2. АПАРАТНО-ПРОГРАМНА АРХІТЕКТУРА IA-32 ПРОЦЕСОРІВ INTEL

2.1. АРХІТЕКТУРА ЕЛЕКТРОННО-ОБЧИСЛЮВАЛЬНОЇ МАШИНИ

На сучасному комп'ютерному ринку спостерігається велика різноманітність різних типів комп'ютерів. Тому можливо припустити виникнення в споживача питання – як оцінити можливості конкретного типу (чи моделі) комп'ютера та його відмітні особливості від комп'ютерів інших типів (моделей). Вивчення для цього тільки структурної схеми комп'ютера недостатньо, оскільки вона принципово мало чим відрізняється в різних машинах: у всіх комп'ютерах є оперативна пам'ять (ОП), процесор, зовнішні пристрої.

Різними є способи, засоби й використовувані ресурси, за допомогою яких комп'ютер функціонує як єдиний механізм. Щоб зібрати воєдино усі поняття, що характеризують комп'ютер у розумінні його функціональних програмно-керованих властивостей, існує спеціальний термін – "архітектура ЕОМ".

Однозначно визначити поняття архітектури ЕОМ достатньо важко, оскільки за бажанням до нього можна приєднати все, що пов'язане з комп'ютером загалом.

Архітектура ЕОМ – це абстрактне представлення ЕОМ, яке відбиває його структурну, схемотехнічну та логічну організацію. Поняття архітектури є комплексним і до свого складу включає:

- структурну схему ЕОМ;
- засоби та способи доступу до елементів структурної схеми ЕОМ;
- організацію та розрядність інтерфейсів ЕОМ;
- набір та доступність регістрів процесора ЕОМ;
- організацію та способи адресації пам'яті;
- способи представлення та формати даних ЕОМ;
- набір машинних команд;
- формати машинних команд;
- правила обробки переривань (нештатних ситуацій).

Отже, опис архітектури включає майже всю необхідну для програміста інформацію про комп'ютер.

На сьогодні архітектурні властивості більшості сучасних комп'ютерів підпадають під поняття *фон-неймановської архітектури*, за іменем вченого фон Неймана.

Коли фон Нейман почав займатися комп'ютерами, програмування останніх здійснювалось способом комутації. У перших ЕОМ для генерації потрібних сигналів необхідно було за допомогою перемикачів виконати ручне програмування всіх логічних схем. У перших машинах використовували десяткову логіку, за якою кожен розряд був представлений десятковою цифрою і моделювався 10 електронними лампами. Залежно від потрібної цифри одна лампа вмикалась, решта залишались вимкненими. Фон Нейман запропонував схему ЕОМ з програмою в пам'яті та двійковою логікою замість десяткової. Логічно до складу машини фон Неймана входило п'ять блоків: оперативна пам'ять, арифметико-логічний пристрій (ALU) з акумулятором, блок управління, пристрої введення та виведення.

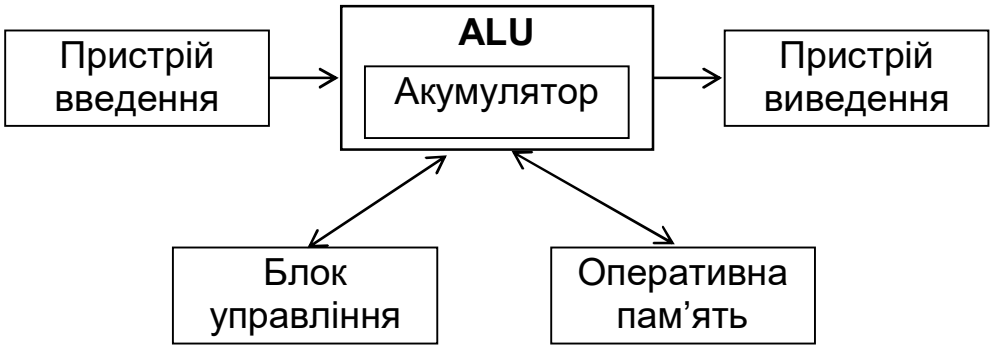

Рис. 2.1. Функціональна схема машини фон Неймана

Нижче викладено властивості та принципи роботи машини фон Неймана.

1. *Принцип програми, що зберігається*. Згідно з цим принципом код програми та її дані знаходяться в одному адресному просторі оперативної пам'яті.

2. *Лінійний простір пам'яті*. Для оперативного зберігання інформації комп'ютер має сукупність комірок з послідовною нумерацією (адресами) 0, 1, 2,… Така сукупність комірок називається *оперативною пам'яттю*.

3. *Принцип мікропрограмування*. Машинна мова не є тією кінцевою субстанцією, яка фізично приводить у дію процеси в машині. До складу процесора входить блок мікропрограмного управління. Цей блок для кожної машинної команди має набір дій-сигналів, які потрібно згенерувати для фізичного виконання необхідної машинної команди.

4. *Послідовне виконання команд*. Процесор вибирає з пам'яті команди виключно послідовно. Для зміни прямолінійного виконання програм необхідно використовувати спеціальні команди (команди умовного та безумовного переходу).

5. *Відсутність різниці між даними та командами в пам'яті*. З погляду процесора принципової різниці між даними та командами немає. Дані та команди знаходяться в одному просторі пам'яті у вигляді послідовності нулів та одиниць. Тому в програмі важливо завжди чітко розділяти простір даних і команд.

6. *Байдужість до цільового призначення даних*. Для машини немає значення, яке логічне навантаження несуть дані, які обробляються.

