Тема 4. Типи і формати команд. Архітектура системи команд

4.1 Типи команд

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

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

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

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

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

Команди для роботи з рядками. Для роботи з рядками в АСК зазвичай передбачаються команди, що забезпечують переміщення, порівняння і пошук

рядків. У більшості машин перераховані операції просто імітуються за рахунок інших команд.

SIMD-команди. Назва даного типу команд є абревіатурою від Single Instruction Multiple Data – буквально «одна інструкція – багато даних». На відміну від звичайних команд, що оперують двома числами, SIMD – команди обробляють відразу дві групи чисел (у принципі їх можна називати груповими командами). Операнди таких команд зазвичай подані в одному з упакованих форматів.

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

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

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

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

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

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

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

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

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

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

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

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

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

Однією з форм команд умовного переходу є команди пропуску. У них адреса переходу відсутня, а під час виконання умови відбувається пропуск наступної команди, тобто передбачається, що відсутня в команді адреса наступної команди еквівалентна адресі поточної команди, збільшеної на довжину команди, що пропускається. Такий прийом дозволяє скоротити довжину команд передачі управління.

Для всіх мов програмування характерне інтенсивне використання механізму процедур.

Процедурний механізм базується на командах виклику процедури, що забезпечують перехід з поточної точки програми до початкової команди

процедури, і на командах повернення з процедури, для повернення в точку, безпосередньо розташовану за командою виклику. Такий режим припускає наявність засобів для збереження поточного стану вмісту лічильника команд у момент виклику (запам'ятовування адреси точки повернення) і його відновлення під час виходу з процедури.

4.2 Формати команд

Типова команда, в загальному випадку, повинна указувати:

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

Відповідно до цього команда складається з двох частин: операційної і адресної (рис. 4.1).

Рисунок 4.1 – Структура команди

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

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

Довжина команди. Це найважливіша обставина, що впливає на організацію і ємність пам'яті, структуру шин, складність і швидкодію ЦП. З одного боку, зручно мати в розпорядженні могутній набір команд, тобто якомога більше кодів операцій, операндів, способів адресації, і максимальний адресний простір. Проте все це вимагає виділення більшої кількості розрядів під кожне поле команди, що приводить до збільшення її довжини. Разом з тим, для прискорення вибірки з пам'яті бажано, щоб команда була якомога коротша, а її довжина була рівна або кратна ширині шини даних. Для спрощення апаратури і підвищення швидкодії ОМ довжину команди зазвичай вибирають кратною байту, оскільки в більшості ОМ основна пам'ять організована у вигляді 8-бітових комірок. В рамках системи команд одної ОМ можуть використовуватися різні формати команд. Зазвичай це зв'язано із застосуванням різних способів адресації. У такому разі в склад коду команди вводиться поле для завдання способу адресації (СА), і узагальнений формат команди набуває вигляд, показаний на рис. 4.2.

Рисунок 4.2 – Узагальнений формат команди

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

$$R_K = \sum_{i=1}^{l} R_{Ai} + R_{KOn} + R_{CA} \qquad (4.1)$$

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

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

Розрядність поля коду операції. Кількість двійкових розрядів, що відводяться під код операції, вибирається так, щоб можна було подати будь-яку

з операцій. Якщо система команд припускає N_{KOn} операцій, то мінімальна розрядність поля коду операції R_{KOn} визначається таким чином:

$$R_{KOn} = \text{int}(\log_2 N_{KOn}) \tag{4.2}$$

де int означає округлення у більшу сторону до цілого числа.

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

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

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

Розрядності полів RAi і RCA розраховуються за формулами:

$$R_{Ai} = \text{int}(\log_2 N_i) \tag{4.3}$$

$$R_{CA} = \text{int}(\log_2 N_{CA}) \tag{4.4}$$

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

Кількість адрес у команді. Для визначення кількості адрес, що включаються в адресну частину, використовуватимемо термін адресність. У

«максимальному» варіанті необхідно вказати три компоненти: адреса першого операнда, адреса другого операнда і адреса комірки, куди заноситься результат операції. У принципі може бути додана ще одна адреса, що вказує місце зберігання наступної інструкції. У результаті має місце чотириадресний формат команди (рис. 4.3). Такий формат підтримувався у ОМ EDVAC, розробленою в 1940-х роках.

