В. І. Дужий, В. В. Дужа

АРХІТЕКТУРА КОМП'ЮТЕРІВ. ВСТУП

2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний аерокосмічний університет ім. М. Є. Жуковського
“Харківський авіаційний інститут”

В. І. Дужий, В. В. Дужа

АРХІТЕКТУРА КОМП'ЮТЕРІВ. ВСТУП

Навчальний посібник

Харків «ХАІ» 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний аерокосмічний університет ім. М. Є. Жуковського
“Харківський авіаційний інститут”

В. І. Дужий, В. В. Дужа

АРХІТЕКТУРА КОМП'ЮТЕРІВ. ВСТУП

Навчальний посібник

Харків «ХАІ» 2020

УДК 004.2(075.8)
Д81

Рецензенти: д-р техн. наук, проф. Г. А. Кучук,
канд. техн. наук, доц. О. В. Щербаков

Дужий, В. І.

Д81 Архітектура комп'ютерів. Вступ [Електронний ресурс]: навч. посіб. / В. І. Дужий, В. В. Дужа. – Харків: Нац. аерокосм. ун-т ім. М. Є. Жуковського «Харків. авіац. ін-т», 2020. – 93 с.

Викладено принципи організації процесорів і пам'яті, а також їхні характеристики, які повинні знати спеціалісти в області інформаційних технологій.

Описано архітектури пам'яті й процесорів. Розглянуто основні характеристики процесорів і пам'яті. Наведено приклади процесорів з основними видами архітектур.

Для студентів очної й заочної форм навчання за напрямом "Інформаційні технології" за спеціальностями "Комп'ютерна інженерія" та "Кібербезпека" при вивченні дисциплін "Основи функціонування комп'ютерів", "Технології програмування", "Архітектура комп'ютерів", "Комп'ютерна логіка", "Операційні системи", "Мікропроцесорні системи".

Іл. 41. Табл. 15. Бібліогр.: 8 назв

УДК 004.2(075.8)

ЗМІСТ

СПИСОК СКОРОЧЕНЬ 5
1. БАЗОВІ ПОНЯТТЯ ПРО ОРГАНІЗАЦІЮ КОМП'ЮТЕРА 6
1.1. Структура типового комп'ютера 6
1.2. Поняття архітектури комп'ютера 7
1.3. Історія розроблення архітектури комп'ютерів 10
1.4. Принципи організації комп'ютера фон Неймана 10
1.5. Гарвардська архітектура 12
1.6. Структура комп'ютера з архітектурою фон Неймана 13
Запитання до розділу 1 14
2. АРХІТЕКТУРА ПІДСИСТЕМИ ПАМ'ЯТІ 14
2.1. Характеристики підсистеми пам'яті 15
2.1.1. Організація пам'яті 16
2.1.2. Місткість пам'яті комп'ютера 18
2.1.3. Одиниці виміру місткості пам'яті МЕК і IEEE 19
2.1.4. Більш крупні одиниці виміру місткості пам'яті 20
2.1.5. Тип пам'яті 20
2.1.6. Порядок запису байтів мультибайтових чисел 22
2.1.7. Спосіб розміщення в пам'яті мультибайтових чисел 23
2.1.8. Розподіл адресного простору комп'ютера 25
2.1.9. Розподіл адресного простору МП x86 25
2.1.10. Розподіл адресного простору МК MCS-51 27
2.2. Моделювання пам'яті 30
Запитання до розділу 2 31
3. АРХІТЕКТУРА ЦЕНТРАЛЬНИХ ПРОЦЕСОРІВ 32
3.1. Класифікація процесорів 33
3.2. Структура найпростішого процесора 36
3.3. Формати машинних команд 38
3.4. Основний машинний цикл процесора 39
3.4.1. Етапи виконання команди 40