2.2. ОСНОВИ ФУНКЦІОНУВАННЯ СУЧАСНИХ ПРОЦЕСОРІВ

Робота всіх пристроїв комп'ютерної системи, включаючи процесор, синхронізуються імпульсами тактової частоти, яка задається електронними схемами процесора або материнської плати. Схематично це можна представити так, як показано на рис.2.2.

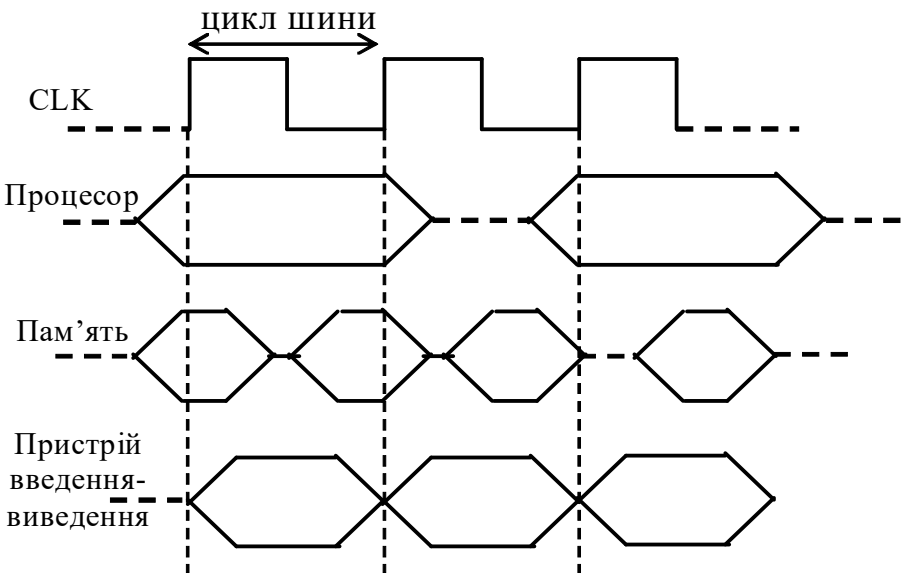

Рис. 2.2. Синхронізація комп'ютерної системи

Послідовність синхронізуючих імпульсів на рис. 2.2 позначено як CLK. Як правило, зміна стану пристроїв системи здійснюється за фронтом імпульсів CLK, хоча електронні схеми самих пристроїв використовують у процесі функціонування також і внутрішню синхронізацію. Джерелом тактових імпульсів у сучасних системах є окремий пристрій.

Усі часові залежності в комп'ютерній системі вимірюються в циклах шини. *Цикл шини* – це період повторення синхроімпульсів (рис. 2.2).

Другою особливістю роботи комп'ютера є те, що різні пристрої системи мають різну апаратну швидкодію. Найбільш швидкодіючим пристроєм комп'ютера є процесор, більш повільними є пам'ять та пристрої введення-виведення. Під апаратною швидкодією розуміють максимальну частоту синхронізації, з якою може працювати пристрій. Наприклад, за тактовою частотою шини 200 *МГц* процесор *Intel Pentium 4*, який працює з внутрішньою тактовою частотою 2,4 *ГГц*, повинен помножити зовнішню частоту на певний коефіцієнт, у цьому випадку на 12.

Різниця в швидкостях обробки даних процесора й зовнішніми відносно нього пристроями могло б призвести до затримки обробки даних. Щоб запобігти цьому, використовують ряд підходів.

Перший підхід полягає в тому, щоб протягом одного циклу шини передавати якомога більший об'єм даних. Наприклад, процесори *AMD* можуть подвоювати кількість даних, що передаються (*DDR*, *Dual Data Rate*), а процесори *Intel Pentium* передавати почетверений об'єм (*QDR – Quad Data Rate*) даних протягом циклу шини (рис. 2.3).

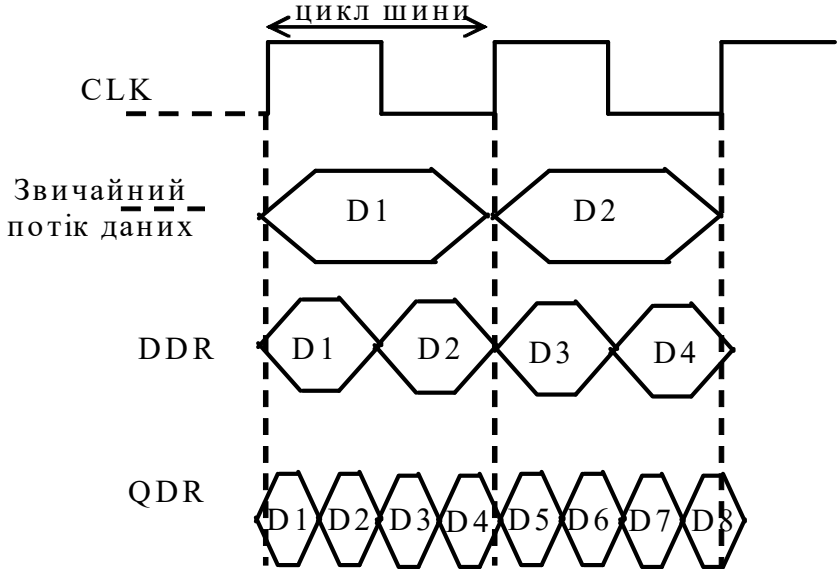

Рис. 2.3. Обмін даними з пам'яттю для різних типів процесорів

При другому підході дані кешуються процесором. Кеш являє собою швидку оперативну пам'ять відносно невеликого об'єму, яка розташовується, як правило, на кристалі процесора й призначена для тимчасового зберігання команд та даних при обміні з відносно повільною оперативною пам'яттю.