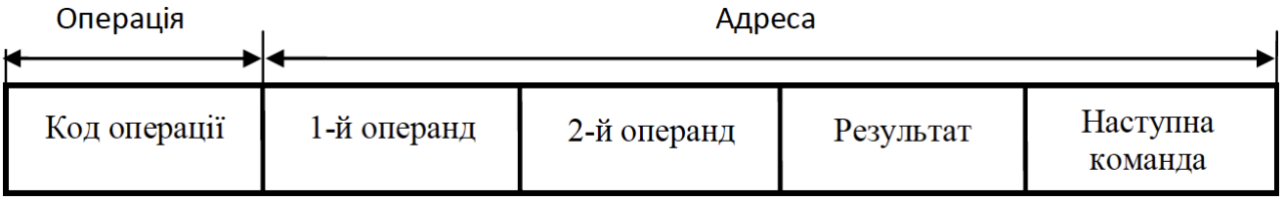

Рисунок 4.3 – Чотириадресний формат команди

У фон-нейманівских ОМ необхідність у четвертій адресі відпадає, оскільки команди розташовуються в пам'яті в порядку їх виконання, і адреса чергової команди може бути отримана за рахунок простого збільшення адреси поточної команди в лічильнику команд. Це дозволяє перейти до триадресного формату команди (рис. 4.4).

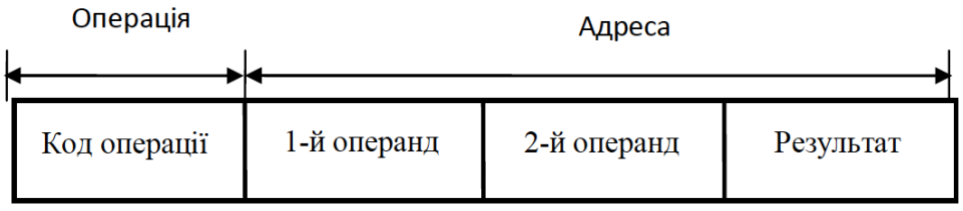

Рисунок 4.4 – Триадресний формат команди

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

Якщо за умовчанням взяти за адресу результату адресу одного з операндів (зазвичай другого), то можна обійтися без третьої адреси, і у результаті

отримуємо двоадресний формат команди (рис. 4.5). Природно, що в цьому випадку відповідний операнд після виконання операції втрачається.

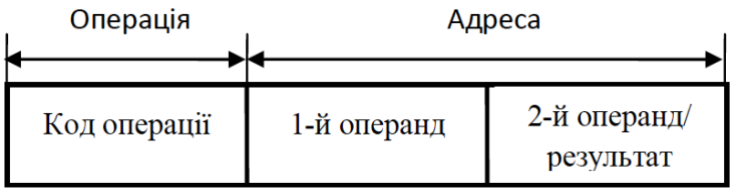

Рисунок 4.5 – Двоадресний формат команди

Команду можна ще більш скоротити, перейшовши до одноадресного формату (рис. 4.6), що можливо у випадку виділення певного стандартного місця для зберігання першого операнда і результату. Зазвичай для цієї мети використовується спеціальний регістр центрального процесора (ЦП), відомий під назвою акумулятора.

Рисунок 4.6 – Одноадресний формат команди

Застосування єдиного регістра для зберігання одного з операндів і результату є обмежуючим чинником, тому крім акумулятора часто використовують і інші регістри ЦП. Оскільки число регістрів в ЦП невелике, для вказівки одного з них у команді досить мати порівняно коротке адресне поле. Відповідний формат носить назву півтораадресного або регістрового формату (рис. 4.7).

Нарешті, якщо для обох операндів вказати чітко задане місцеположення, а також у разі команд, що не вимагають операнда, можна отримати нульадресний формат команди (рис. 4.8).

