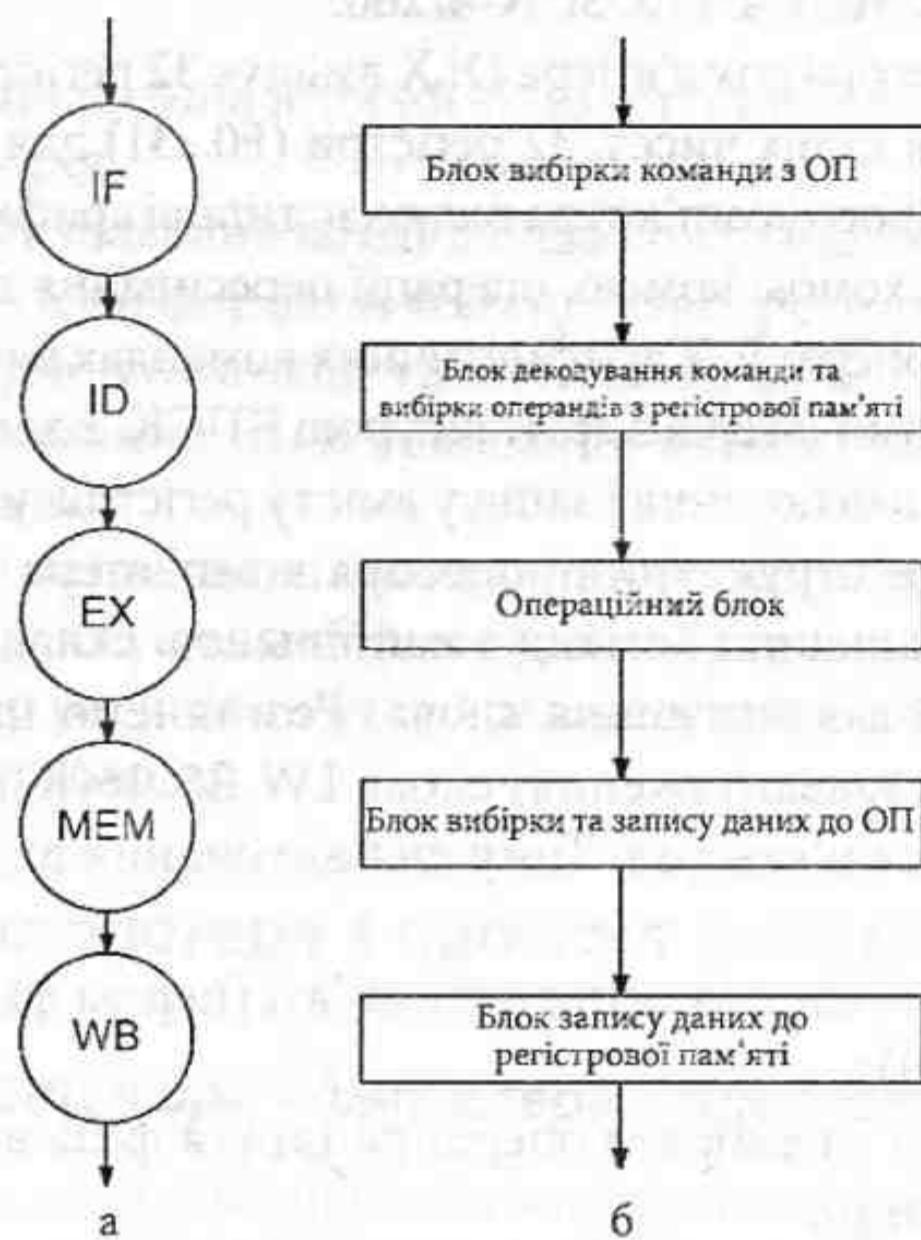


Рис. 4.5 а – отоковий граф виконання команди; б – структура процесора, орієнтованого на його реалізацію

Для того, щоб команда виконувалася за один такт, потрібно апаратно відобразити алгоритм її виконання, тобто поставити у відповідність кожному оператору алгоритму функціональні вузли процесора, які їх виконують, та з'єднати їх між собою згідно із зв'язками вершин потокового графа алгоритму. Тоді структура процесора комп'ютера з простою системою команд, який виконує названі фази, може бути подана наступним рисунком (рис. 4.5б). Як бачимо, процесор містить п'ять послідовно з'єднаних блоків: вибірки команди з основної пам'яті, декодування операндів та вибірки команди з реєстрової пам'яті, операційний, вибірки та запису даних до основної пам'яті, запису даних до реєстрової пам'яті. Кожен з цих блоків виконує відповідну фазу командного циклу та передає результати до наступного блоку. Результатом послідовної роботи цих блоків є виконання команди.

Деталізована структура процесора комп'ютера DLX, яка побудована на основі описаного вище підходу, представлена на рис. 4.6.

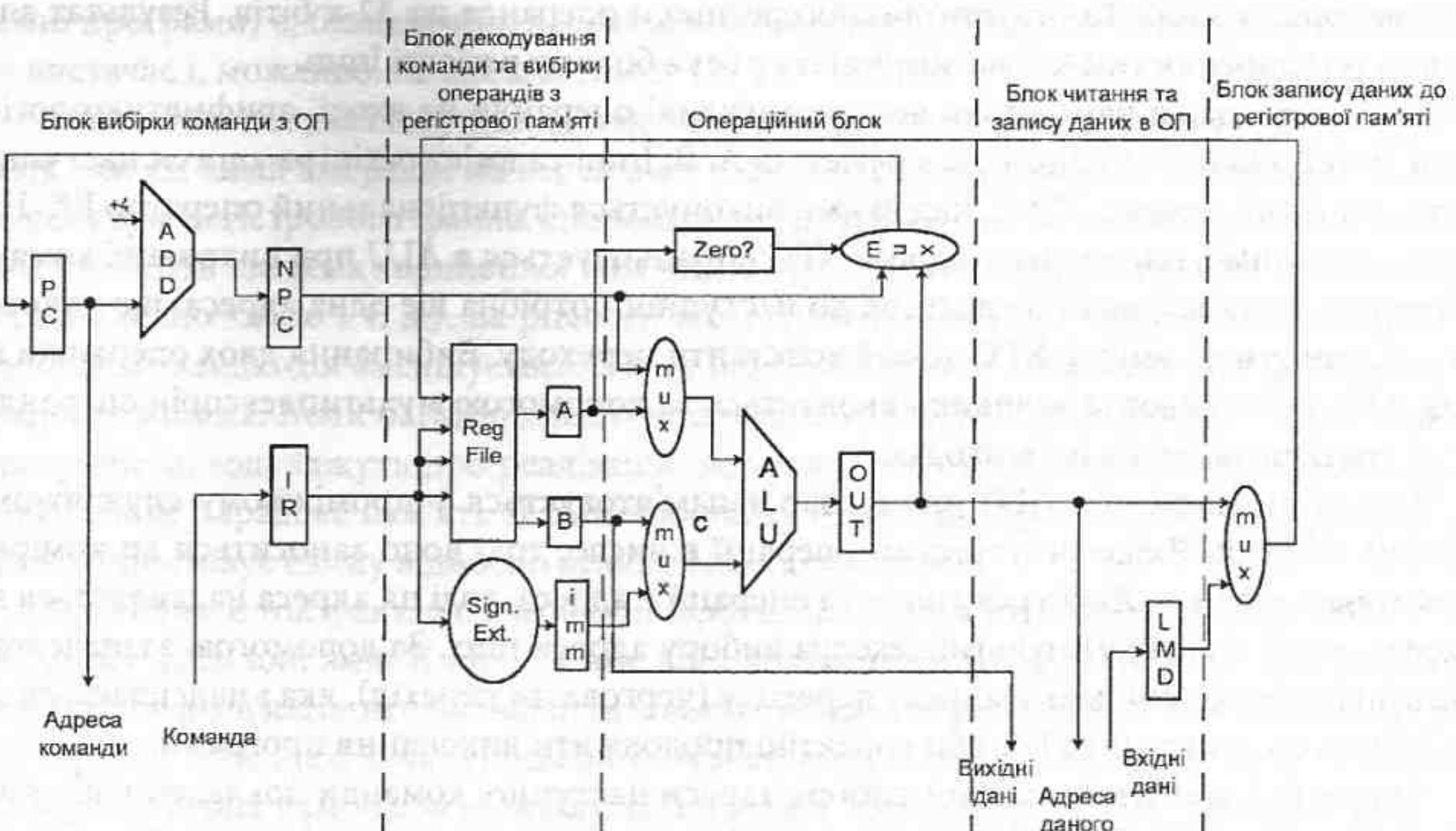


Рис. 4.6. Структура процесора комп’ютера з простою системою команд

На цій схемі лініями відділені блоки процесора, показані на рис. 4.5б, які виконують відповідну фазу командного циклу потокового графа алгоритму виконання команди. Перший оператор IF виконується на наступних елементах: програмному лічильнику PC (Program Counter), суматорі Adder та двох реєстрах NPC і IR. Вміст програмного лічильника PC визначає адресу команди в основній пам’яті. Комбінаційний суматор Adder обирає адресу наступної за чергою виконання команди. При цьому враховано, що впорядкована послідовність команд (програма) складається з чотирибайтових команд (усі команди мають формати довжиною 32 біти), які розміщено в основній пам’яті за послідовними адресами 0, 4, 8, C і т. д. Через це константа зсуву адреси (пересування по-кажчика на наступну за чергою команду) дорівнює +4. Визначене за допомогою суматора значення адреси вибрання наступної команди зберігається у реєстрі NPC (next PC). Зчитаний з основної пам’яті код поточної команди записується до реєстра команди IR.

Поля щойно вибраної команди містять адреси програмно-доступних реєстрів реєстрової файлів процесора. Вміст зазначених полів формату команди в рамках оператора ID надсилається на адресні входи реєстрової пам’яті Regs, а відповідні надісланим адресам коди операндів завантажуються до внутрішніх, програмно-недосяжних, тобто службових, реєстрів A і B.

Існує ще один тип операнда з назвою “безпосередній” (Imm). Його задають прямо у форматі команди. Як правило, довжина безпосереднього операнда не перевищує половини довжини формату команди. В комп’ютері DLX безпосередній операнд має довжину  $32/2 = 16$  бітів. У той самий час бажано зафіксувати довжину формату даних такою, що дорівнює довжині формату команди, адже різноманітість довжин форматів суттєво пригальмовує комп’ютер. Якщо усі формати даних, як і формати команд, матимуть довжину 32 біти, тоді безпосередньому операнду не вистачатиме ще 16 бітів, аби бути стандартним за довжиною. Тому тут використаний комбінаційний вузол Sign Extend, який виконує

знакове розширення 16-бітового безпосереднього операнда до 32-х бітів. Результат знакового розширення тимчасово зберігають у службовому реєстрі Imm.

В цілому можна нарахувати чотири можливі операнди на вході арифметико логічного пристрою ALU процесора: з реєстрів A, B, Imm та вміст реєстра адреси наступної для виконання команди NPC, над якими виконується функціональний оператор EX. Наперед зазначимо, що операнд-адреса NPC опрацьовується в ALU при виконанні команд умовного переходу, коли на додаток до наступної потрібна ще одна адреса, що утворена додаванням до вмісту NPC деякої константи переходу. Вибирання двох операндів на вхід ALU із чотирьох можливих виконується за допомогою мультиплексорів операндів тих, розташованих на його входах.

Результат операції з ALU тимчасово запам'ятовується у проміжному службовому реєстрі ALUout. Якщо результатом операції є число, тоді воно заноситься до комірки реєстрового файла. Якщо результатом операції є адреса, тоді ця адреса надсилається до (верхнього на рисунку) мультиплексора вибору адреси тих. За допомогою зазначеного мультиплексора вибирають адресу переходу (чергова чи переход), яка і надсилається до програмного лічильника PC, аби коректно продовжити виконання програми.