Усі сучасні процесори, включаючи *Intel Pentium 4* мають у своєму складі кеші команд/даних, які дозволяють здійснювати попередній вибір інформації з ОП, знижуючи тим самим час очікування процесора. Як правило, на кристалі процесора розміщуються три кеші (рис. 2.4).

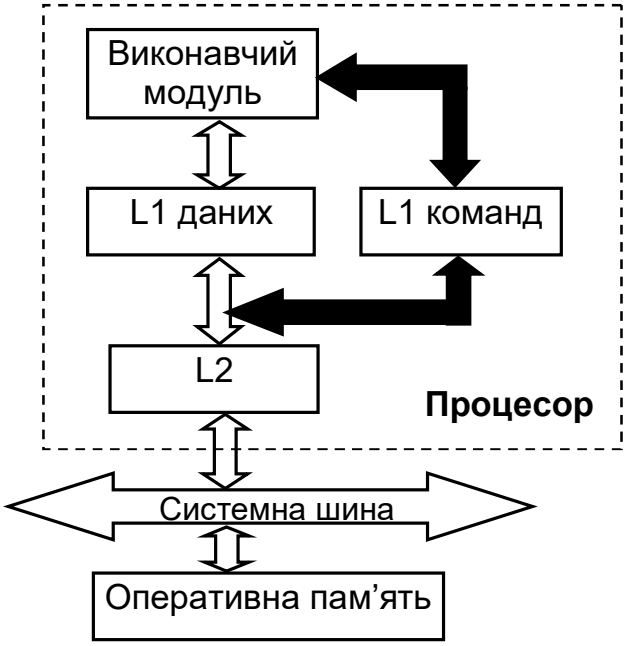

Рис. 2.4. Кешування пам'яті в процесорах

Кеші команд/даних реалізуються на кристалі процесора у вигляді швидкодіючої статичної пам'яті. Кеш заведено позначати символом L, за яким вказують рівень кеша (1, 2, 3). Рівень кеша характеризує його фізичне віддалення від виконавчих модулів процесора та в більшості випадків його швидкодію. Наприклад, кеш команд 1-го рівня (L1) розміщується безпосередньо біля пристроїв вибору та декодування команд процесора. Цьому кешу властива найвища швидкодія, оскільки з нього здійснюється вибір команд на внутрішній частоті процесора.

У кеші даних L1 розміщуються дані, які обробляються командами. Його швидкодія схожа зі швидкодією кеша L1 команд.

Кеш загального призначення L2 розміщується біля інтерфейсу системної шини й призначений для обміну даними з оперативною пам'яттю комп'ютера. Швидкодія кеша L2 набагато вища за швидкодію оперативної пам'яті, яка має динамічний тип, але менша ніж у кешах L1. Однак, об'єм кеша L2 набагато більший за об'єм кеша L1.

Виконавчий модуль процесора (рис. 2.4) – це умовна назва цілої групи взаємозв'язаних між собою пристроїв, які забезпечують повний цикл виконання команди. До складу цієї групи входять:

- пристрої вибору та декодування інструкцій;
- декілька ALU, серед них модулі обробки чисел із рухомою комою (FPU);
- планувальник черги мікроінструкцій;
- диспетчер мікроінструкцій та інші пристрої.

На рис. 2.5 наведено функціональну схему гіпотетичного процесора, у яку включені основні вузли, що є в складі всіх сучасних процесорів.

Рис. 2.5. Послідовність виконання команд процесора

Послідовність виконання команд процесора. При виборі команд/даних з ОП всі вони розміщуються в кеші L2 загального призначення (для архітектури IA-32 кеш L2 може мати розмір 256 *Кбайт* та більше). Потім дані в кеші L2 розділяються: інструкції процесора потрапляють у кеш команд L1, а дані – у кеш даних L1. Далі починається процес виконання команд, який здійснюється за типовим для всіх сучасних процесорів алгоритмом:

1) здійснюється звернення до кеша L1 команд (інструкцій) для вибору наступної команди;

2) якщо інструкція знаходиться в кеші, то виконується звичайна послідовність дій: вибірка, декодування інструкції та її виконання. Якщо інструкція в кеші відсутня, здійснюється звертання до кеша L2;

3) якщо інструкція знайдена, виконується завантаження чергової групи команд в кеш команд L1, після чого дана команда вибирається з кеша L1 і починається цикл її виконання. Якщо при зверненні до кеша L2 інструкції там не виявлено, то процесор ініціює звернення до основної пам'яті для завантаження чергової сторінки;

4) команда, що вибрана з кеша L1, декодується, внаслідок чого формується послідовність мікроінструкцій, або, інакше, мікрооперацій. Мікрооперації являють собою спрощені команди, які виконують відносно прості дії. Процесори платформи IA-32 оперують набором команд CISC (*Complete-Instruction-Set Computing*). При виконанні операцій у самому процесорі вони перетворюються у формат RISC (*Reduced-Instruction-Set Computing*). Власне, мікрооперації і є такими RISC-інструкціями. Як правило, мікроінструкції вибираються із постійного запам'ятовуючого пристрою (ROM), у якому міститься мікрокод для всіх інструкцій процесора.

5) послідовність мікроінструкцій надходить в одну з черг мікроінструкцій, що формуються планувальником мікрооперацій відповідно до типу мікрооперації. Для гіпотетичного процесора, представленого на рис. 2.5, будуть формуватися черги для ALU та математичного сопроцесора (FPU). До виконання операцій планувальником послідовність операцій відповідає послідовності команд у програмі. Після планувальника мікрооперацій порядок виконання мікроінструкцій може змінитися через те, що ряд послідовних інструкцій процесора зможуть виконуватися паралельно оскільки процесор містить декілька модулів ALU та FPU.