3.4.2. Моделювання основного машинного циклу 41
3.4.3. Інші машинні цикли процесора 42
Запитання до розділу 3 43
4. ХАРАКТЕРИСТИКИ ПРОЦЕСОРА 44
4.1. Тип організації комп'ютера 44
4.2. Ідеологія формування системи команд процесора 45
4.3. Розрядність процесора 46
4.4. Адресний простір процесора 47
4.5. Архітектура процесора 48
4.5.1. Архітектура з акумулятором 49
4.5.2. Архітектура з регістрами загального призначення 49
4.5.3. Архітектура зі стеком 51
4.6. Кількість адрес пам'яті, які задаються в команді 53
Запитання до розділу 4 56
5. АРХІТЕКТУРА ПРОЦЕСОРІВ 57
5.1. Процесор, який має архітектуру з акумулятором 57
5.2. Процесор, який має архітектуру з регістрами загального призначення 63
5.3. Процесор, який має архітектуру зі стеком 68
5.4. Порівняльний аналіз архітектур процесорів 77
5.5. Неархітектурні характеристики процесорів 81
5.6. Літографія 83
5.7. Закон Мура 84
5.8. Архітектура процесорів фірми Intel 88
5.8.1. Архітектура процесорів Intel x86 88
5.8.2. Архітектура процесорів Intel x64 88
5.8.3. Архітектура процесорів Intel IA-64 89
Запитання до розділу 5 89
Вправи до розділу 5 90
ВИСНОВКИ 91
БІБЛІОГРАФІЧНИЙ СПИСОК 92

СПИСОК СКОРОЧЕНЬ

BIOS – Basic Input Output System, базова система введення-виведення
CISC – Complex Instruction Set Command, складна система команд
CPU – Central Processing Unit; те саме, що і ЦП
EEPROM – Electrically Erasable Programmable ROM; те саме, що і ЕППЗП
IEC – International Electrotechnical Commission, Міжнародна електротехнічна комісія
IEEE – The Institute of Electrical and Electronics Engineers, Інститут інженерів з електротехніки та електроніки, IIEP (США)
IP – Instruction Pointer, покажчик команд
PC – Program Counter, лічильник команд
PROM – Programmable ROM; те саме, що і ППЗП
RAM – Random Access Memory; те саме, що і ОЗП
RISC – Reduced Instruction Set Command, спрощена система команд
ROM – Read Only Memory; те саме, що і ПЗП
SCSI – Small Computer Systems Interface, інтерфейс малих комп'ютерних систем
SP – Stack Pointer, покажчик стеку
ВІС – велика інтегральна схема
ЕППЗП – електрично програмований постійний запам'ятовувальний пристрій
ЗП – запам'ятовувальний пристрій
МЕК – Міжнародна електротехнічна комісія
МК – мікроконтролер
МП – мікропроцесор
ОА – операційний автомат
ОЗП – оперативний запам'ятовувальний пристрій
ОП – операційний пристрій
КП – керувальний пристрій
ПЗП – постійний запам'ятовувальний пристрій
ПЛК – програмований логічний контролер
ППЗП – перепрограмований постійний запам'ятовувальний пристрій
РЗП – регістр загального призначення
ЦП – центральний процесор
СП – співпроцесор
ШД – шина даних
ША – шина адрес
ШК – шина керування
ЦВиб – цикл вибірки команди
ЦВик – цикл виконання команди

1. БАЗОВІ ПОНЯТТЯ ПРО ОРГАНІЗАЦІЮ КОМП'ЮТЕРА

1.1. Структура типового комп'ютера

Перші комп'ютери в їх сучасному вигляді з'явилися в 40-ві роки XX століття. Це були пристрої для виконання обчислень, які споживали потужність сотні кіловат, займали площу сотні квадратних метрів і мали швидкодію кілька тисяч операцій за секунду. Технологічний прогрес дав змогу зменшити розміри комп'ютерів і споживану потужність, одночасно збільшивши швидкодію в мільйони разів і створивши персональний комп'ютер в його сучасному вигляді. Однак основні функціональні вузли, що входять до складу комп'ютера, і принципи функціонування комп'ютерів зазнали лише незначних змін.

Типовий сучасний комп'ютер складається з п'яти підсистем: *центрального процесора (ЦП)*, *пам'яті*, *підсистеми введення-виведення*, *комунікаційної підсистеми* і *системної магістралі* (рис. 1.1).