Керування мультиплексором вибору адреси наступної команди покладено на вузол Zero?, де вміст службового реєстра A порівнюється із нулем (дорівнює нулю, більше нуля, менше нуля і т. д., залежно від виду виконуваної у поточний час операції умовного переходу). Результат порівняння є бінарним логічним значенням (так або ні). Саме цей бінарний результат керує роботою мультиплексора вибирания адреси наступної команди.

При виконанні фази MEM результат-адреса з виходу ALU надсилається до основної пам'яті як отримана адреса комірки цієї пам'яті (для команд збереження/завантаження)

Результатом на виході правого на рисунку мультиплексора може бути або вміст основної пам'яті (при виконанні команди завантаження LW слова з основної пам'яті до реєстра реєстрового файла), або результат виконання арифметичної, зсувної, логічної чи іншої операції в ALU (наприклад, при виконанні команд ADD, SUB і т. д.). Такий результат в рамках виконання фази WB засобами мікропрограмування зберігають в реєстрі реєстрового файла. Отже, зазначений мультиплексор, керований реєстром поточної команди, комутує на вхід реєстрового файла потрібну інформацію

Таким чином, апаратно відобразивши потоковий граф алгоритму виконання команди, вдалося забезпечити вимогу, щоб вона виконувалася за один такт. Як видно з вище приведеного опису роботи процесора, для спрощення пояснення тут були використані проміжні реєстри для запису операндів, які можуть бути видалені. Порівняно з процесором комп'ютера із складною системою команд виконання команди в приведеній структурі процесора суттєво спростилося. Далі розглянемо це детальніше, але спочатку проведемо аналіз взаємодії процесора з основною пам'яттю.

#### **4.2.3. Взаємодія процесора з пам'яттю в комп'ютері з простою системою команд**

Відомий так званий парадокс пам'яті – пам'ять може мати малий об'єм, проте бути швидкою і задовольняти вимоги процесора щодо швидкодії, або мати відносно великий об'єм і бути повільною. Немає пам'яті відносно великої і, водночас, швидкої. Ставити зараз питання про основну пам'ять, об'єм якої задовольняє системного програміста (сис-

темні програми) є, безперечно, нереальним завданням. Об'єму основної пам'яті завжди не вистачає і, можливо, не вистачатиме.

Аби подолати зазначену невідповідність, вибудовують багаторівневу ієархічну систему пам'яті комп'ютера, де на верхньому рівні ієархії знаходяться програмно-доступні реєстри регистрового файла процесора, на другому – кеш, потім комірки основної пам'яті, потім система зовнішньої пам'яті (диск, магнітні стрічки, архівна пам'ять з лазерною технологією і т. д.). За рівнями ієархії, згори донизу, об'єм пристрій пам'яті зростає, а швидкодія зменшується. Керує інформаційними обмінами між рівнями операційна система. Коли багаторівневість та ієархію пам'яті приховано від прикладного програміста, тоді кажуть про реалізацію віртуальної пам'яті, де, ніби, скасовано зазначений вище парадокс пам'яті. В цьому випадку прикладному програмісту здається, що він використовує єдину відносно велику і швидку пам'ять.

Важливим є той факт, що звернення процесора до пам'яті завжди локалізовано в невеликому діапазоні змін її адрес. Саме він і дозволяє застосовувати ієархічну систему пам'яті, аби розв'язати невідповідність швидкодії процесора і системи пам'яті з одним рівнем ієархії. Між процесором і основною пам'яттю розташована кеш пам'ять (рис. 4.7) – це швидка буферна пам'ять невеликого об'єму. Кеш пам'ять працює на близькій тактовій частоті до процесора і не пригальмовує його роботу. Кеш (cache у перекладі з англійської – тайник) лишається прозорою для програміста, тому що система команд процесора, як правило, не містить команд роботи з кеш пам'яттю.



Рис. 4.7. Кеш пам'ять у складі ядра комп'ютера

При поясненні роботи кеш пам'яті можна допустити, що процесор також не «бачить» кеш і генерує адреси пам'яті так, ніби кеш пам'яті не має. Проте кеш пам'ять, як правило, існує, і на апаратному рівні перехоплює сигнали процесора читання/запису, а коли треба, то надає процесору швидкі копії інформаційних кодів, які тимчасово зберігає у власній робочій пам'яті. Якщо кеш пам'ять спроможна підмінити собою основну пам'ять (у понад 96–98 відсотків випадків), тоді вона за рахунок власних ресурсів задовільняє запит процесора. Процесор не пригальмовується і продовжує працювати на повній швидкості. Коли «підміна» основної пам'яті неможлива (менше від двох–четирьох відсотків випадків), тоді кеш пам'ять залучає до роботи основну пам'ять, обмін з якою суттєво пригальмовує процесор.

Усі завдання, пов'язані із перехопленням запитів від процесора до основної пам'яті, вирішує контролер кеш пам'яті, який є її складовою частиною. Другою частиною кеш пам'яті є невелика робоча пам'ять, де зберігають вміст копій комірок основної пам'яті, що брали участь в обслуговуванні останніх, тобто найбільш «свіжих» запитів процесора. Важливо, що вміст комірок основної пам'яті копіюється до кеш пам'яті разом із своїми адресами. Саме ці копійовані адреси і дозволяють контролеру кеш пам'яті приймати

рішення про спроможність задовольнити конкретний процесорний запит без залучення до обміну повільної основної пам'яті.

Спрощений варіант структури комп'ютера, в якому використовується кеш пам'ять, подано на рис. 4.8. Пристрій керування надсилає керуючі сигнали до процесора та основної пам'яті. З процесора сигнали станів, якими можуть бути біти регістра команди і інше, надходять до пристрою керування, аби реалізувати розгалуження мікропрограми.

Для зберігання даних і команд використано розділені кеш пам'яті даних і команд Гарвардської архітектури. В свою чергу, кеш пам'яті зв'язані з єдиною пам'яттю Принстонської архітектури. Ясно, що обмін в підсистемі «основна пам'ять – кеш пам'ять даних» є двостороннім, а в підсистемі «основна пам'ять – кеш пам'ять команд» – одностороннім.



Рис. 4.8. Спрощена структура комп'ютера DLX

Можна побачити, що наведена структура ефективно поєднала риси Принстонської та Гарвардської архітектур.

Надалі при розгляді принципів виконання команд в процесорі будемо розглядати і його взаємодію з основною пам'яттю через кеш пам'ять команд та даних.

#### 4.2.4. Виконання команд в процесорі комп'ютера з простою системою команд

##### 4.2.4.1. Фаза вибирання команди

Виконання команд в процесорі комп'ютера з простою системою команд розглянемо на прикладі процесора комп'ютера DLX, структура якого була розглянута в п. 4.2.2.

В рамках фази вибирання команди IF виконуються наступні мікродії (мікрооперації):

$$IR = IM [PC];$$

$$NPC = PC+4.$$

Виконання першої мікродії спричинює завантаження до 32-бітового реєстра поточної команди IR вмісту чотирьох послідовно розташованих комірок пам'яті команд, починаючи від адреси  $(PC) + 0$  і завершути адресою  $(PC) + 3$ . Як і завжди, за допомогою  $(PC)$  позначене вміст програмного лічильника PC.

Друга мікродія ( $NPC = PC + 4$ ) вираховує “планову” адресу наступної за чергою команди з послідовного потоку. Тобто тут визначається і тимчасово зберігається у реєстрі NPC вміст програмного лічильника. “Логічне” вибирання вмісту чотирьох послідовно розташованих однобайтових комірок замість однієї чотирибайтової підкреслює те, що навіть на рівні мікродій адресування комірок пам'яті вказують логічно. За умови коли адреса байта кратна чотирьом, тобто дотримано вирівнювання границь адрес команд, із пам'яті, сразу (за одне звернення) вибирають чотири байти і навіть цілі пакети по чотири байти. Отриманий з пам'яті даних код тлумачать як 32-розрядне слово, а з пам'яті команд – як одну команду

Зауважимо, що обидві наведені мікродії теоретично є сумісними в часі. Саме тому вони можуть і мати виконуватися паралельно. Ці мікродії утворюють єдину мікрокоманду

Важливою властивістю фази IF є те, що вона не приймає до уваги існування відомого «парадокса пам'яті». Дійсно, у фазі IF за часовими витратами мікродія вибирання з повільних комірок пам'яті команд є тотожною мікродії, що працює з внутрішніми, а саме тому надшвидкими реєстрами процесора

#### 4.2.4.2. Фаза декодування команди

В рамках фази декодування команди ID виконуються наступні мікродії (мікрооперації):

$$\begin{aligned} A &= \text{Regs}[\text{IR}_{6..10}]; \\ B &= \text{Regs}[\text{IR}_{11..15}]; \\ \text{Imm} &= ((\text{IR}_{16})^{16} \# \# \text{IR}_{16..31}). \end{aligned}$$

У наведеному мікроході, що складений з трьох сумісних у часі, тобто придатних до одночасного (паралельного) виконання мікродій, вжито наступні позначення:

- A, B, Imm – внутрішні службові 32-розрядні реєстри для проміжного збереження кодів; всі три реєстри є “прозорими” для програміста в тому сенсі, що вони аж ніяк не відбиті в машинних командах.
- Regs[address] – номер реєстра реєстрового файла процесора;
- R<sub>x..y</sub> – поле реєстра команд, яке містить групу послідовно розташованих бітів – від біта з номером X до біта з номером Y включно; звернемо увагу на нумерацію бітів у слові процесора – лівий біт має 0-й номер, а правий – 31-й номер.
- IR<sub>6..10</sub> – поле реєстра команд, що містить бінарний номер реєстра – джерела даного (див. формат команд); довжина поля рівна 5 бітам, що дозволяє позначати та адресувати 32 реєстри – від R<sub>0</sub> до R<sub>31</sub>
- IR<sub>11..15</sub> – також п'ятибітове поле номера ще одного реєстра – джерела даного з множини R<sub>0..R<sub>31</sub></sub>

Вже зауважувалося, що поле безпосереднього операнда (Immediate або Imm) у форматі команди має довжину лише 16 бітів. В той же час, розрядність інформаційного тракту в усіх трьох службових реєстрів рівна 32 біти. До того, як записати 16-бітовий код IR<sub>16..31</sub> безпосереднього операнда з реєстра команд до службового реєстра Imm, цей

код треба розширити з врахуванням знака (тобто розряду  $IR_{16}$ ) до 32-бітового значення а наступним стандартним алгоритмом знакового розширення:

$$IR_{16} \# \# IR_{16} \# \# \dots \# \# IR_{16} \# IR_{16..31}$$

Тут символом  $\# \#$  позначено операцію зчеплення (конкатенацію).

Важливо, що в рамках фази ID виконується декодування (роздільання) команди та вибирання усіх можливих варіантів операндів поточної команди, незалежно від її типу, з метою збільшення швидкодії. З іншого боку, можливість одночасного вибирання усіх варіантів операндів потенційно обумовлено відповідною структурою форматів команд, де фіксовано розташування полів-показників на джерела даних та приймачі операндів і результатів. Іншими словами, формати команд процесора втілюють техніку кодування форматів команд, що відома під назвою “fixed-field coding”.

#### 4.2.4.3. Фаза виконання та формування ефективної адреси

Мікродії, що виконують в рамках фази виконання та визначення ефективної адреси EX, вже залежать від типу поточної команди. Враховуючи це, вигідно поділити усі команди процесора на наступні три групи:

- команди виконання операцій завантаження операндів з основної пам'яті та збереження результатів у основній пам'яті;
- команди виконання операцій арифметико-логічного пристрою;
- команди виконання операцій умовних та безумовних переходів.