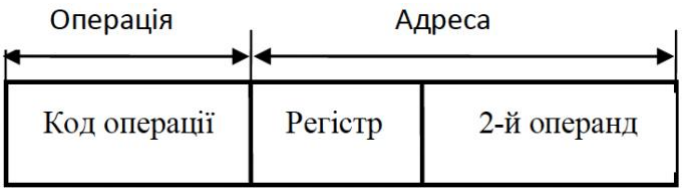

Рисунок 4.7 – Півтораадресний формат команди

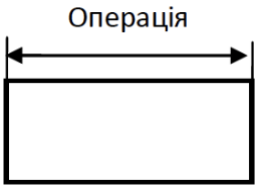

Рисунок 4.8 – Нульадресний формат команди

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

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

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

Адресність і ємність запам'ятовуючого пристрою. Якщо позначити кількість команд у програмі – N_A, розрядність команди, визначувану відповідно до формули (4.1) – R_{KA}, індекс, вказуючий адресність команд програми – A, то ємність запам'ятовуючого пристрою для зберігання програми E_A можна оцінити із співвідношення

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

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

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

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

Для триадресної команди останнє підсумовується з чотирьох складових часу:

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

Одноадресна команда вимагає двох звернень до пам'яті:

- вибірки команди;
- вибірки операнда.

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

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

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

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

У комбінованому алгоритмі обчислювальний процес утворюють як послідов-ні, так і паралельні частини. Тут найбільш переважними є одноадресні команди.

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

Враховуючи вищевикладене, перевагу слід віддати одноадресним командам.

4.3 Архітектура системи команд

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

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

1. Якого вигляду дані будуть подані в обчислювальній машині і у якій формі?

2. Де ці дані можуть зберігатися крім основної пам'яті?

3. Яким чином здійснюватиметься доступ до даних?

4. Які операції можуть бути виконані над даними?

5. Скільки операндів може бути присутніми в команді?

6. Як визначатиметься адреса чергової команди?

7. Яким чином будуть закодовані команди?

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

Серед мотивів, що найчастіше зумовлюють перехід до нового типу АСК, зупинимося на двох найбільш істотних. Перший – це склад операцій, що виконуються обчислювальною машиною, та їх складність. Другий – місце

зберігання операндів, що впливає на кількість і довжину адрес, які вказуються в адресній частині команд обробки даних.

4.3.1. Класифікація АСК за складом і складністю команд

Сучасна технологія програмування орієнтована на мови високого рівня (МВР), головна мета яких – полегшити процес програмування. Перехід до МВР, однак, породив серйозну проблему: складні оператори, характерні для МВР, істотно відрізняються від простих машинних операцій, що реалізовуються в більшості обчислювальних машин. Проблема отримала назву семантичного розриву, а її наслідком стає недостатньо ефективне виконання програм на ОМ. Намагаючись подолати семантичний розрив, розробники обчислювальних машин у даний час вибирають один з трьох підходів і, відповідно, один з трьох типів АСК:

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

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

- наявність у процесорі порівняно невеликого числа регістрів загального призначення;

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

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

Відмічений недолік і його наслідки привели до серйозного перегляду традиційних рішень, наслідком чого стала поява RISC-архітектури. Ідея полягає в обмеженні списку команд ОМ найбільш часто використовуваними простими командами, оперуючими даними, що розміщені тільки в регістрах процесора. Звернення до пам'яті допускається лише за допомогою спеціальних команд читання і запису. Різко зменшена кількість форматів команд і способів указівки адрес операндів. Скорочення числа форматів команд і їх простота, використання обмеженої кількості способів адресації, відділення операцій обробки даних від операцій звернення до пам'яті дозволяє істотно спростити апаратні засоби ОМ і підвищити їх швидкодію. RISC-архітектура розроблялася так, щоб зменшити час виконання програми за рахунок скорочення середньої кількості тактів процесора, що доводяться на одну команду, і тривалості тактового періоду. Як наслідок, реалізація складних команд за рахунок послідовності з простих, але швидких RISC-команд виявляється не менш ефективною, чим апаратний варіант складних команд в CISC-архітектурі.