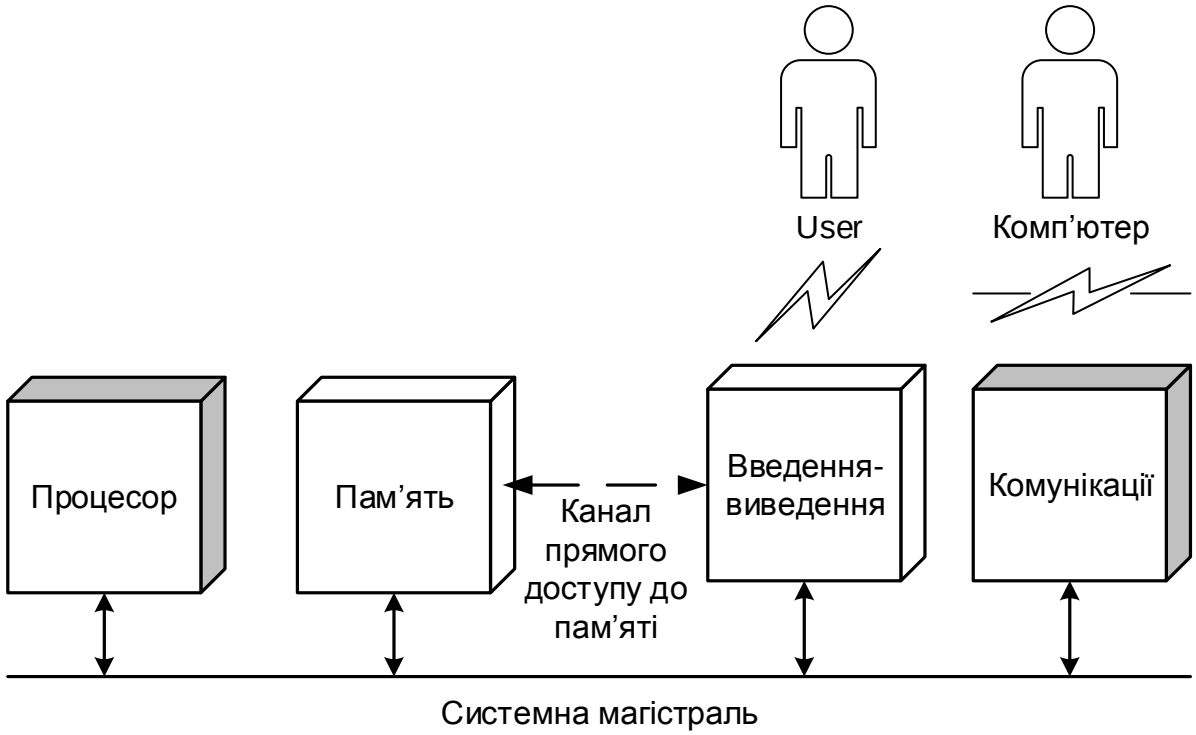


Рис. 1.1. Структура типового комп'ютера

Центральний процесор призначений для оброблення даних і керування комп'ютером. Сучасні ЦП зазвичай виконують за технологією великих інтегральних схем (ВІС), і такі ЦП називаються *мікропроцесорами* (МП). *Підсистема пам'яті* використовується для зберігання даних і команд.

Підсистема введення-виведення потрібна для обміну даними між користувачем і комп'ютером. Завдяки їй дані потрапляють у комп'ютер із навколишнього середовища, а результати роботи комп'ютера виводяться назовні. *Комунікаційна підсистема* призначена для обміну інформацією між комп'ютерами, які підключені до локальної або глобальної мережі [1].

Усі підсистеми об'єднуються між собою за допомогою системної магістралі. *Системна магістраль* (або системний інтерфейс) – уніфікований набір ліній, призначених для обміну інформацією всередині комп'ютера.

1.2. Поняття архітектури комп'ютера

Архітектура системи – це компоненти, які входять до її складу, зв'язки між ними, а також взаємодія компонентів під час функціонування системи. Це загальне визначення, справедливе для будь-якої системи (природної, біологічної, технічної або соціальної). У чому саме полягають особливості архітектури комп'ютерних систем розглянемо далі.