Далі розглядаємо притаманні кожній групі команд мікродії

#### Команди виконання операції звернення до пам'яті даних

В рамках фази EX операції звернення до пам'яті даних (load/store instructions) готують значення ефективної адреси, тобто вираховують значення бінарного коду, що є адресою комірки пам'яті даних. Виконується наступна мікродія:

$$ALUoutput = A + Imm.$$

Позначене в мікродії додавання безпосереднього операнда (однієї з компонент адреси) з вмістом робочого реєстра A (другої компоненти) реалізується в ALU. Отримана сума (вона завжди ціла і додатна, переносом нехтують) записується до ще одного, прозорого для програміста, реєстра ALUoutput. Цей реєстр має розрядність 32 біти. Він зберігає коди результатів, що з'являються на виході комбінаційної схеми з назвою ALU. Якщо пригадати дію команди LW, тоді призначення та форма запису наведеної мікродії стає зрозумілою.

#### Команди виконання операції арифметико логічного пристрою типу реєстр реєстр.

Прикладом команди типу реєстр-реєстр є команда ADD R1,R2,R3. Виконується вказаною командою наступна мікродія

$$ALUoutput = A op B.$$

Тут узагальнене позначення (op) можна конкретизувати як (add), (sub) тощо, залежно від конкретного виду поточної команди, яка опрацьовується в інформаційному тракті комп'ютера. Підкреслимо, що на попередній щодо EX фазі, а саме на фазі ID, до службових реєстрів уже завантажено вміст вибраних реєстрів реєстрового файла з адресами R2 і R3. Результат дії поки що не збережено в реєстрі R1, але його тимчасово записано до службового реєстра ALUoutput.

**Команда виконання операції арифметико-логічного пристрою типу регістр-безпосередній операнд.**

Прикладом команди типу регістр-безпосередній операнд є команда ADD R1,R2,#23. Виконується наступна мікродія:

$$\text{ALUoutput} = A \ op \ Imm.$$

Ясно, що замість вмісту реєстра R3 в операції бере участь безпосередній операнд, який з урахуванням «знакового» розширення до 32-х бітів читається із службового реєстра Imm.

### **Команда умовного переходу.**

Прикладом команди умовного переходу є команда BNEZ R5, data. Виконуються наступні мікродії:

$$\begin{aligned}\text{ALUoutput} &= \text{NPC} + \text{Imm}; \\ \text{Cond (ition)} &= (A \ op \ 0).\end{aligned}$$

Обидві мікродії є сумісними в часі. За допомогою першої мікродії вираховується цільова адреса умовного переходу. При цьому до вже визначеній адреси наступної команди (NPC), тобто такої, яка розташована безпосередньо за командою умовного переходу, додається константа зі службового реєстра Imm, (саме він містить значення data). Друга мікродія визначає (істинним чи хибним) є значення умови Condition умовного переходу. Заради цього вміст службового реєстра A, що є збіжним із вмістом реєстра R5 (у команді-прикладі), порівнюється на основі операції (op) з нулем. Згідно з семантикою команди BNEZ отримується, що Cond = true, коли вміст реєстра R5 ненульовий, або Cond = false, якщо реєстр R5 містить нуль. Фізично Cond є однобітним реєстром у складі вузла Zero?

Звернемо увагу на те, що зараз сформовано лише значення двох необхідних елементів виконання умовного переходу, а саме: однобітний код умови Cond та цільова адреса переходу, яка тимчасово зберігається у службовому вихідному реєстрі ALU, тобто в ALUoutput. Безпосереднє виконання умовного переходу, що полягає в природній (за чергою) чи неприродній (коли виконують стрибок до цільової адреси) зміні вмісту програмного лічильника PC, виконується на наступній фазі.

#### **4.2.4.4. Фаза звернення до пам'яті та завершення умовного переходу**

##### **Звернення до пам'яті.**

В рамках фази звернення до пам'яті MEM виконуються наступні мікродії:

$$\begin{aligned}\text{LMD} &= \text{DM} [\text{ALUoutput}]; \\ \text{DM} [\text{ALUoutput}] &= \text{B}.\end{aligned}$$

Звернення до пам'яті застосовується в командах завантаження (наприклад, LW R6, 112(R3)) та в командах збереження (наприклад, SW 112(R3),R6).

Перша мікродія виконується тільки у випадку команди Load (завантаження). Тут за допомогою попередньо розрахованої адреси пам'яті даних, яка тимчасово зберігається у службовому реєстрі ALUoutput, здійснюється доступ до пам'яті, що і позначено записом DM[ALUoutput].

Фізично з пам'яті завжди читається 32 біти (або навіть цілі пакети з 32-х бітових структурних одиниць). Отриманий з пам'яті даних бінарний код тимчасово завантажу-

ється до ще одного службового реєстра LMD (Load Memory Data). І тільки на наступній фазі WB, яка поки що не розглядалася, зчитаний код пересилається з реєстра LMD до конкретної комірки (в нашому прикладі – це комірка з адресою R6) реєстрового файла.

Друга мікродія реалізується лише при виконанні команди Store (збереження). Тут до комірки пам'яті даних за адресою, що зберігається в службовому реєстрі ALUoutput = (R3) + 112, засилається вміст службового реєстра В. При цьому (в наведеному прикладі) для команди SW має місце тотожність (B)=(R6).

#### **Умовний переход.**

Виконується наступна мікродія:

`if (condition) PC = ALUoutput else PC = NPC.`

Здійснюється природна (cond=0) або неприродна (cond =1) заміна вмісту програмного лічильника PC з метою реалізації наступної за умовним переходом команди.

#### **4.2.4.5. Фаза зворотного запису**

На цій, вже останній фазі виконання команди, у разі потреби результат, що отримано на попередніх фазах, записується до деякої комірки RX реєстрового файла. Наприклад, в R1, якщо поточна є команда ADD R1,R2,R3, або в R6 для команди LW R6,#112(R3). Тому вона і має назву зворотного запису (write/back – WB).

Потрібно звернути увагу, що на цій фазі змінюється програмний стан комп'ютера. Після її виконання поновлення попереднього стану є неможливим.

**Команди виконання операції арифметико-логічного пристрою типу “реєстр-реєстр”.**

Виконується наступна мікродія:

$\text{Regs[IR}_{16\ldots 20}\text{]} = \text{ALUoutput}.$

Мікродія зберігає результат операції ALU в реєстрі призначення (наприклад, в реєстрі R1 на прикладі команди ADD). Зрозуміло, що біти 16...20 відповідного формату команди містять бінарний номер реєстра призначення.

**Команди виконання операції арифметико-логічного пристрою типу “реєстр-безпосередній операнд”.**

Виконується наступна мікродія:

$\text{Regs[IR}_{11\ldots 15}\text{]} = \text{ALUoutput}.$

Ця мікродія також зберігає результат операції, наприклад, SUB R5, R4, #1002, в комірці R5 реєстрового файла.

#### **Команда завантаження.**

Прикладом команди завантаження є команда LW R6,112(R3). Виконується наступна мікродія:

$\text{Regs[IR}_{11\ldots 15}\text{]} = \text{LMD}.$

Наведена мікрокоманда надсилає до комірки R6 реєстрового файла вміст комірки пам'яті даних за адресою 112 + (R3), яке на попередніх фазах вже було вибране з пам'яті і тимчасово зберігалося в службовому реєстрі LMD. На цьому опис мікродій керування роботою комп'ютера DLX завершено. В табл. 4.1. наведено приклади алгоритмів виконання команд в комп'ютері DLX.

Таблиця 4.1

Приклади алгоритмів виконання команд в комп'ютері DLX

Приклад команди	Алгоритм виконання команди
<b>Арифметичні та логічні команди</b>	
ADD R1, R2, R3	$Regs[R1] = Regs[R2] + Regs[R3]$
ADDI R1, R2, #3	$Regs[R1] = Regs[R2] + 3$
LHI R1, #42	$Regs[R1] = 42 \# 0^{16}$
SLLI R1, R2, #5	$Regs[R1] = Regs[R2] \ll 5$
SLT R1, R2, R3	$\text{if } (Regs[R2] < Regs[R3]) \text{ } Regs[R1] = 1 \text{ else } Regs[R1] = 0$
<b>Команди збереження й завантаження</b>	
LW R1, 30(R2)	$Regs[R1] = {}_{32} \text{DM}[30 + Regs[R2]]$
LW R1, 1000(R0)	$Regs[R1] = {}_{32} \text{DM}[1000 + 0]$
LB R1, 40(R3)	$Regs[R1] = {}_{32} (\text{DM}[40 + Regs[R3]])_0 \# {}_{32} \text{DM}[40 + Regs[R3]]$
IUB R1, 40(R3)	$Regs[R1] = {}_{32} 0^{24} \# {}_{32} \text{DM}[40 + Regs[R3]]$
LH R1, 40(R3)	$Regs[R1] = {}_{32} (\text{DM}[40 + Regs[R3]])_0^{16} \# {}_{32} \text{DM}[40 + Regs[R3]] \# {}_{32} \text{DM}[41 + Regs[R3]]$
SW 500(R4), R3	$DM[500 + Regs[R4]] = {}_{32} Regs[R3]$
SH 502(R2), R3	$DM[502 + Regs[R2]] = {}_{16} Regs[R3]_{16...31}$
SB 41(R3), R2	$DM[41 + Regs[R3]] = {}_8 Regs[R2]_{24...31}$
<b>Команди керування програмою</b>	
J name	$PC = \text{name};$ $((PC + 4) - 2^{25}) \leq \text{name} \leq ((PC + 4) + 2^{25});$
JAL name	$R31 = PC + 4; PC = \text{name};$ $((PC + 4) - 2^{25}) \leq \text{name} \leq ((PC + 4) + 2^{25});$
JALR R2	$Regs[R31] = PC + 4; PC = Regs[R2];$
BEQZ R4, name	$\text{if } (Regs[R4] == 0) PC = \text{name};$ $((PC + 4) - 2^{15}) \leq \text{name} \leq ((PC + 4) + 2^{15});$
BNEZ R4, name	$\text{if } (Regs[R4] != 0) PC = \text{name};$ $((PC + 4) - 2^{15}) \leq \text{name} \leq ((PC + 4) + 2^{15});$

Пояснимо на прикладі синтаксис запису алгоритмів виконання окремих машинних команд комп'ютера DLX (див. таблицю 4.1). Розглянемо запис:

$$\text{Regs}[R19]_{16...31} = {}_{16}(\text{DM}[Regs[R8]])_0^8 \# {}_{16} \text{DM}[Regs[R8]].$$

Запис означає наступне. Оновлюють лише 16 молодших (правих) бітів регістра R19. До них пересилається двобайтовий бінарний код, в якому молодший правий байт береться з пам'яті даних DM за адресою, що є збіжною з вмістом регістра R8. Старший лівий байт утворюється восьмиразовим повторюванням нульового (старшого) розряду щойно згаданого правого байта. Парою символів  $\#$  позначено операцію конкатенації (зчленення) двох байтів до двобайтового півслова. Наведеною таблицею 4.1 достатньо, аби синтезувати неподані алгоритми виконання інших команд комп'ютера DLX.

#### 4.2.5. Конвеєрна структура процесора комп'ютера з простою системою команд

##### 4.2.5.1. Конвеєрний процесор

Конвеєрну структуру процесора комп'ютера з простою системою команд розглянемо на прикладі вище описаного комп'ютера DLX. Вище виконання типової команди в комп'ютері DLX було розділено на наступні фази:

1. IF – вибірка команди (за адресою, заданою лічильником команд, із пам'яті читається команда).

2. ID – декодування команди/вибірка операндів з регістрів.

3. EX – виконання операції та обчислення ефективної адреси пам'яті.

4. MEM – звернення до пам'яті.

5. WB – запам'ятування результату.

На рис. 4.6 представлена схема процесора, що виконує вказані вище фази виконання команд без перекриття. Щоб конвеєризувати цю схему, можна просто розбити виконання команд на вказані вище фази, відвівши для виконанняожної фази один такт синхронізації, і починати в кожному такті виконання нової команди. Природно, для зберігання проміжних результатівожної фази необхідно використовувати конвеєрні регістри. На рис. 4.9 показана схема процесора з конвеєрними регістрами, які забезпечують передачу даних і керуючих кодів з одного ярусу конвеєра на наступний.Хоча загальний час виконання однієї команди в такому конвеєрі складатиме п'ять тактів, у кожному такті апаратура виконуватиме в суміщеному режимі п'ять різних команд.