Протягом виконання мікрооперацій можуть встановлюватися різні прапорці, за допомогою яких програма може виконувати розгалуження. Кожен процесор має свій алгоритм обробки переходів та розгалужень. При виникненні таких ситуацій, як правило, змінюється вміст кешем і виконання послідовності операцій починається з нової адреси.

2.3. МЕХАНІЗМИ ПОКРАЩЕННЯ ЕФЕКТИВНОСТІ РОБОТИ ПРОЦЕСОРІВ

У ранніх процесорах кожна наступна інструкція починала виконуватись тільки після завершення попередньої. На сьогодні інструкції можуть виконуватись одночасно завдяки деяким технологіям побудови процесорів.

Конвеєр команд (***pipeline***). Ідея конвеєрної обробки полягає в тому, що виконання однієї інструкції процесора розбивається на

декілька кроків або стадій. Кількість цих кроків і операцій, які виконуються на них, визначаються архітектурою процесора. Наприклад, в *Intel Pentium 4* конвеєр команд включає 20 кроків. На кожній стадії виконується будь-яка операція з обробки команди: вибір, декодування, формування мікрооперацій, поставлення мікрооперацій до черги тощо, при цьому для виконання окремої операції може знадобитися декілька кроків. Кожен крок конвеєра передбачає обробку інструкції окремим пристроєм: пристроєм вибору команд, декодером команд, диспетчером команд тощо.

Використання конвеєра команд дозволяє зменшити затримки між виконанням інструкцій. Конвеєр команд не дозволяє здійснювати паралельне виконання команд через те, що різні команди перебувають на різних стадіях виконання. Однак, за його допомогою значно зменшується час простоювання процесора. Конвеєр команд процесора може виглядати так, як показано на рис. 2.6 (насправді конвеєр виглядає набагато складніше).

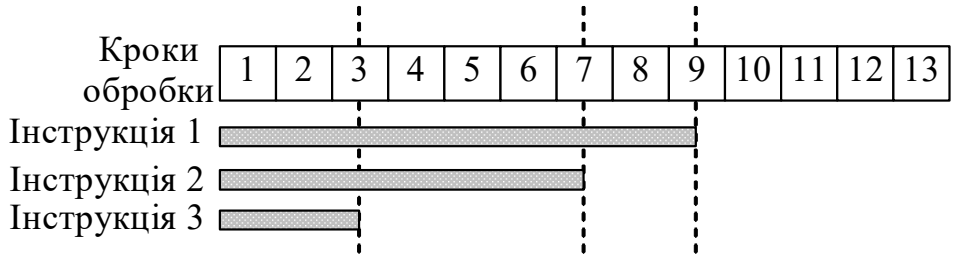

Рис. 2.6. Приклад конвеєра команд

На рис. 2.6 інструкція 1 знаходиться на 9-му кроці виконання, інструкція 2 – на 7-му, а інструкція 3 – на 3-му кроці.

Уперше для процесорів *Intel* конвеєр був реалізований в архітектурі процесора i80486. Конвеєр мав п’ять етапів. Архітектуру процесора з одним конвеєром називають *скалярною,* а з декількома – *суперскалярною*.

Паралельне виконання команд. Усі сучасні процесори у своєму складі мають декілька ALU, декілька пристроїв обробки чисел із рухомою комою та можуть включати деякі інші модулі для паралельної обробки мікроінструкцій. При цьому створюється враження (ілюзія) того, що паралельно працюють декілька процесорів.

Нехай на виконання надходить така послідовність інструкцій процесора (для зручності команди пронумеровані):

```
1. add eax, edx
2. inc ebx
```

3. sub eax, ecx
4. mov mem1, esi
5. fmul
6. shr ecx, 2.

Перед початком виконання першої інструкції зі складу цієї послідовності є декілька вільних у цей момент виконавчих модулів (рис. 2.7). Серед них один ALU, один FPU та модуль завантаження й вивантаження даних. Перша команда (add eax, edx) послідовності повинна виконуватись у модулі ALU, тому після декодування планувальник черги розмістить послідовність мікроінструкцій у чергу, яка відповідає даному типу інструкції.

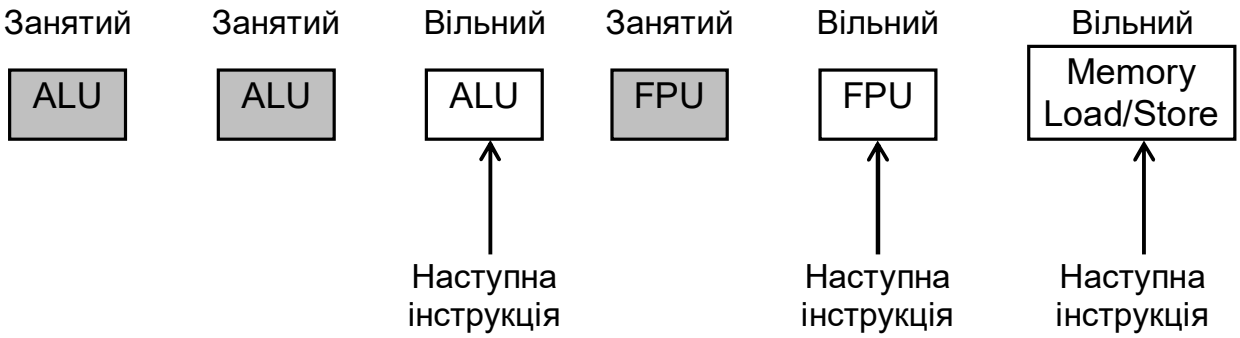

Рис. 2.7. Паралельне виконання команд

Через те, що в даний момент є вільний пристрій ALU, то диспетчер черги мікроінструкцій спрямує мікроінструкції команди на виконання саме в цей ALU. Перед виконанням другої команди (inc ebx) ситуація буде такою, як показано на рис. 2.8.