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

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

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

4.3.2. Класифікація АСК за місцем зберігання операндів

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

З цих позицій розрізняють такі види архітектур системи команд: стекову; акумуляторну; регістрову; з виділеним доступом до пам'яті.

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

Стекова архітектура

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

Стек утворює множину логічно взаємозв'язаних комірок, що взаємодіють за принципом «останнім увійшов, першим вийшов» (LIFO - Last In First Out). Верхню комірку називають вершиною стека. Для роботи зі стеком передбачено

дві операції: push (проштовхування даних у стек) і pop (виштовхування даних із стека). Запис можливий тільки у верхню комірку стека, при цьому вся інформація, що зберігається в стеку, заздалегідь проштовхується на одну позицію вниз. Читання допустиме також тільки з вершини стека. Інформація, що витягнута, видаляється із стека, а його вміст, що залишився, просувається вверх. У обчислювальних машинах, де реалізована АСК на базі стека (їх зазвичай називають стековими), операнди перед обробкою поміщаються в дві верхні комірки стекової пам'яті. Результат операції заноситься в стек.

Основні вузли та інформаційні тракти одного з можливих варіантів ОМ на основі стекової АСК показані на рис. 4.9.

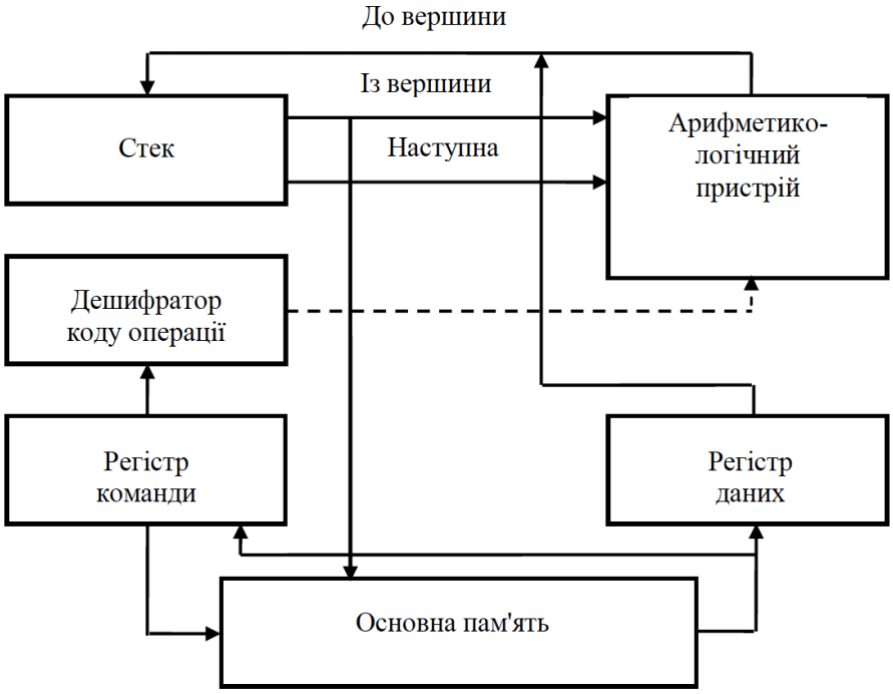

Рисунок 4.9 – Архітектура обчислювальної машини на базі стека

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

Збереження вмісту вершини стека в комірці пам'яті з адресою x проводиться командою pop x. За цією командою вміст верхньої комірки стека подається на шину, з якою і проводиться запис у комірку x, після чого вся інформація, що знаходиться в стеку, проштовхується на одну позицію вверх. Для виконання арифметичної або логічної операції на вхід АЛП подається інформація, лічена з двох верхніх комірок стека (при цьому вміст стека просувається на дві позиції вверх, тобто операнди із стека видаляються). Результат операції заштовхується у вершину стека. Можливий варіант, коли результат відразу ж переписується в пам'ять за допомогою автоматично виконуваної операції pop x.