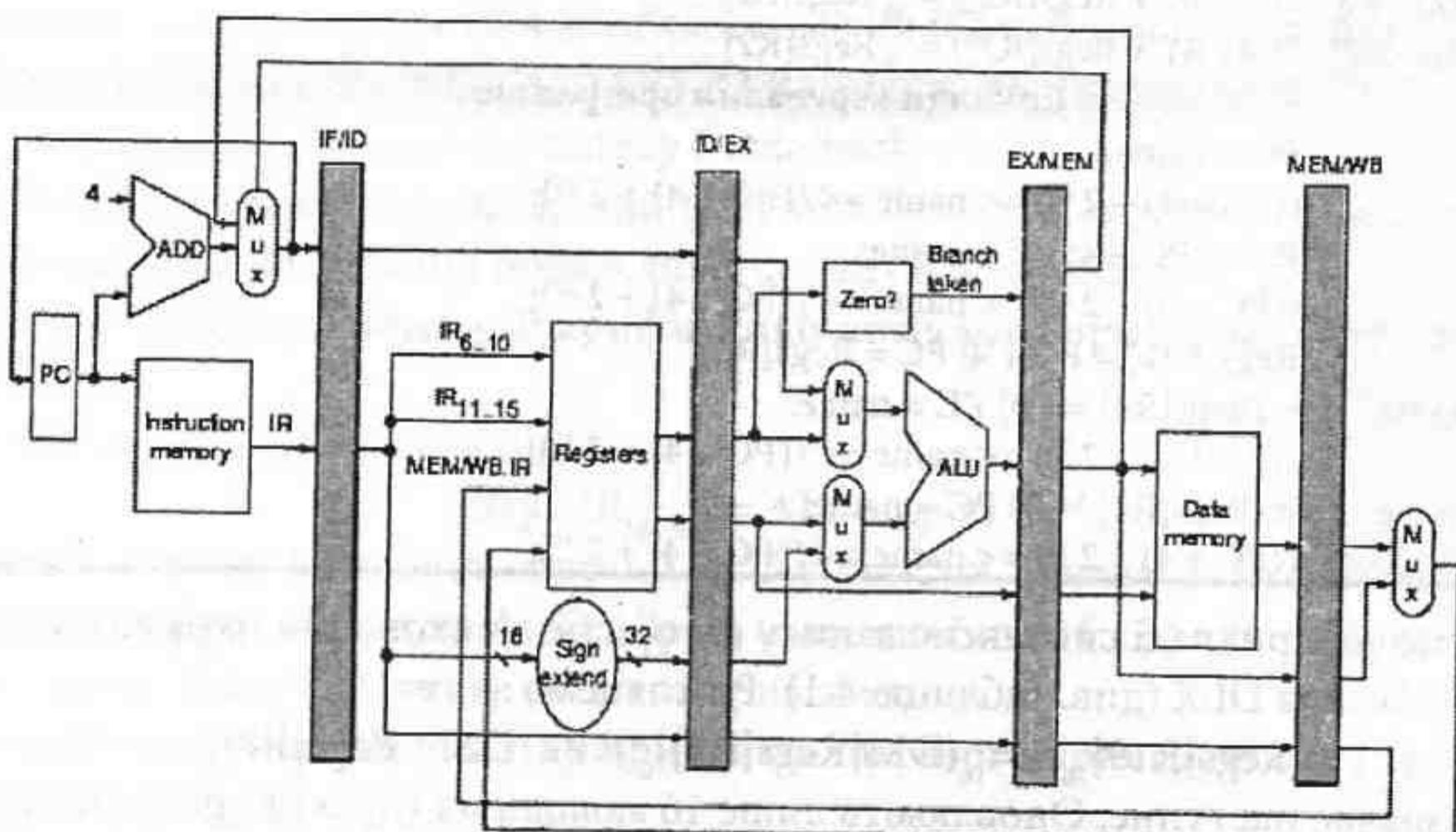


Рис. 4.9. Конвеєрна структура процесора комп'ютера DLX (з кеш пам'яттю даних та команд)

Конвеєрний процесор комп'ютера DLX структуровано наступними ярусами конвеєра: IF, ID, EX, MEM, WB. Апаратура кожного ярусу реалізує притаманні їй мікрооперації. Наприклад, на першому ярусі виконується вибирання команди з пам'яті команд IM за вмістом програмного лічильника PC, приріст (інкремент) на +4 (з врахуванням логічного байтового адресування пам'яті команд) поточної адреси за допомогою комбінаційного суматора ADD та занесення значення наступної адреси до поля NPC (Next PC), інтегрованого до конвеєрного регістра IF/ID. Мультиплексор Mux, що керується відповідним однобітним полем конвеєрного регістра EX/MEM, визначає джерело запису до NPC – або наступна за чергою адреса, або цільова адреса умовного чи безумовного переходу. Важливо, що обов'язково змінювати природне адресування послідовності вибирання команд з пам'яті команд покладено на вміст команди, яка пройшла фазу конвеєра MEM.

Конвеєрні реєстри виконують функцію збереження вмісту інтегрованих до них реєстра команди IR, робочих реєстрів A, B тощо. Конвеєрні реєстри розташовано на межах ярусів. Вони мають назви, відповідні граничним ярусам, наприклад IF/ID. Тоді поле A конвеєрного реєстра позначається як EX/MEM.A.

До апаратури другого ярусу ID належать реєстровий файл Regs, який містить множину програмно-доступних реєстрів, та поширювач знаку Sign extend, що конвертує 16-бітові безпосередні знакові константи у 32-бітові стандартні операнди формату з фіксованою комою.

Апаратура третього ярусу містить комбінаційний ALU із мультиплексорами на кожному вході і схему (Zero?) визначення істинності чи хибності умови команди умовного переходу.

Призначення інших вузлів є зрозумілим з рисунка. Можна на додачу зауважити, що реєстровий файл має два порти на читання і один на запис. Ця особливість є прямим наслідком запроваджених в комп'ютері DLX системи команд і конвеєрного принципу роботи. Реєстровий файл Regs працює у кожній команді на двох ярусах конвеєра – ID (два читання) та WB (один запис). Означені фази двох різних команд можуть збігатися у часі. Аби запобігти колізії, потрібна реалізація одночасного читання та подвійного запису до цього файла. Крім того, необхідно запобігти запису даних до того ж самого реєстра.

Роботу конвеєра можна умовно представити у вигляді часової діаграми суміщеного в часі виконання фаз команди, як це було показано раніше, або у вигляді зміщених в часі схем тракту даних комп'ютера DLX (рис. 4.10), де кожна схема зображає ту ж саму фазу команди. Тут CCi – тактові інтервали, IM – пам'ять команд, REG – реєстровий файл, ALU – арифметико логічний пристрій, DM – пам'ять даних.

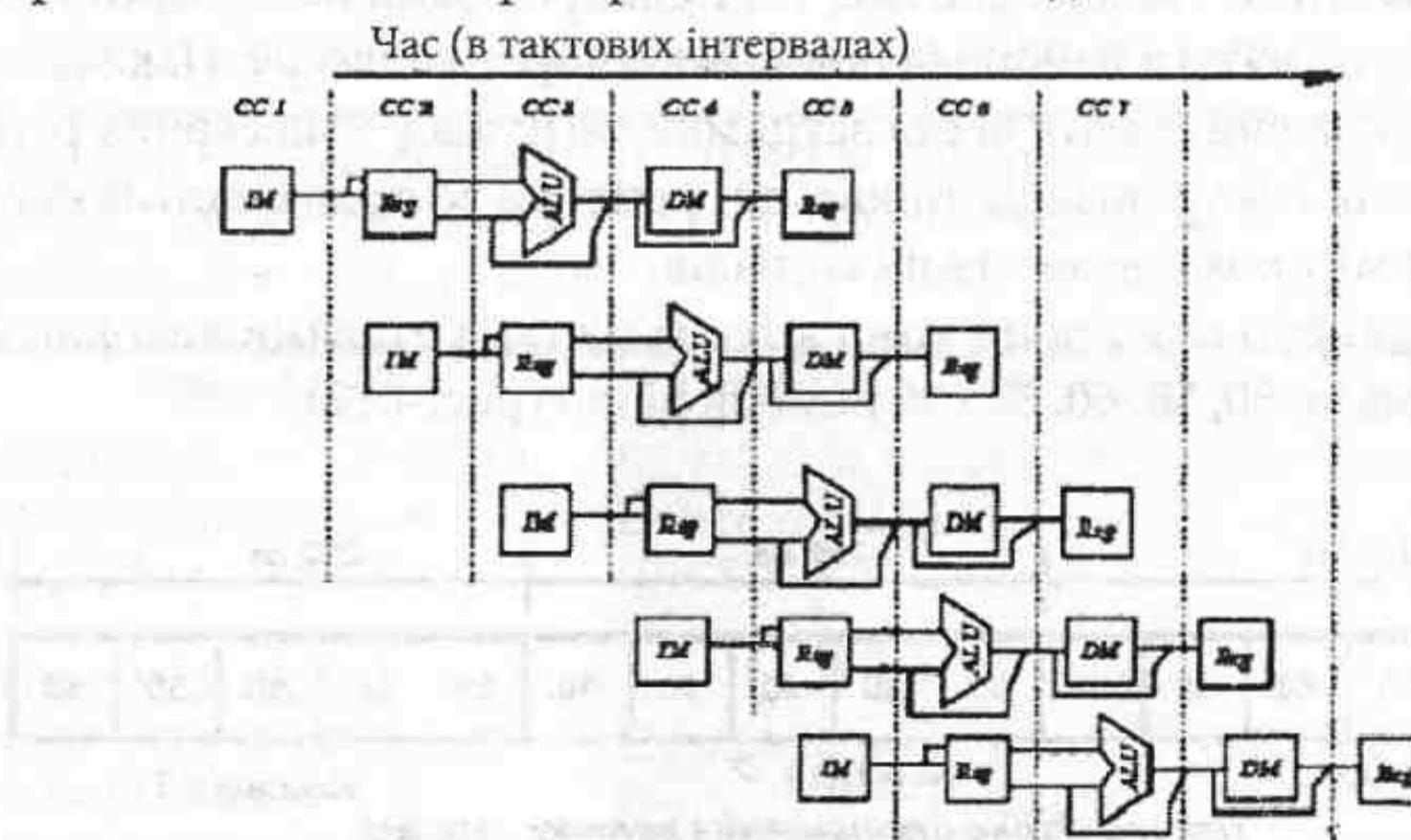


Рис. 4.10. Робота конвеєра у вигляді зміщених в часі схем тракту даних

Цей рисунок добре відображає поєднання в часі виконання різних етапів команд. Тут перша опрацьована команда знаходиться вгорі, остання – знизу, СС – тактовий інтервал. Проте частіше для представлення роботи конвеєра використовуються часові діаграми (рис. 4.11), на яких зазвичай зображаються виконувані команди, номери тактів і етапи виконання команд.