Сучасні комп'ютери мають дуже складну організацію, і її можна розглядати з двох точок зору – розробника апаратної частини і розробника програмного забезпечення, яке безпосередньо взаємодіє з апаратурою комп'ютера. Щоб абстрагуватися від несуттєвих деталей функціонування, комп'ютер розглядають у вигляді певної сукупності елементів, які використовують розробники апаратної або програмної частини. Абстрактне подання комп'ютера у вигляді набору взаємозв'язаних і взаємодіючих компонентів, які використовують розробники, називається *архітектурою комп'ютера*. Отже, можна розглядати як апаратну, так і програмну архітектури комп'ютера.

Архітектура комп'ютера – це організація комп'ютера з точки зору програміста. Оскільки комп'ютер складається з трьох компонентів, то, відповідно, виділяють архітектуру процесора, архітектуру підсистеми пам'яті й архітектуру підсистеми введення-виведення. Одні апаратні характеристики комп'ютера впливають на його архітектуру, інші – визначають його споживчі і цінові показники. Наприклад, кількість регістрів процесора, система команд і режими адресації, реалізовані в даному процесорі, визначають його архітектуру. Зміна будь-якого елемента архітектури може призвести до того, що програми, розроблені для даної архітектури, перестануть коректно функціонувати (якщо вони взагалі будуть працювати). Однак тактова частота процесора не є архітектурною характеристикою і при коректно розроблених програмах вона впливає лише на їх час виконання, а не на працездатність.

Архітектура комп'ютера не є чимось застиглим і незмінним – вона має

тенденцію до *змінювання (розширення чи звуження)*. Розширюється система команд процесора, додаються нові режими адресації, збільшується розрядність оброблюваних процесором даних. Ці та багато інших змін можуть призвести до того, що машинні коди програм, написаних для попереднього покоління процесорів, будуть неправильно працювати з новим процесором. Тому поряд із розширенням архітектури розробники прагнуть зберегти її *сумісність* з уже існуючою за принципом "знизу вгору". Такий принцип означає, що програми, написані для попередньої архітектури, будуть коректно виконуватись на процесорах з новою архітектурою. Це дає можливість використовувати нові архітектурні властивості для розроблення більш ефективного програмного забезпечення і водночас коректно працювати програмам, розробленим для попередньої архітектури.

Розрізняють два види архітектурної сумісності: "знизу вгору" і "згори вниз". Якщо елементи попередньої архітектури є підмножиною нової, то вважають, що архітектури комп'ютерів сумісні "знизу вгору". У цьому випадку раніше написані програми можуть виконуватися на комп'ютері з новою архітектурою. Якщо, навпаки, елементи нової архітектури є підмножиною попередньої, то кажуть, що архітектури комп'ютерів сумісні "згори вниз". У цьому випадку програми, написані для нової архітектури, на попередній архітектурі можуть не працювати. Однак зазвичай розробники прагнуть забезпечити сумісність процесорів "знизу вгору", розширюючи архітектурні можливості процесора.

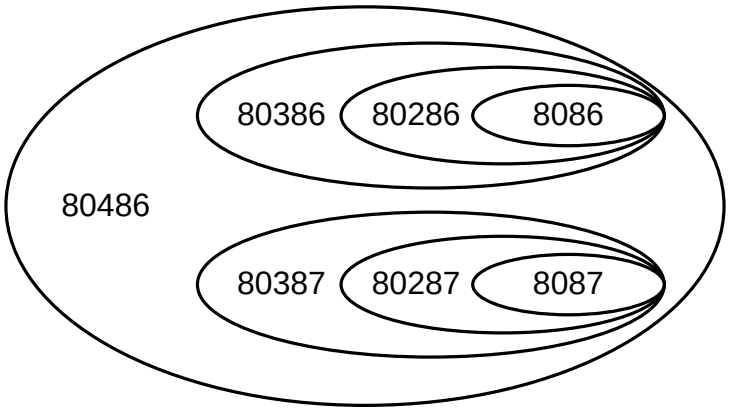


Рис. 1.2. Сумісність системи команд мікропроцесорів Intel від 8086 до Pentium

При вивченні архітектури процесорів деякі їхні архітектурні особливості можна пояснити лише прагненням зберегти сумісність із колишньою архітектурою. Наприклад, система команд МП Intel 8086 є підмножиною системи команд МП Intel 80286, яка, в свою чергу, є підмножиною системи