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

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

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

Останні події в області обчислювальної техніки свідчать про відродження інтересу до стекової архітектури ОМ. Зв'язано це з популярністю мови Java і розширенням сфери застосування мови Forth, семантиці яких найбільш близька

саме стекова архітектура. Серед сучасних ОМ із стековою АСК можна згадати машини JEM1 і JEM2 компанії Systems і Clip фірми Imsys. Особливо слід зазначити стекову машину IGNITE компанії Patriot Scientist, яку її автори вважають представником нового виду АСК – архітектурою з безоперандним набором команд. Для позначення таких ОМ вони пропонують абревіатуру ROSC (Removed Operand Set Computer).

Акумуляторна архітектура

Архітектура на базі акумулятора історично виникла одною з перших. У ній для зберігання одного з операндів арифметичної або логічної операції в процесорі є виділений регістр – акумулятор. У цей же регістр заноситься і результат операції. Оскільки адреса одного з операндів зумовлена, в командах обробки досить явно вказати місцеположення тільки другого операнда. Спочатку обидва операнди зберігаються в основній пам'яті, і до виконання операції один з них потрібно завантажити в акумулятор. Після виконання команди обробки результат знаходиться в акумуляторі і, якщо він не є операндом для подальшої команди, його потрібно зберегти в комірці пам'яті. Для завантаження в акумулятор вмісту комірки x передбачена команда завантаження load x. За цією командою інформація зчитується з комірки пам'яті x, вихід пам'яті підключається до входів акумулятора і відбувається занесення зчитаних даних в акумулятор.

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

Типова архітектура ОМ на базі акумулятора показана на рис. 4.10.

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

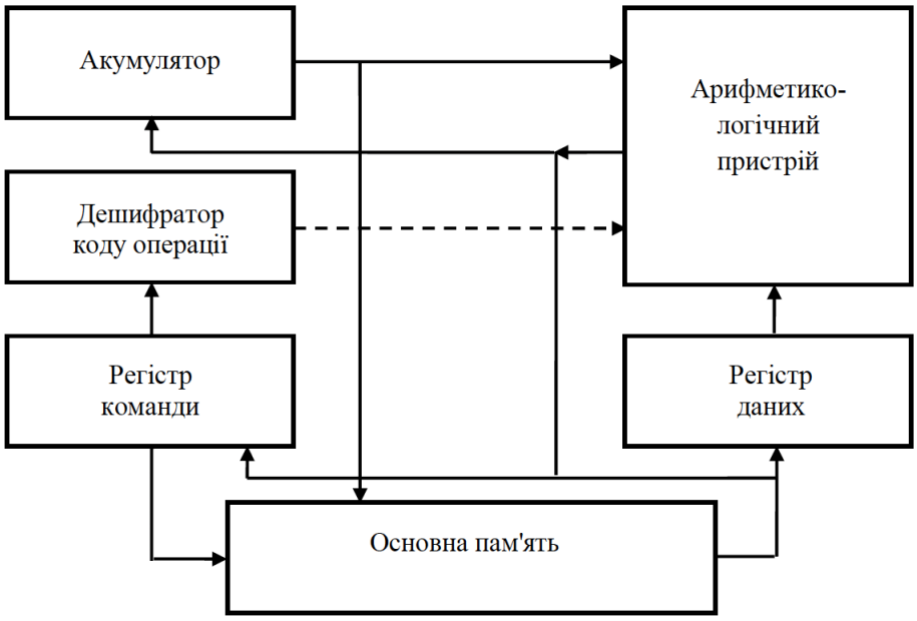

Рисунок 4.10 – Архітектура обчислювальної машини на базі акумулятора

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

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

Регістрова архітектура

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

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

сотень), проте типова для таких ОМ довжина команди (звичайно 32 розряди) дозволяє визначити в команді до трьох регістрів.

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

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

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

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

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

Можливу структуру та інформаційні тракти обчислювальної машини з регістровою архітектурою системи команд ілюструє рис. 4.11.