Номер команди	Номер такту								
	1	2	3	4	5	6	7	8	9
I	IF	ID	EX	MEM	WB				

$i+1$		IF	ID	EX	MEM	WB			
$i+2$			IF	ID	EX	MEM	WB		
$i+3$				IF	ID	EX	MEM	WB	
$i+4$					IF	ID	EX	MEM	WB

Рис. 4.11. Часова діаграма роботи конвеєра

З приведеної часової діаграми стає зрозуміло, для чого потрібні дві незалежні пам'яті – команд та даних. Це дозволить на тому ж самому тактовому інтервалі без колізії поєднувати виконання фаз IF та MEM двох команд. При цьому, безперечно, зростають вимоги до швидкодії запам'ятовуючих пристройів. Такі пристрої повинні відпрацьовувати звертання за один цикл (в нашому випадку за один тактовий імпульс), що обумовлює раціональність застосування відокремлених кеш пам'ятей команд та даних.

Конвеєризація збільшує пропускну спроможність процесора (кількість виконаних команд за одиницю часу), але вона не скорочує час виконання окремої команди. Насправді, вона навіть дещо збільшує час виконання кожної команди із-за затримок в конвеєрних регістрах. Проте збільшення пропускної спроможності означає, що програма виконуватиметься швидше порівняно зі скалярною (не конвеєрною) схемою.

Той факт, що час виконання кожної команди в конвеєрі не зменшується, накладає деякі обмеження на практичну довжину конвеєра. Okрім обмежень, пов'язаних із затримкою конвеєра, є також обмеження, що виникають в результаті незбалансованості затримки на кожному його ярусі та із-за накладних витрат на конвеєризацію. Частота синхронізації не може бути вищою, а, отже, тakt синхронізації не може бути меншим, ніж час, необхідний для роботи найбільш повільного яруса конвеєра. Накладні витрати на організацію конвеєра виникають через затримку сигналів у конвеєрних регістрах та із-за перекосів сигналів синхронізації. Конвеєрні регістри до тривалості такту додають час установки і затримку розповсюдження сигналів.

Як приклад розглянемо скалярний комп'ютер із п'ятьма етапами виконання операцій, які мають тривалість 50, 50, 60, 50 і 50 нс відповідно (рис. 4.12).



Рис. 4.12. Конвеєр потоку команд

Хай накладні витрати на організацію конвеєрної обробки складають 5 нс. Тоді середній час виконання команди в скалярному процесорі буде рівний 260 нс. Якщо ж використовується конвеєрна організація, тривалість такту буде рівна тривалості найповільнішої фази обробки плюс накладні витрати, тобто 65 нс. Цей час відповідає середньому часу виконання команди в конвеєрі. Таким чином, маємо чотирикратне прискорення, одержане в результаті конвеєризації.

Конвеєризація ефективна тільки тоді, коли завантаження конвеєра близьке до повного, а швидкість подачі нових команд і операндів відповідає максимальній продуктивності конвеєра. Якщо відбудеться затримка, то паралельно виконуватиметься менше операцій, і сумарна продуктивність знизиться. Такі затримки можуть відбуватися в результаті виникнення конфліктних ситуацій. Далі будуть розглянуті різні типи конфліктів, що виникають при виконанні команд в конвеєрі, і способи їх вирішення. Але спочатку розглянемо мікродії ярусів вже для варіанта конвеєрного процесора.

#### 4.2.5.2. Мікродії ярусів конвеєрного процесора

Відразу зазначимо, що всі мікродії одного яруса конвеєра мають бути сумісними в часі та виконуються паралельно в рамках однієї фази (як правило, за один тактовий інтервал). Мікродії, що реалізуються в кожному ярусі конвеєра комп'ютера DLX, зведені в табл. 4.2.

Таблиця 4.2

Ярус	Мікродії в конвеєрі DLX		
IF	IF/ID.IR $\leftarrow$ Mem[ PC ]; IF/ID.NPC, PC $\leftarrow$ ( if EX/MEM.cond { EX/MEM.NPC } else (PC+4));		
ID	ID/EX.A $\leftarrow$ Regs[IF/ID.IR <sub>6..10</sub> ]; ID/EX.B $\leftarrow$ Regs[ IF/ID.IR <sub>11..15</sub> ]; ID/EX.NPC $\leftarrow$ IF/ID.NPC; ID/EX.IR $\leftarrow$ IF/ID.IR; ID/EX.Imm $\leftarrow$ (IR <sub>16</sub> ) <sup>16</sup> ## IR <sub>16..31</sub> ;		
EX	команди АЛП EX/MEM.IR $\leftarrow$ ID/EX.IR; EX.MEM.ALUoutput $\leftarrow$ ID/EX.A op ID/EX.B; or EX/MEM.ALUoutput $\leftarrow$ ID/EX. A op ID/EX. Imm; EX/MEM. cond $\leftarrow$ 0;	команди читання / запису EX/MEM.IR $\leftarrow$ ID/EX.IR; EX/MEM.ALUoutput $\leftarrow$ ID/EX.A + ID/EX.Imm;  EX/MEM.cond $\leftarrow$ 0; EX/MEM. B $\leftarrow$ ID/EX.B;	команди переходу EX/MEM.ALUoutput $\leftarrow$ ID/EX.NPC + ID/EX.Imm;  EX/MEM.cond $\leftarrow$ (ID/EX.A op 0);
MEM	MEM/WB.IR $\leftarrow$ EX/MEM.IR; MEM/WB.ALUoutput $\leftarrow$ EX/MEM.ALUoutput;	MEM/WB.IR EX/MEM.IR; MEM/WB.LMD Mem[EX/MEM.ALUoutput]; or Mem[EX/MEM.ALUoutput] EX/MEM.B;	Дії немає
WB	Regs[MEM/WB.IR <sub>16..20</sub> ] $\leftarrow$ MEM/WB.ALUoutput; or Regs[MEM/WB.IR <sub>11..15</sub> ] $\leftarrow$ MEM/WB.ALUoutput;	Regs[MEM/WB.IR <sub>11..15</sub> ] $\leftarrow$ MEM/WB.LMD;	Дії немає

### Мікродії ярусу IF.

Перша мікродія вибирає нову команду з пам'яті команд за адресою, що зберігається в PC, і записує її до поля IR (Instruction Register) конвеєрного регістра IF/ID. В той самий час друга мікродія змінює вміст поля NPC конвеєрного регістра і програмний лічильник за алгоритмом: якщо бітове поле cond (condition – умова) попередньої команди, яка пройшла фазу EX, є одиницею (true), тоді порушується природна черговість і вміст IF/ID.NPC та PC отримує значення поля EX/MEM.NPC конвеєрного регістра EX/MEM; інакше записується наступна адреса (PC+4) з врахуванням байтової логічної структури адреси пам'яті.

### Мікродії ярусу ID.

Усі чотири мікродії є сумісними і виконуються в часі паралельно. Перша мікродія вибирає перший операнд з програмно керованого регістра регістрового файла до службового регістра A, що є інтегрованим до конвеєрного регістра ID/EX. При цьому адреса програмно керованого регістра визначається вмістом розрядів 8..10 поля IR конвеєрного регістра IF/ID. Тут вибирається операнд. Такі ж за призначенням дії виконує друга мікрооперація, але з іншим джерелом і приймачем. Третя і четверта мікродії зберігають контекст команди, що знаходиться на поточній сходинці. Це необхідно для її коректного просування конвеєром. Четверта мікродія вибирає (та знаково розширює з 16 до 32-х бітів) до службового регістра Imm (immediate – безпосередній) операнд, який містився у розрядах 16...31 поля IR конвеєрного регістра. Поточну фазу ID можна розширити у назві додатковим означенням Operand Fetch (вибирання операндів).

### Мікродії ярусу EX (команди арифметико-логічного пристрою).

Важливо відзначити, що на фазі EX вперше від початку виконання команди має бути визначеним її тип. Перша мікродія зберігає вміст регістра команди. Четверта мікродія забороняє командам ALU впливати на послідовність вибирання команд з пам'яті. Друга і третя мікродії утворюють альтернативу (або). Кожна з них визначає пару операндів для операції op і при цьому записує результат op до службового (програмно-некерованого) вихідного регістра ALU під назвою ALUoutput.

### Мікродії ярусу EX (команди load/store).

Перша мікродія зберігає контекст регістра команди, друга вираховує виконавчу (ефективну) адресу пам'яті даних на основі бази (Immediate – безпосередній операнд), третя зберігає вміст службового, програмно-некерованого регістра B, четверта забороняє поточній команді змінювати природний порядок адресування команд.

### Мікродії ярусу EX (команда branch).

Перша мікродія вираховує цільову адресу можливого переходу та зберігає її у робочому (некерованому програмістом) вихідному регістрі ALUoutput, а конкретно – у полі ALUoutput конвеєрного регістра EX/MEM. Друга мікродія вираховує істинне або хибне значення логічної умови, що визначається порівнянням в деякому, тобто op розумінні, службового регістра A, вказаного за вмістом на фазі ID, з нулем (дорівнює нулю, не дорівнює нулю, тощо). Логічне значення умови записується до поля cond конвеєрного регістра EX/MEM з метою дозволу зміни природного порядку вибирання команд програми, коли cond=1. Контексти не зберігаються, що свідчить про неформальне завершення опрацювання цієї команди в конвеєрі.

### **Мікродії ярусу МЕМ (команди арифметико логічного пристрою).**

Активних мікродій обробки інформації немає, що свідчить про транзитний характер опрацювання команди на цій сходинці. Обидві мікродії лише зберігають для подальшого користування вміст регістра команд і вихідного регістра ALU.

### **Мікродії ярусу МЕМ (команди load/store).**

Перша мікродія виконує транзитне пересилання вмісту коду операції з відповідного поля вхідного конвеєрного регістра до відповідного поля вихідного конвеєрного регістра ярусу. Це свідчить про те, що виконання команди (лише – завантаження) має продовжуватися в наступному ярусі конвеєра. При завантаженні виконується друга мікродія, а при збереженні – третя. Виконавча (ефективна) адреса пам'яті даних визначається вмістом службового вихідного регістра ALU. При завантаженні вміст комірки пам'яті даних зберігається в проміжному регістрі LMD (Load Memory Data), а при збереженні вміст службового регістра В записується до комірки пам'яті даних

Важливо, що дана мікропрограма ігнорує існування відомого парадоксу пам'яті, що коректно тільки за умови використання кеш пам'яті даних та системи переривань у випадку «невлучення до кеш» («покарання» за невлучення – це певна кількістю додаткових тактових інтервалів, аби погодити швидкодію процесора і пам'яті даних за рахунок пригальмовування операцій в скалярному процесорі).

### **Мікродії ярусу WB (команди арифметико логічного пристрою).**

Завжди виконується лише одна мікрооперація з двох зазначених. В кожному випадку результат обробки операндів в ALU з поля конвеєрного регістра MEM/WB.ALUoutput записується до регістра регистрового файла процесора. Використання двох мікрокоманд замість однієї пояснюється тим, що у форматі команд load DLX повного дотримання правила «фіксоване розташування полів» немає. За рахунок цього адреса призначення у форматі команди рухається: може визначатися розрядами 16...20 або розрядами 11...15 команди. Так чи інакше, але вказана «рухомість» адреси поля призначення ускладнює апаратний пристрій керування і може зменшити його швидкодію

### **Мікродії ярусу WB (команда load).**

Зазначимо, що команда store (збереження) на цьому ярусі виконання не потребує мікродій. Тут завершується виконання лише команди завантаження операнда з комірки пам'яті даних до регістра регистрового файла процесора

Операнд зберігається у полі LMD вхідного конвеєрного регістра MEM/WB, а адреса комірки (регістра) регистрового файла міститься у полі MEM/WB.IR 11...15. Важливо, що регистровий файл повинен реалізувати два порти, а саме, два порти на читання та один порт на запис. При цьому, якщо дві адреси на читання постачає конвеєрний регістр (IF/ID), то адресу на запис і дані постачає щойно розглянутий конвеєрний регістр (MEM/WB).