команд МП Intel 80386 і т.д. Таким чином, система команд МП Intel Pentium є розширенням системи команд МП Intel 8086, і ці процесори сумісні між собою за принципом "знизу вгору" (рис. 1.2).

Регістр прапорців лінійки мікропроцесорів Intel x86 також є хорошим прикладом розширення архітектури із збереженням сумісності архітектур. 8-розрядний мікропроцесор Intel 8080 мав 8-розрядний регістр прапорців, а 16-розрядний МП Intel 8086 – 16-розрядний регістр прапорців, молодші 8 бітів якого повністю збігалися з регістром прапорців МП Intel 8086. При цьому зазвичай використовують молодші 12 бітів регістра прапорців (з 0 по 11 біт). У МП Intel 80286 використовують 15 молодших бітів (з 0 по 14 біт). При цьому склад, функціональне призначення і розташування бітів повністю збігаються зі складом, функціональним призначенням і розташуванням бітів у МП Intel 8086. Цей підхід зберігся і в наступних моделях мікропроцесорів. У МП Intel 80386 вже використовувалися 19 молодших бітів (з 0 по 18 біт), а в МП Intel 80486 – 22 біти (з 0 по 21 біт) (рис. 1.3).

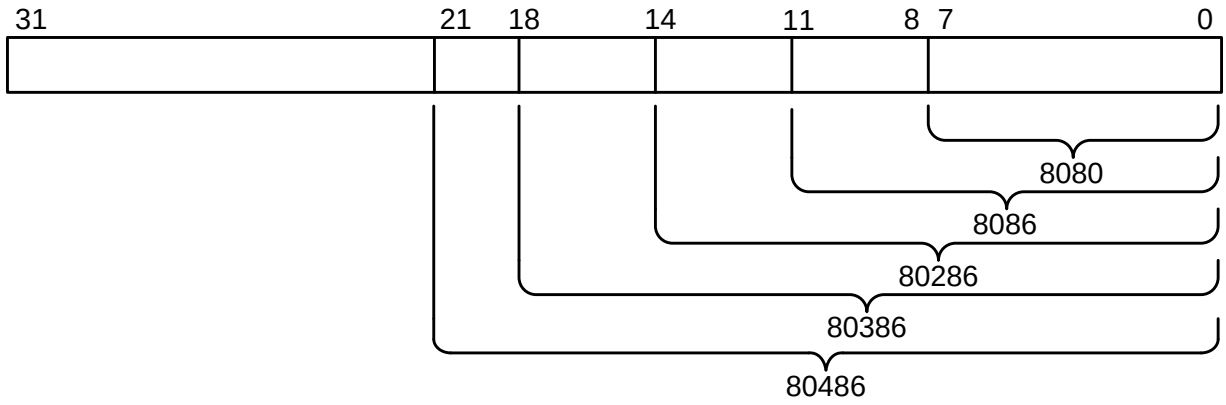


Рис. 1.3. Регістр прапорців мікропроцесорів Intel від 8080 до Pentium

Таким чином, якщо два комп'ютери мають однакові або сумісні архітектури процесорів, то програми в машинних кодах для одного процесора будуть коректно виконуватися і на іншому. Якщо два процесори мають несумісні архітектури, то програми в машинних кодах, написані для одного процесора, працювати на іншому не будуть.