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

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

Виконання операції в АЛП включає:

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

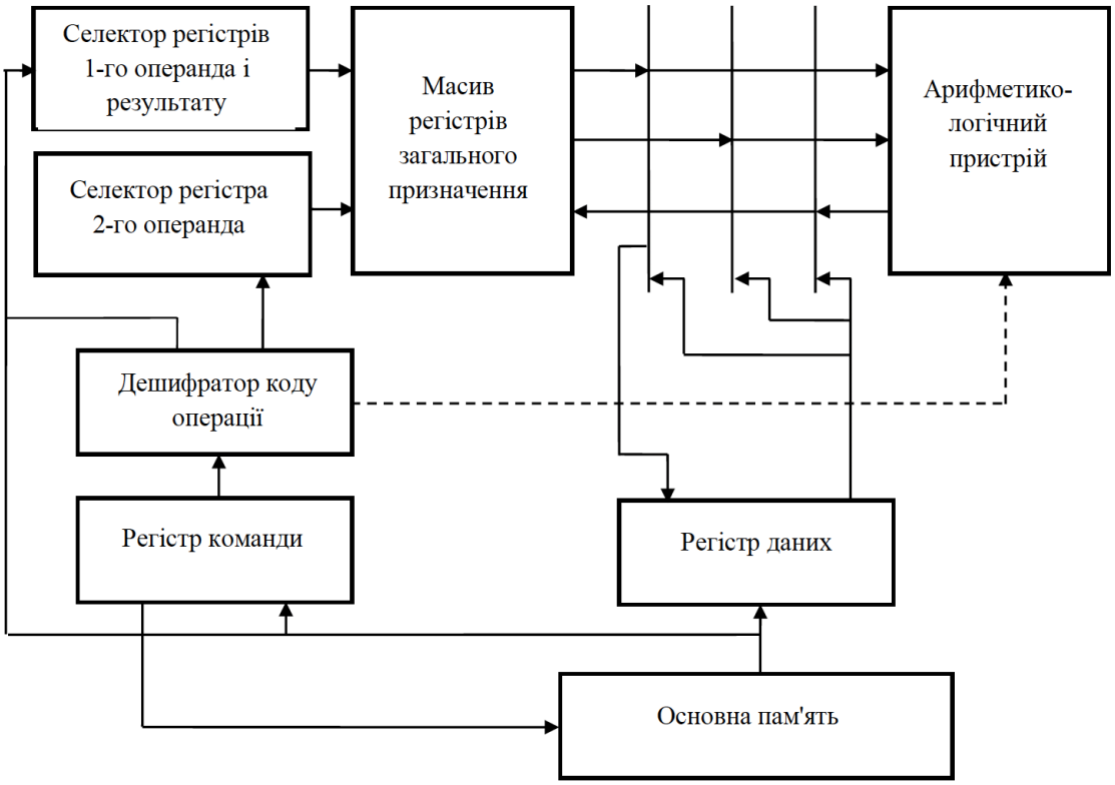

Рисунок 4.12 – Архітектура обчислювальної машини на базі регістрів загального призначення

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

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

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

Архітектура з виділеним доступом до пам'яті

В архітектурі з виділеним доступом до пам'яті звернення до основної пам'яті можливо тільки за допомогою двох спеціальних команд: load і store. В англійській транскрипції дану архітектуру називають Load/Store architecture. Команда load (завантаження) забезпечує зчитування значення з основної пам'яті

і занесення його в регістр процесора (у команді зазвичай вказується адреса комірки пам'яті і номер регістра). Пересилка інформації в протилежному напрямі проводиться командою store (збереження). Операнди у всіх командах обробки інформації можуть знаходитися тільки в регістрах процесора (найчастіше в регістрах загального призначення). Результат операції також заноситься в регістр. В архітектурі відсутні команди обробки, що допускають пряме звернення до основної пам'яті.

Склад та інформаційні тракти ОМ з виділеним доступом до пам'яті показані на рис. 4.13.