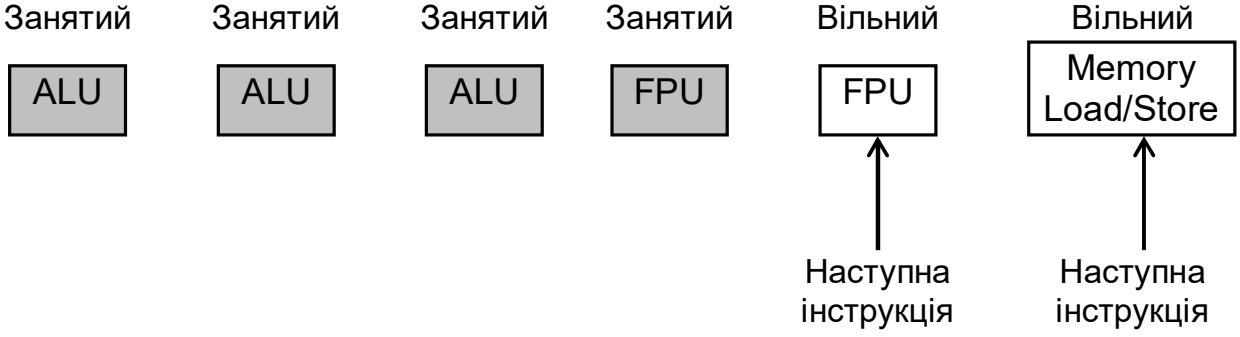

Рис. 2.8. Стан виконавчих модулів після виконання першої команди

Інструкція inc ebx повинна розміщуватись в один з ALU. Однак усі вони зайняті виконанням попередніх команд. Тому процесор переходить до інструкції 3 (sub eax, ecx). Її також неможливо виконати

через те, що всі ALU зайняті. Процесор переходить до інструкції 4 (mov mem1, esi), яка може виконуватись модулем обміну з пам'яттю (*Memory Load/Store*), вільним у цей момент і починає виконання інструкції. При переході до 5-ї інструкції процесор виявляє, що модуль FPU вільний, тому починається виконання цієї інструкції.

Такий механізм виконання команд процесора має загальноприйняту назву "механізм позачергового виконання команд" (*Out-Of-Order engine*, OOO). Цей механізм використовується всіма сучасними процесорами, через те що досягається висока продуктивність та виключаються затримки при виконанні послідовності команд.

Спекулятивне виконання команд (*Speculative Execution*). Даний механізм застосовується для розв'язання ситуацій розгалуження та переходу в інші точки програмного коду. Механізм спекулятивного виконання команд дозволяє одночасно виконувати декілька гілок розгалуженої програми у вільних виконавчих модулях (ALU, FPU), наприклад, за наявності умовного оператора (рис. 2.9). Після виконання обох віток алгоритму здійснюється перевірка умови. Результат, який відповідає неправильній альтернативі просто ігнорується. При цьому затримок у роботі процесора не спостерігається через те, що інструкції, які відповідають паралельним віткам алгоритму, виконувались у вільних виконавчих модулях.

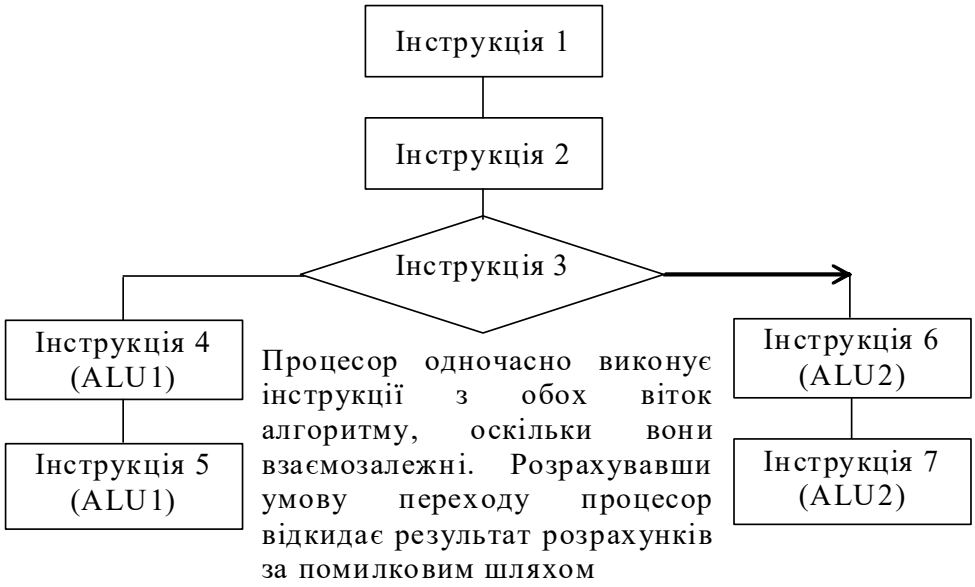

Рис. 2.9. Алгоритм спекулятивного виконання команд

Розглянуті механізми оптимізації продуктивності є загальними. Однак кожен процесор має свої додаткові механізми підвищення швидкості виконання програмного коду.

2.4. ПРОГРАМНА МОДЕЛЬ ПРОЦЕСОРА IA-32

2.4.1. Склад програмної моделі

Будь-яка програма, що виконується, має у своєму розпорядженні визначений набір ресурсів процесора. Ці ресурси необхідні для виконання та збереження в пам'яті команд програми, даних та інформації про поточний стан програми й процесора. Набір таких ресурсів представляє програмну модель процесора.