## **4.3. Суперконвеєрні процесори**

Можлива така організація виконання деякої послідовності команд в процесорі, коли всі одноіменні фази виконання цих команд послідовно, тобто спочатку проводиться вибірка всіх команд, далі їх декодування і т. д., як це показано на рис. 4.13. для послідовності із двох команд

IF1	IF2	ID1	ID2	EX1	EX2	ME1	ME2	WB1	WB2	IF1	IF2	ID1	ID2	EX1	EX2	ME1	ME2	WB1	WB2
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

Rис. 4.13. Послідовне виконання одноіменних фаз двох команд

Такий підхід не прискорює роботу процесора, але при конвеєрному опрацюванні команд може виявиться доцільним, оскільки в ярусах конвеєра (рис. 4.14) знаходяться результати виконання декількох фаз різних команд, що при наявності конфліктів дозволяє ефективніше їх вирішувати, аніж у звичайному конвеєрі команд. Процесор з конвеєром команд, в якому послідовно виконуються декілька фаз над різними командами, називається суперконвеєрним.

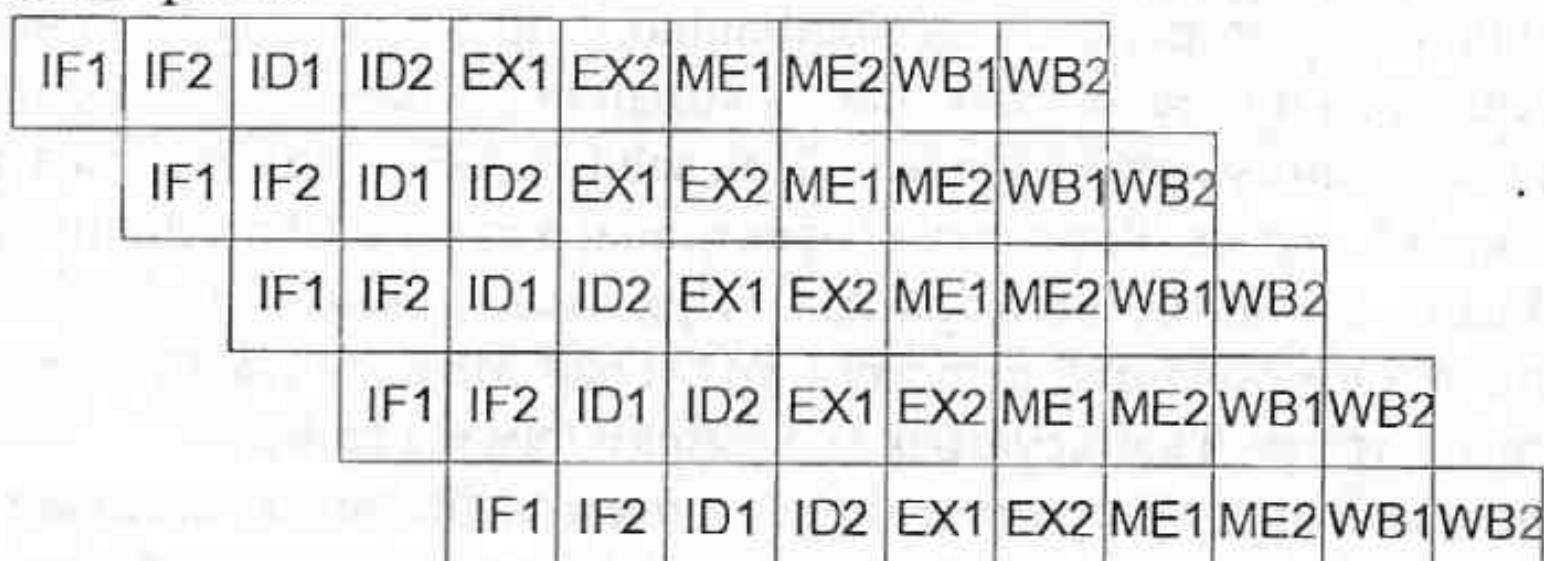


Рис. 4.14. Діаграма виконання команди в суперконвеєрному процесорі при послідовному виконанні фаз двох команд

Як видно з приведеної на рис. 4.14 діаграми, при послідовному виконанні фаз двох команд в одному такті роботи конвеєра кожна з фаз повинна виконуватись двічі. Коли послідовно виконується k фаз команд, то в кожному такті кожна з фаз має виконуватися k раз. Це говорить про те, що внутрішня частота роботи ярусів конвеєра суперконвеєрного процесора є в k разіввищою їх зовнішньої частоти, з якою відбувається обмін інформацією між ярусами.

Потрібно відзначити, що для організації суперконвеєрного опрацювання команд необхідне деяке додаткове обладнання порівняно з конвеєрним. Це, зокрема, регістри для зберігання проміжних результатів послідовно виконуваних фаз різних команд.

#### 4.4. Суперскалярні процесори

Вище була розглянута конвеєрна структура процесора, коли засоби виконання ярусів потокового графа алгоритму розділяються конвеєрними регістрами. Щоб підвищити продуктивність конвеєрного процесора потрібно далі спрощувати операції його ярусів та поглиблювати глибину конвеєра. Це і робиться в сучасних процесорах, в яких глибина конвеєра досягає двадцяти і більше ярусів. Наприклад, процесор комп'ютера UltraSPARC III має 10 ярусів конвеєра, а процесор комп'ютера Pentium IV – 20 ярусів конвеєра. Однак процес спрощення операцій ярусів конвеєра має межу, коли операції не піддаються поділу. Наприклад, фаза вибірки команди з пам'яті не може бути поділеною на простіші фази. Тоді для підвищення продуктивності процесора необхідно використовувати паралельне включення декількох конвеєрів команд. Такі процесори з декількома конвеєрами команд дозволяють одночасно виконувати кілька скалярних команд, а тому дістали назву суперскалярних.

Першу суперскалярну архітектуру розробив Джон Коук (John Cocke, IBM, 1987 рік), що отримала назву America. Він і запропонував термін “суперскаляр”. Вже потім модифікований варіант архітектури America під назвою POWER-1 (Performance Optimization With Enhanced RISC) впровадили до серійних систем RISC System/6000 фірми IBM.

Нарешті, підмножину архітектури POWER-1 реалізовано в процесорах Power PC, які є основою комп'ютерів Apple Macintosh. Іншими прикладами суперскалярних процесорів є процесори систем UltraSparc фірми Sun та Alpha фірми DEC.

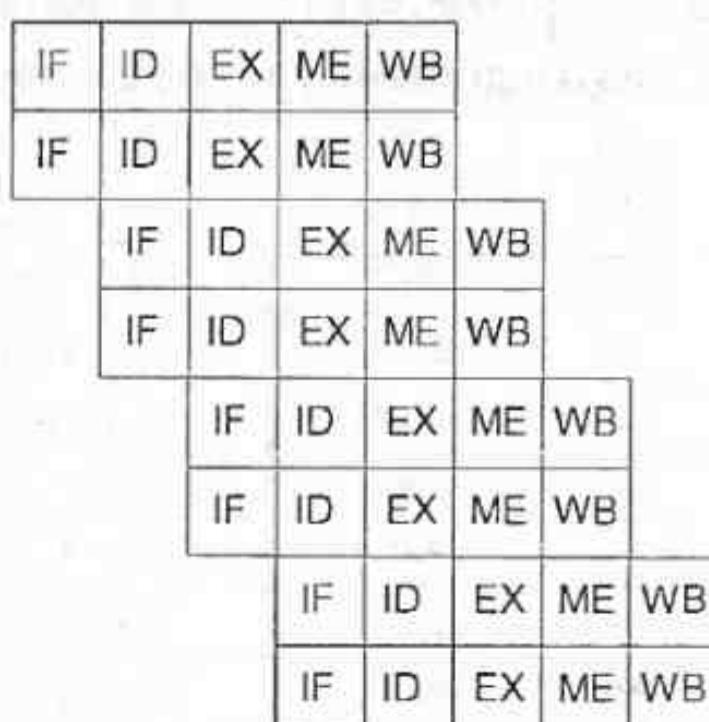
Структура суперскалярного процесора та його зв'язки з кеш пам'яттю даних і команд, показані на рис. 4.15.



Рис. 4.15. Структура суперскалярного процесора

Тут одночасно вибирається та декодується декілька команд, а блок виконання команд включає кілька функціональних блоків. Для забезпечення одночасного читання та запису кількох операндів кеш пам'ять буде за модульним принципом.

Зрозуміло, що підвищення продуктивності такого процесора досягається шляхом його конвеєризації. Діаграма виконання команд в суперскалярному процесорі, який має два конвеєри команд, показана на рис. 4.16а.



a)



b)

Рис. 4.16. Діаграма виконання команд в суперскалярному процесорі з двома конвеєрами команд, коли в одному такті виконується одна (a) та дві (b) фази команд

Можливе суміщення суперскалярного та суперконвеєрного опрацювання команд, як це показано на рис. 4.16 b.

## 4.5. Процесор векторного комп'ютера

Вище були розглянуті скалярні та суперскалярні процесори, в яких операції виконуються над скалярними даними. Однак існує значна кількість завдань, коли опрацюванню за одними процедурами підлягають великі масиви (вектори) даних. У цьому випадку виглядає доцільним розгляд можливості модифікації комп'ютера під виконання цього класу завдань. До цих пір така модифікація здійснювалась в потужних комп'ютерах, але на даний час вона почала поширюватись на всі типи комп'ютерів. Відповідно комп'ютери, орієнтовані на опрацювання векторів даних, дістали назву векторних.

Різницю між виконанням скалярної та векторної операції наглядно відображає рис. 4.17, з якого видно, що скалярна операція передбачає виконання додавання над двома даними, тоді як векторна – над двома векторами даних.

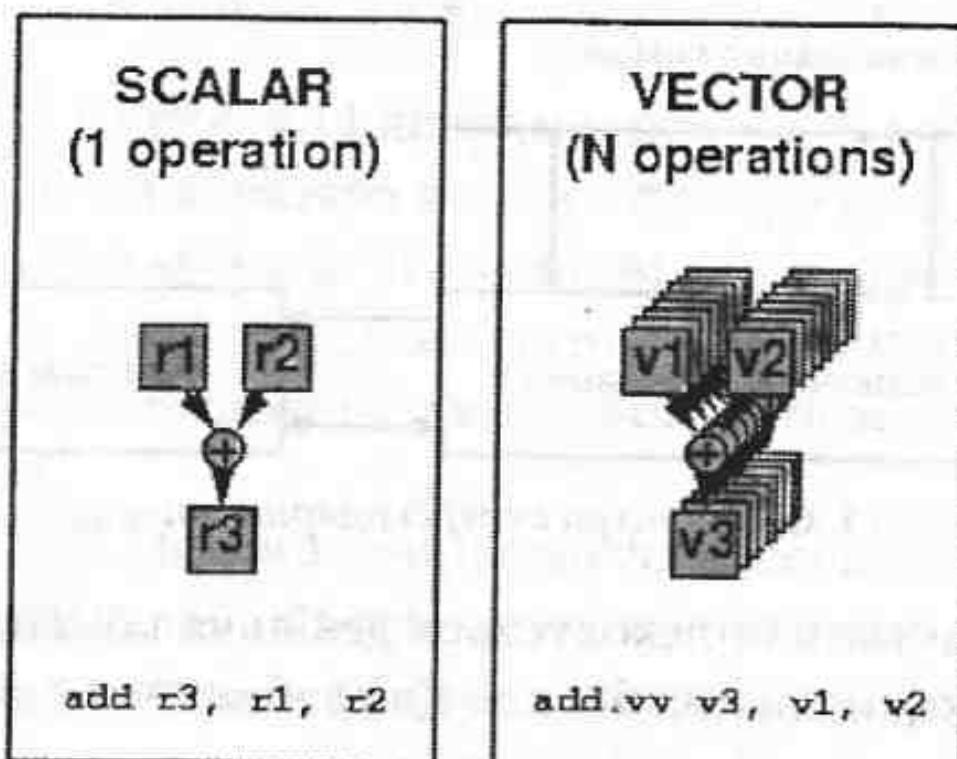


Рис. 4.17. Виконання скалярної та векторної операцій додавання

Аби зрозуміти стиль програмування векторних комп'ютерів, наведемо приклад програми із скалярними і векторними кодами. Запишемо програму обчислення виразу  $Y = a * X + Y$ , де  $Y, X$  – вектори, а  $a$  – скаляр. Нехай вектори мають довжину по 64 елементи. Векторна програма має вигляд:

LD	F0, a	; load scalar a
LV	V1, Rx	; load vector X
MULTS	V2, F0, V1	; vector-scalar mult.
LV	V3, Ry	; load vector Y
ADDV	V4, V2, V3	; add
SV	Ry, V4	; store vector

Відповідна скалярна програма має вигляд:

Loop:	LD	F0,a	
	ADDI	R4,Rx,#512 ; last address to load	
	LD	F2,0(Rx) ; load X(l)	
	MULTD	F2,F0,F2 ; a*X(l)	
	LD	F4, 0(Ry) ; load Y(l)	
	ADDD	F4,F2,F4 ; a*X(l)+Y(l)	
	SD	F4,0(Ry) ; store into Y(l)	
	ADDI	Rx,Rx,#8 ; increment index	
	ADDI	Ry,Ry,#8 ; increment index	
	SUB	R20,R4,Rx ; compute bound	
	BNZ	R20, loop ; check if done	

У скалярній програмі курсивом позначено залежності, яких немає у векторному варіанті програми. Обидва варіанти програми можна порівняти за наступними кількісними характеристиками:

- За кількістю операцій: 578( $2+9*64$ ) проти 321( $1+5*64$ ); кількість операцій у векторній програмі зменшено в 1,8 разу.
- За кількістю команд: 578( $2+9*64$ ) проти 6-ти команд у векторній програмі; перевага в 96 разів.

В таблиці 4.3 наведені характеристики кількох промислових векторних комп'ютерів, з якої видно доцільність їх створення з огляду на досягнуту продуктивність.

Таблиця 4.3

Тип машини	Рік випуску	Частота, MHz	Кількість реєстрів	Кількість елементів	Кількість пристрій float point	Кількість пристрій load/store	Продуктивність (MFLOPS)
Cray-1	1976	80	8	64	6	1	160
Cray XMP	1983	120	8	64	8	2L, 1S	940
Cray YMP	1988	166	8	64	8	2L, 1S	2667
Cray C-90	1991	240	8	128	8	4	15238(16)
Cray T-90	1996	455	8	128	8	4	57600(32)
Conv. C-1	1984	10	8	128	4	1	20(1)
Conv. C-4	1994	133	16	128	3	1	3240(4)
Fuj. VP200	1982	133	8-256	32-1024	3	2	533(1)
Fuj. VP300	1996	100	8-256	32-1024	3	2	N/A
NEC SX/2	1984	160	8 + 8K	256 + var	16	8	1300(1)
NEC SX/3	1995	400	8 + 8K	256 + var	16	8	25600(4)

Таким чином, процесори векторних комп'ютерів виконують команди над векторами даних. Структура цих процесорів за складом та зв'язками повторює вже розглянуті вище структури процесорів, тобто це можуть бути процесори векторних комп'ютерів із

складною та простою системою команд, конвеєрні та суперконвеєрні, а також процесори супервекторних комп'ютерів, коли в процесорі є декілька конвеєрів команд. Основна їх відмінність – забезпечення одночасного виконання однієї команди над вектором даних. Це, зокрема, дозволяє будувати їх блоки виконання команд за конвеєрним принципом і при цьому позбутися конфліктів, які суттєво гальмують роботу конвеєра чи ускладнюють його структуру.

Для вияснення базових принципів побудови процесорів векторних комп'ютерів розглянемо структуру та систему команд процесора векторного варіанта комп'ютера DLX, а саме комп'ютера DLXV. До складу процесора, структура якого приведена на рис. 4.18, входять пристрій векторного читання запису, регістрові файли з векторними та скалярними регістрами, а також операційний пристрій з набором конвеєрних операційних пристроїв додавання, множення та ділення з рухомою комою та виконання арифметичних і логічних операцій над цілими числами.

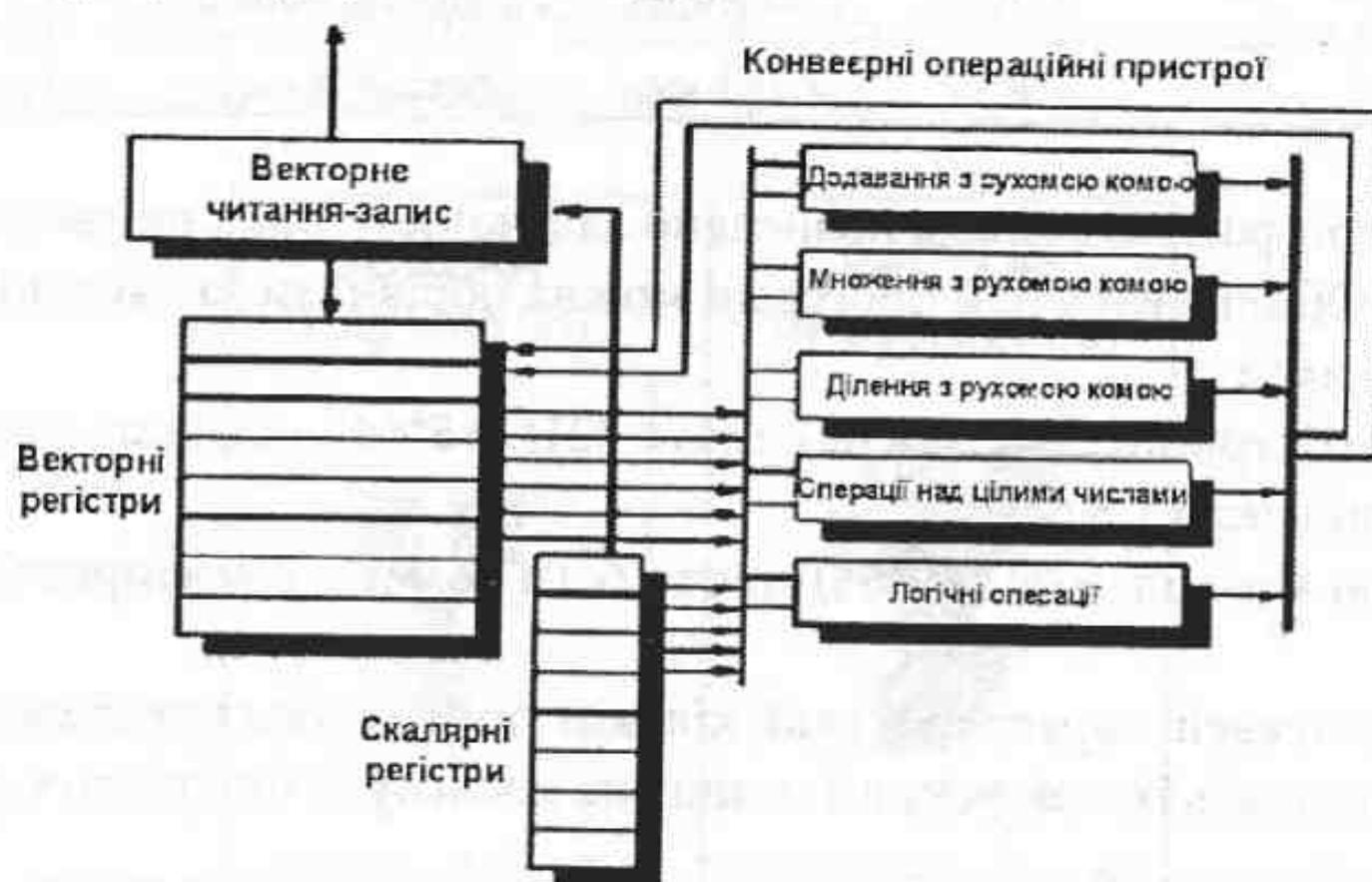


Рис. 4.18. Структура процесора комп'ютера DLXV

Цей комп'ютер має векторні команди, які наведено в табл. 4.4.

Таблиця 4.4

Команда	Операнди	Операція	Коментар
ADDV	V1, V2, V3	V1=V2+V3	VECTOR+VECTOR
ADDSV	V1, F0, V2	V1=F0+V2	SCALAR+VECTOR
MULTV	V1, V2, V3	V1=V2xV3	VECTORxVECTOR
MULSV	V1, F0, V2	V1=F0xV2	SCALAR x VECTOR
LV	V1, R1	V1=M[R1..R1+63]	LOAD, STRIDE=1
LVWS	V1, R1, R2	V1=[R1..R1+63*R2]	LOAD, STRIDE=R2
LVI	V1, R1, V2	V1=[R1+V2i, i=0..63]	indirect ("gather")
CeqV	VM, V1, V2	VMASKi=(V1 I=V2I)	comp. Set mask
MOV	VLR, R1	Vec. Len. Reg = R1	set vector length
MOV	VM, R1	Vec. Mask=R1	set vector mask

Приведений процесор є процесором комп'ютера з простою системою команд. До цього типу належать усі векторні суперком'ютери: Cray, Convex, Fujitsu, Hitachi, NEC.

Хоча потрібно зауважити, що існують і векторні комп'ютери з архітектурою «пам'ять-пам'ять», коли всі векторні операції є операціями типу пам'ять-пам'ять наприклад, CDC-6600.

Подібно до приведеного на рис. 4.18, процесори векторних комп'ютерів містять наступні основні компоненти

- Векторні регістри; це регістрий файл фіксованої ємності що вміщує вектор даних. Цей файл має як мінімум 2 порти на читання і один порт на запис та зазвичай включає 8–32 векторних регістри, кожний з яких є 64–128-розрядним
- Конвеєрні операційні пристрої. Зазвичай застосовують 4–8 операційних пристріїв, а саме: додавання, множення і ділення з фіксованою та рухомою комою, зсуви тощо
- Векторний вузол читання-запису, також конвеєрний, який опрацьовує вектори даних. Водночас застосовують декілька таких вузлів
- Скалярні регістри, які містять один скаляр з рухомою комою або адресу
- Багатошинні магістралі або комутаційні мережі, які з'єднують між собою всі зазначені компоненти, аби прискорити роботу процесора в цілому

Перші векторні комп'ютери STAR-100 фірми CDC та ASC фірми TI були створені в 1972 році. Це були векторні комп'ютери з архітектурою типу пам'ять-пам'ять

Значний внесок в теорію побудови векторних комп'ютерів зробив видатний американський конструктор векторних суперком'ютерів Сеймур Крей. На рис. 4.19 зображені його перший векторний суперком'ютер Cray-1 та векторні суперком'ютери Cray-2 і Cray-YMP.