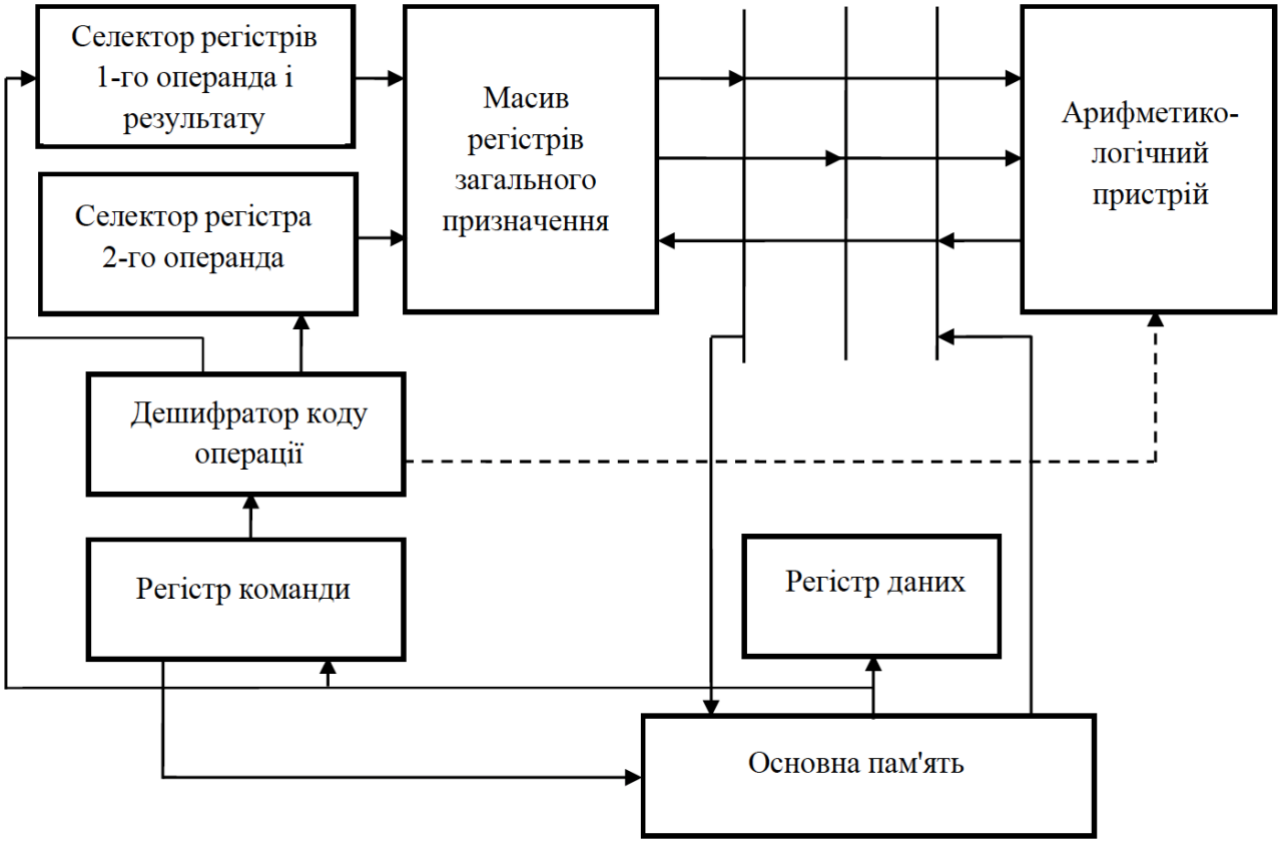

Рисунок 4.13 – Архітектура обчислювальної машини з виділеним доступом до пам'яті

Дві з трьох шин, розташованих між масивом РЗП і АЛП, забезпечують передачу в арифметико-логічний пристрій операндів, що зберігаються в двох регістрах загального призначення. Третя служить для занесення результату у

виділений для цього регістр. Ці ж шини дозволяють завантажити в регістри вміст елементів основної пам'яті і зберегти в ОП інформацію, що знаходиться в РЗП.

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