Програмна модель – це ресурси процесора, як доступні для використання в програмах, так і приховані від програміста (використовуються тільки процесором).

Базова програмна модель включає:

- простір адресовної пам'яті до 2^{32}–1 байт (4 *Гбайт*), для *Pentium III/IV* – до 2^{36}–1 байт (64 *Гбайт*);
- набір регістрів для зберігання даних загального призначення;
- набір сегментних регістрів, призначених для зберігання шести селекторів сегментів;
- набір регістрів стану та управління;
- набір регістрів пристрою розрахунків із рухомою комою (сопроцесор);
- набір регістрів цілочислового MMX-розширення;
- набір регістрів MMX-розширення з рухомою комою;
- програмний стек.

Це основний набір ресурсів. Крім того, до ресурсів, які підтримуються архітектурою IA-32, необхідно віднести порти введення-виведення, лічильники моніторингу продуктивності.

Набір регістрів. *Регістрами* називають області швидкодіючої пам'яті, які розташовані всередині процесора в безпосередній близькості від його виконавчого ядра. Доступ до них здійснюється набагато швидше, ніж до комірок оперативної пам'яті. Машинні команди з операндами в регістрах виконуються максимально швидко.

Більшість регістрів мають визначене функціональне призначення. З погляду програміста їх можна розділити на дві великі групи.

Першу групу утворюють користувацькі регістри, до яких належать:

- регістри загального призначення (EAX/AX/AH/AL, EBX/BX/BH/BL, EDX/DX/DH/DL, ECX/CX/CH/CL, EBP/BP, ESI/SI, EDI/DI, ESP/SP) використовуються для збереження даних та адрес, програміст може використовувати їх для реалізації своїх алгоритмів однак із деякими обмеженнями;

– сегментні регістри CS, DS, SS, ES, FS, GS використовуються для збереження адрес сегментів у пам’яті;

– регістри сопроцесора st(0), st(1), st(2), st(3), st(4), st(5), st(6), st(7) призначені для написання програм, що використовують тип даних з рухомою комою;

– цілочислові регістри MMX-розширення mmx0, mmx1, mmx2, mmx3, mmx4, mmx5, mmx6, mmx7;

– регістри MMX-розширення з рухомою комою xmm0, xmm1, xmm2, xmm3, xmm4, xmm5, xmm6, xmm7;

- регістри стану й управління – це регістри, які містять інформацію про стан процесора, виконуваної програми й дозволяють змінювати цей стан (регістри прапорців EFLAGS/FLAGS та регістр покажчика команд EIP/IP).

До другої групи входять системні регістри, тобто регістри призначені для підтримання різних режимів роботи, сервісних функцій, а також регістри, які є специфічними для різних моделей процесорів. До їх складу входять:

- регістри управління CR0…CR4 визначають режими роботи процесора та характеристики поточної виконуваної задачі;

- регістри управління пам’яттю GDTR, IDTR, LDTR, TR використовуються в захищеному режимі роботи процесора для локалізації керуючих структур цього режиму;

- регістри відлагодження DR0…DR7 призначені для моніторингу та управління різними аспектами відлагодження;

- регістри типів областей пам’яті MTRR використовуються для апаратного управління кешуванням із метою надання відповідних властивостей областям пам’яті;

- машинно-залежні регістри MSR використовуються для управління процесором, контролю за його продуктивністю, отримання інформації про помилки.

Префікс e означає extended.

2.4.2. Регістри загального призначення

Регістри загального призначення використовуються для збереження:

– операндів логічних і арифметичних операцій;

– компонент адрес;

– покажчиків на комірки пам’яті.

Усі ці регістри доступні для зберігання операндів без особливих обмежень, хоча в деяких випадках деякі з них мають певне функціональне призначення.

Усі регістри цієї групи дають можливість звертатися до своїх “молодших” частин. Звернення здійснюється за їх іменами. Важливо зазначити, що використовувати для самостійної адресації можна тільки 16- та 8-розрядні частини цих регістрів. Старші 16 бітів для самостійної роботи не доступні.

До регістрів загального призначення належать регістри, які фізично розміщені всередині ALU, тому їх інколи називають регістрами ALU:

- EAX/AX/AH/AL (*Accumulator register*) – регістр-акумулятор, застосовується для збереження результатів проміжних операцій, використовується у всіх операціях введення-виведення, в арифметичних операціях, у деяких командах його використання обов’язкове;

- EBX/BX/BH/BL (*Base register*) – базовий регістр, застосовується для збереження базової адреси деякого об’єкта в пам’яті (єдиний, який використовується в індексній адресації), може використовуватись при розрахунках для зберігання проміжних результатів;

- ECX/CX/CH/CL (*Count register*) – регістр-лічильник, використовується як лічильник циклів та елементів при виконанні строкових операцій, може використовуватись у розрахунках для зберігання проміжних результатів;

- EDX/DX/DH/DL (*Data register*) – регістр даних так само, як і регістр EAX/AX/AH/AL призначений для зберігання проміжних даних, у деяких командах його пряме використання обов’язкове, а в деяких він використовується неявно.

- ESI/SI (*source index register*) – регістр індексу джерела, містить адресу даних, які розміщуються в сегменті, що адресується регістром DS, а в ланцюгових операціях містить поточну адресу елемента в ланцюгу джерелі;

- EDI/DI (*destination index register*) – регістр індексу приймача, адресує дані, які розміщуються в сегменті, базова адреса якого знаходиться в регістрі ES, також може містити зміщення рядка-приймача при виконанні строкових операцій.