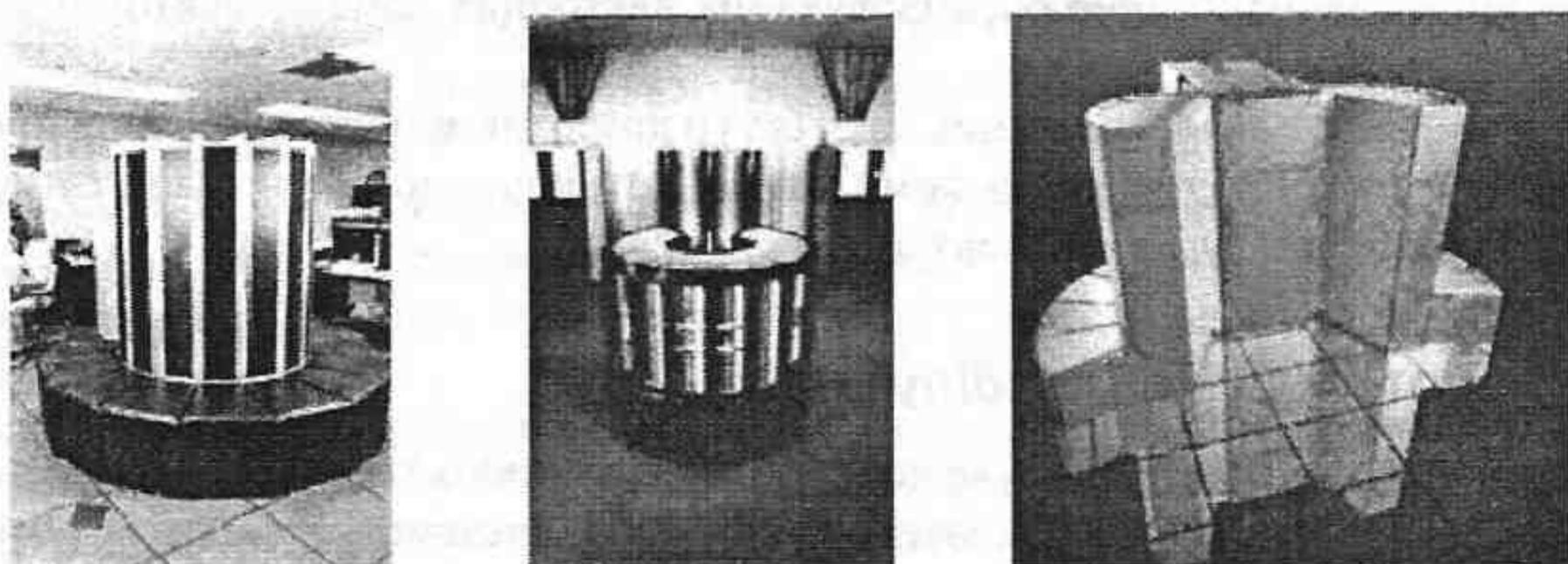


Рис. 4.19. Векторні суперком'ютери Cray-1, Cray-2 і Cray-YMP

Комп'ютер CRAY-1 був створений в 1976 році і мав архітектуру типу «регистр-регистр», тому він був найшвидшим серед векторних та скалярних комп'ютерів свого часу. В 1981 році на ринку з'явився значно потужніший векторний комп'ютер CYBER-205 фірми CDC з тією ж базовою архітектурою, що і STAR-100, але більшою кількістю векторних функціональних блоків. Це були векторні комп'ютери з архітектурою типу “пам'ять-пам'ять”. В 1983 році фірма Cray Research поставила на ринок векторний суперком'ютер CRAY X-MP, а через кілька років – CRAY-2, який мав вищу тактову частоту та іще більший рівень конвеєризації. В 1988 році фірма Cray Research створила значно швидший, ніж X-MP суперком'ютер CRAY Y-MP, котрий мав 8 конвеєрних процесорів, кожний з яких працював з тактом 6 нс.

Одночасно потужні векторні суперком'ютери почали створюватись і в інших державах. Зокрема, в середині 80-х років в Японії були створені суперком'ютери Fujitsu

VP100 і VP200, за ними Hitachi S810 і NEC SX/2, які за технічними характеристиками не поступалися комп'ютерам фірми Cray Research.

Протягом наступних 20 років векторні комп'ютери мали швидкий розвиток і з екзотичних перетворились в широковживаний клас потужних комп'ютерів.

## **4.6. Класифікація архітектури комп'ютера за рівнем суміщення опрацювання команд та даних**

Виходячи з вищезгаданого розгляду різних принципів побудови процесорів, можна зробити наступну класифікацію архітектури комп'ютера за рівнем суміщення в них опрацювання команд та даних:

- за відсутністю та наявністю конвеєра команд: комп'ютери без конвеєра команд та комп'ютери з конвеєром команд
- за відсутністю та наявністю конвеєра даних: комп'ютери без конвеєра даних та комп'ютери з конвеєром даних
- за кількістю послідовно виконуваних фаз команд в конвеєрі: конвеєрні та суперконвеєрні
- за кількістю одночасно опрацьовуваних даних за однією командою: скалярні та векторні
- за кількістю одночасно опрацьовуваних скалярних команд: скалярні та суперскалярні
- за кількістю одночасно опрацьовуваних векторних команд: векторні та супервекторні

Проведений вище аналіз названих архітектур дозволяє зробити висновок про те, що для побудови високопродуктивних комп'ютерів потрібно, щоб вони мали суперконвеєрну суперскалярну та супервекторну архітектуру.

## **4.7. Короткий зміст розділу**

Розглянуто місце процесора в комп'ютері, його функції та склад. Приведена однoshинна структура процесора комп'ютера із складною системою команд та розглянуто виконання на ній основних операцій процесора: вибірки слова із пам'яті, запам'ятовування слова в пам'яті, обміну між реєстраторами, виконання арифметичних і логічних операцій. Проведено порівняння одношинної з багатошинною структурою процесора.

Виділено особливості побудови процесора комп'ютера із складною системою команд та сформовано вимоги до процесора комп'ютера з простою системою команд. Описані базові принципи побудови процесора комп'ютера з простою системою команд, а також взаємодія процесора з основною пам'яттю в комп'ютері з простою системою команд.

Розглянуто фази виконання команд в процесорі комп'ютера з простою системою команд: вибирання та декодування команди, виконання та формування ефективної адреси, звернення до пам'яті/завершення умовного переходу, зворотного запису. Описана конвеєрна структура процесора з простою системою команд на прикладі процесора комп'ютера DLX. Розглянуто принципи побудови суперконвеєрним та суперскалярних процесорів, а також процесорів векторних комп'ютерів. Виходячи з вищезгаданого розгляду різних принципів побудови процесорів, зроблено класифікацію архітектури комп'ютера за рівнем суміщення в ньому опрацювання команд та даних.

## 4.8. Література для подальшого читання

Питання побудови процесорів комп’ютерів з складною системою команд детально розглянуті в багатьох підручниках та наукових працях, зокрема [1–4, 7, 8]. Вимоги до процесора комп’ютера з простою системою команд та базові принципи його побудови детально описані в роботах [5, 6]. Там же, а також в роботах [7–8] запропоновано принципи побудови процесора комп’ютера DLX і процесора векторних комп’ютерів, а в роботах [9–10] розглянуті принципи побудови суперконвеєрним та суперскалярних процесорів.

## 4.9. Література до розділу 4

1. Каган Б.М. Электронные вычислительные машины и системы. – М.: Энергия, 1979. – 528 с.
2. Каган Б. М., Каневский М. М. Цифровые вычислительные машины и системы. – М.: Энергия, 1974. – 680 с.
3. Tanenbaum, A. *Structured Computer Organization*, 4<sup>th</sup> ed. Upper Saddle River, NJ: Prentice Hall, 1999.
4. Stallings, W. *Computer Organization and Architecture*, 5th ed., New York, NY: Macmillan Publishing Company, 2000.
5. D. Patterson, J. Hennessy. Computer Architecture. A Quantitative Approach. Morgan Kaufmann Publishers, Inc. 1996.
6. Patterson, D. A., & Hennessy, J. L. *Computer Organization and Design, The Hardware/Software Interface*, 2nd ed., San Mateo, CA: Morgan Kaufmann, 1997.
7. AGERWALA, T. AND J. COCKE [1987]. “High performance reduced instruction set processors”, IBM Tech. Rep. (March).
8. Bakoglu, H. B., G. F. Grohoski, L. E. Thatcher, J. A. Kahle, C R. Moore, D. P. Tuttle, W. E. Maule, W. R. Hardell, D. A. Hicks, M. Nguyen phu, R. K. Montoye, W. T. Glover, and S. Dhawan [1989]. «IBM second-generation RISC processor organization,» Proc. Int'l Conf. on Computer Design, IEEE (October), Rye, N.Y., 138–142.
9. Johnson, M. [1990]. Superscalar Microprocessor Design, Prentice Hall, Englewood Cliffs, N.J.
10. JOUPPI, N. P. AND D. W. WALL [1989]. “Available instruction-level parallelism for superscalar and superpipelined processors”, Proc. Third Conf. on Architectural Support for Programming Languages and Operating Systems, IEEE/ACM (April), Boston, 272–282.

## 4.10. Питання до розділу 4

1. Місце процесора в комп’ютері та його функції.
2. Що таке командний цикл?
3. Дві основні фази командного циклу.
4. Основні вузли процесора.
5. Одношинна структура процесора комп’ютера із складною системою команд і його зв’язки з іншими пристроями комп’ютера.
6. Виконання процесором операції “Вибірка слова з пам’яті”.
7. Виконання процесором операції “Запам’ятування слова в пам’яті”.
8. Виконання процесором операції обміну між регістрами.
9. Виконання процесором арифметичних і логічних операцій.
10. Порівняння одношинної та багатошинної структур процесора комп’ютера із складною системою команд.

11. Чому в процесорі комп'ютера із складною системою команд команда виконується за багато тактів?
12. Чому в процесорі комп'ютера із складною системою команд потрібна складна система розпізнавання команди?
13. Чому в процесорі комп'ютера із складною системою команд організація конвеєризації виконання команд складніша, ніж у процесорі комп'ютера з простою системою команд?
14. Основні вимоги до процесора комп'ютера з простою системою команд.
15. Сформуйте правила вибору системи команд комп'ютера з простою системою команд.
16. Чому в системі команд комп'ютера з простою системою команд відносно небагато операцій та способів адресації?
17. Чому в комп'ютері з простою системою команд команди обробки даних мають реалізовуватися лише у формі “регистр-регистр”?
18. Чому в комп'ютері з простою системою команд обміни з основною пам'яттю виконуються лише за допомогою команд завантаження/запису?
19. Чому в процесорі комп'ютера з простою системою команд дешифрування команд із спрощеними форматами має виконуватися лише апаратно?
20. Що є основою проектування структури процесора комп'ютера з простою системою команд?
21. Як будується процесор для того, щоб команда виконувалася за один такт?
22. Поясніть принципи роботи процесора комп'ютера DLX.
23. Опишіть фази виконання команди в процесорі комп'ютера DLX.
24. Поясніть роботу конвеєрного процесора комп'ютера DLX.
25. Проаналізуйте та поясніть мікродії, що виконуються на сходинці IF конвеєра комп'ютера DLX.
26. Проаналізуйте та поясніть мікродії, що виконуються на сходинці ID конвеєра комп'ютера DLX.
27. Проаналізуйте та поясніть мікродії, що виконуються на сходинці EX конвеєра комп'ютера DLX при виконанні команди ALP.
28. Проаналізуйте та поясніть мікродії, що виконуються на сходинці EX конвеєра комп'ютера DLX при виконанні команд завантаження і збереження (load/store).
29. Проаналізуйте та поясніть мікродії, що виконуються на сходинці EX конвеєра комп'ютера DLX при виконанні команди умовного переходу (branch).
30. Проаналізуйте та поясніть мікродії, що виконуються на сходинці MEM конвеєра комп'ютера DLX при виконанні команд ALP.
31. Проаналізуйте та поясніть мікродії, що виконуються на сходинці MEM конвеєра комп'ютера DLX при виконанні команд завантаження або збереження.
32. Проаналізуйте та поясніть мікродії, що виконуються на сходинці WB конвеєра комп'ютера DLX при виконанні команд ALP.
33. Проаналізуйте та поясніть мікродії, що виконуються на сходинці WB конвеєра комп'ютера DLX при виконанні команди load.
34. Основна ідея суперконвеєрних процесорів.
35. Суперскалярні процесори – структура та принцип роботи.
36. Процесори векторних комп'ютерів – структура та принцип роботи.
37. Наведіть класифікацію архітектури комп'ютера за рівнем суміщення в ньому опрацювання команд та даних.