В архітектурі мікропроцесора на програмно-апаратному рівні підтримується така структура даних як стек. Для роботи зі стеком існують спеціальні регістри:

- ESP/SP (*Stack Pointer register*) – регістр покажчика стеку, який містить покажчик на верхівку стеку в поточному сегменті стеку;

- EBP/BP (*Base Pointer register*) – регістр покажчика бази кадру стеку призначений для організації довільного доступу до даних всередині стеку.

Розрядність регістрів надано на рис. 2.10.

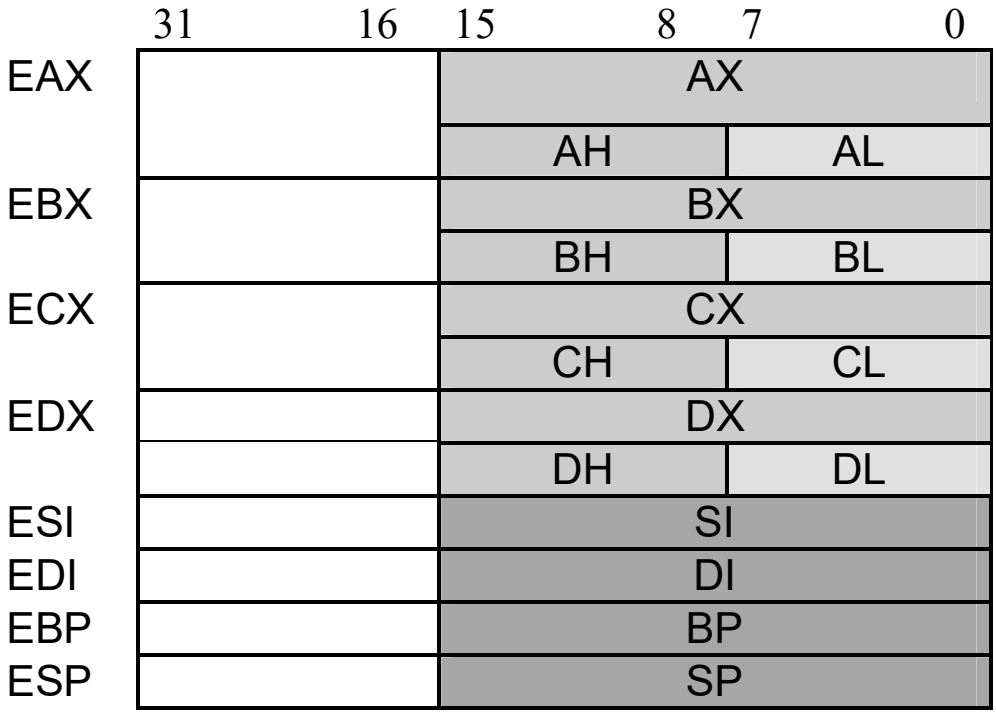

Рис. 2.10. Регістри загального призначення цілочислового пристрою

2.4.3. Сегментні регістри

Процесор *Intel* апартно підтримує сегментну організацію програми. Це означає, що будь-яка програма складається з трьох сегментів: коду, даних та стеку. Логічно машинні команди в архітектурі IA-32 побудовані так, що при виборі кожної команди для доступу до даних програми або стеку неявно використовується інформація, яка розміщена у визначених сегментних регістрах. Залежно від режиму роботи процесора за їх вмістом визначаються адреси пам'яті, з яких починаються відповідні сегменти. У програмній моделі IA-32 є шість сегментних регістрів CS, SS, DS, ES, GS та FS, які призначені для доступу до чотирьох типів сегментів.

Сегмент коду містить команди програми. Для доступу до цього сегмента служить регістр сегмента коду CS (*Code segment register*). Він містить адресу сегмента з машинними командами, до якого має доступ процесор (ці команди завантажуються в конвеєр процесора).

Сегмент даних містить дані, що обробляються програмою. Для доступу до цього сегмента служить регістр сегмента даних DS (*Data segment register*), у якому зберігається адреса сегмента даних поточної програми.

Сегмент стеку являє собою область пам'яті, яка називається стеком. Роботу зі стеком процесор організовує за таким принципом: останній записаний у цю область пам'яті елемент вибирається першим. Для доступу до цієї області призначений регістр сегмента стеку SS (*Stack segment register*), який містить адресу сегмента стеку.

Додатковий сегмент даних. Неявно алгоритми виконання більшості машинних команд передбачають, що дані, які ними обробляються, розташовані в одному сегменті даних, адреса якого розміщена в регістрі сегмента даних DS. Якщо програмі недостатньо одного сегмента даних, то вона має можливість задіяти ще три додаткових сегменти даних. Але на відміну від основного сегмента даних, адреса якого розміщується в DS, при використанні додаткових сегментів даних їх адреси потрібно вказувати явно за допомогою спеціальних префіксів перевизначення сегментів у команді. Адреси додаткових сегментів повинні міститися в регістрах додаткового сегмента даних ES, GS, FS (*Extension Data Segment register*).

2.4.4. Регістри стану та управління

До складу процесора включено два регістри, які містять як інформацію про стан самого процесора, так і програми, команди якої він обробляє:

- регістр-покажчик команд EIP/IP;
- регістр прапорців (EFLAGS/FLAGS).

За допомогою цих регістрів також можна у визначених межах управляти станом процесора.

Регістр-покажчик команд EIP/IP (*Instruction Pointer register*) має розрядність 32(16) біти та містить зміщення наступної команди, яка буде виконуватись, відносно вмісту регістра сегмента коду CS в поточному сегменті команд. Цей регістр програмісту безпосередньо не доступний, але завантаження та зміна його значення здійснюється різними командами управління, до яких команди умовних та безумовних переходів, виклику процедур та повернення з процедур. Виникнення переривань також призводить до модифікації регістра EIP/IP. Вміст цього регістра можна прочитати виконавши команду *call*, після чого переглянути вміст стеку – воно буде вказувати на наступну команду.

Розрядність *регістра прапорців* EFLAGS/FLAGS (*flag register*) дорівнює 32(16) *біт*. Окремі розряди цього регістра мають своє функціональне призначення й називаються *прапорцями*. Молодша частина

регістра EFLAGS повністю відповідає регістру FLAGS процесора i8086. На рис. 2.11 показано структуру регістра EFLAGS.

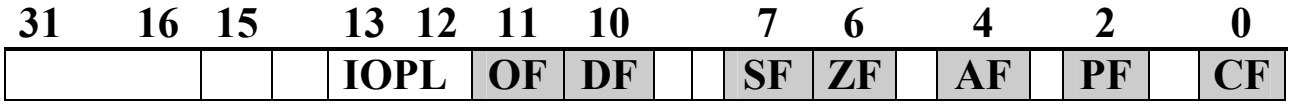

Рис. 2.11. Структура регістра EFLAGS

Прапорці регістра EFLAGS/FLAGS за функціональним призначенням поділяються на три групи.

До першої групи прапорців входять 8 прапорців стану. Ці прапорці можуть змінюватися після виконання машинних команд. Прапорці стану регістра EFLAGS відбивають особливості результату виконання арифметичних або логічних операцій. Це дає можливість аналізувати стан обчислювального процесу та реагувати на нього за допомогою команд умовних переходів та викликів підпрограм.

Призначення прапорців подано в табл. 2.1.

Таблиця 2.1.

Мнемоніка	Назва прапорця	Номер біта	Вміст та призначення
CF	Прапорець переносу (*Carry Flag*)	0	Встановлюється в 1, якщо арифметична операція зробила перенесення зі старшого біта результату. Старшим є 7, 15 або 31-й біт залежно від розмірності операнда; 0 – перенесення не було
PF	Прапорець паритету (*Parity Flag*)	2	Встановлюється в 1, якщо вісім молодших розрядів результату містять парну кількість одиниць (цей прапорець тільки для 8 молодших розрядів операндів будь-якого розміру). 0 – вісім молодших розрядів результату містять непарну кількість одиниць
AF	Допоміжний прапорець переносу (*Auxiliary Flag*)	4	Тільки для команд, працюючих з BCD-числами. Фіксує факт позики з молодшої тетради результату: 1 – в результаті операції додавання було зроблено перенесення з розряду 3 в старший розряд або при відніманні була позика в розряд 3 молодших тетради зі значення в старшій тетраді; 0 – перенесень і позик в(з) 3 розряд(а) молодшої тетради результату не було
ZF	Прапорець нуля (*Zero Flag*)	6	Встановлюється в 1, якщо результат операції дорівнює 0, і встановлюється в 0,

			якщо результат не нульовий
SF	Прапорець знака (*Sign Flag*)	7	Відбиває стан старшого біта результату (біти 7, 15 або 31), тобто знак результату операції, при від'ємному результаті SF = 1

Закінчення табл. 2.1

Мнемоніка	Назва прапорця	Номер біта	Вміст та призначення
OF	Прапорець переповнення (*Overflow Flag*)	11	Фіксує факт втрати значущого біта при арифметичних операціях, тобто ситуацію переповнення (вихід результату арифметичної операції за межі припустимого діапазону значень), OF = 1, якщо внаслідок операції виникає перенос у старший знаковий біт результату або позика із старшого знакового біта результату, OF=0, якщо внаслідок операції не виникло переносу у старший знаковий біт результату або позиці зі старшого знакового біта результату
IOPL	Рівень привілейованості вводу-виводу (*Input/ Output privilege level*)	12, 13	Використовується в захищеному режимі роботи процесора для контролю доступу до команд вводу-виводу залежно від привілейованості команд

Прапорці стану встановлюються після виконання операцій, результатами яких є беззнакові цілі числа, цілі числа зі знаком та упаковані (BCD) цілі числа. Якщо результатом операції є беззнакове ціле число, то встановлення прапорця переносу CF в 1 (перенос або позика) свідчить про вихід за межі припустимого діапазону. Якщо результатом операції є ціле число зі знаком (двійкове доповнення числа), про це свідчить встановлення прапорця переповнення OF в 1.

До другої групи прапорців (група прапорців управління) регістра EFLAGS/FLAGS входить лише один прапорець DF (*Direction Flag*) – прапорець напрямку, який використовується командами обробки рядків. Значення DF визначає напрямок поелементної обробки в цих операціях: від початку рядка до кінця чи навпаки. Якщо DF = 0, рядок обробляється в прямому напрямку, від менших адрес до старших, якщо DF = 1, обробка ведеться у зворотному напрямку.

До третьої групи прапорців регістра EFLAGS/FLAGS входять 8 системних прапорців, які керують введенням-виведенням, маскованим

перериванням, відлагоджуванням, перемиканням між задачами та режимом віртуального процесора i8086. Прикладним програмам не рекомендується модифікувати без необхідності ці прапорці, через те, що у більшості випадків це спричиняє зупинку роботи програми.

Контрольні питання

1. Які принципи покладено в основу будови сучасних комп'ютерів?
2. Дати визначення поняттю "архітектура ЕОМ".
3. Які елементи ЕОМ включає в себе поняття "архітектура"?
4. Дати визначення поняттю "програмна модель процесора".
5. Назвати набір ресурсів, що входять до складу програмної моделі процесора.
6. Вказати призначення регістрів загального призначення цілочислового пристрою.
7. Вказати призначення сегментних регістрів.
8. Для чого призначений регістр стану та управління?
9. Які прапорці включає до свого складу регістр стану та управління?
10. Назвати призначення прапорців регістра стану та управління.