З погляду розробника апаратури, архітектура комп'ютера являє собою сукупність конструктивних і електричних параметрів функціональних вузлів комп'ютера, взаємодія яких визначається часовими діаграмами обміну інформацією. Очевидно, що зазначені характеристики визначають набір апаратних інтерфейсів, які використовують при розробленні апаратури комп'ютера. Для розробника апаратури важливі такі характеристики процесора, як тип корпусу, тактова частота і частота системної шини. Для

підсистеми пам'яті важливі тип інтерфейсу пам'яті (SIMM, DIMM або інший) і швидкодія пам'яті, а для підсистеми введення-виведення – насамперед тип інтерфейсу, за допомогою якого буде підключатися адаптер цього пристрою до системної шини, а пристрій – до адаптера. Для підключення адаптера до системної шини використовують інтерфейси ISA, PCI і AGP, а для підключення периферійного пристрою до адаптера – інтерфейси COM, LPT і USB. Кожен із зазначених інтерфейсів має свої конструктивні, електричні, часові параметри і свої, властиві тільки йому, протоколи обміну даними.

Слід зазначити, що апаратна архітектура комп'ютера також схильна до змін. З одного боку, до архітектури комп'ютера додаються нові інтерфейси, такі як USB, HDMI, а з іншого – певні інтерфейси вилучаються із складу комп'ютера, наприклад, COM-інтерфейс. COM-інтерфейсу в сучасних ноутбуках зазвичай немає, тому при виборі такого комп'ютера слід ретельно вивчати специфікацію його інтерфейсів, щоб не зазнати втрат при експлуатації використовуваного програмного забезпечення.

Структура комп'ютера – сукупність цифрових елементів і зв'язків між ними. Це – реалізація програмної архітектури за допомогою певної елементної бази. Структура підсистем комп'ютера містить логічні елементи, тригери, регістри та інші елементи середнього ступеня інтеграції. Структура підсистем комп'ютера може мати великі інтегральні схеми, такі як спеціалізовані процесори і елементи пам'яті. Неважко зрозуміти, що саме структура комп'ютера разом з архітектурою визначають його основні споживчі характеристики – продуктивність і ціну.

1.3. Історія розроблення архітектури комп'ютерів

У 30-х роках уряд США доручив Гарвардському і Прінстонському університетам розробити архітектуру комп'ютера для військово-морської артилерії. Перемогла розробка Прінстонського університету (більш відома як архітектура фон Неймана, названа так на ім'я розробника, який першим надав звіт про неї), оскільки вона була простою в реалізації. Гарвардська архітектура не використовувалася аж до кінця 70-х років XX століття.

1.4. Принципи організації комп'ютера фон Неймана

Перші комп'ютери мали два типи пам'яті: в одній пам'яті зберігалася програма, а в іншій – дані. Ідея створення обчислювальної машини, здатної зберігати дані й команди у спільній пам'яті, належить Мочлі і Еккерту. Вона виникла приблизно у 1943 р. під час їх роботи над комп'ютером "EDVAC". Ідеї Мочлі і Еккерта були узагальнені і викладені в роботі математика

угорського походження Джона фон Неймана і тепер відомі як "принципи організації комп'ютера фон Неймана". Ці принципи визначили основні архітектурні риси сучасних комп'ютерів.

Архітектура фон Неймана (англ. Von Neumann architecture) – широко відомий принцип використання пам'яті комп'ютера для спільного зберігання програм і даних. Часто архітектуру фон Неймана називають *прінстонською архітектурою*.

Принципи організації комп'ютера фон Неймана визначили концепцію комп'ютера "*з програмою, яка зберігається у спільній пам'яті*" і в загальних рисах зводяться до такого:

1) дані і команди кодуються за допомогою *двійкового коду*;

2) дані і команди зберігаються *у спільній пам'яті*;

3) дані і команди розрізняються не за способом зберігання, а *за способом їх використання*;

4) команди виконуються, як правило, *послідовно*, хоча є можливість виконувати команди у довільному порядку.

Другий принцип є основоположним, оскільки визначає суть концепції. Сьогодні, через багато років, принципи фон Неймана здаються природними і зрозумілими, але свого часу вони були далеко неочевидними і революційними. Саме завдяки принципам фон Неймана було визначено вектор розвитку комп'ютерів в їх сучасному розумінні.

У перших комп'ютерах, які створювалися у військових лабораторіях і університетах для подання чисел, використовувалася десяткова система счислення. І в наш час деякі автори пропонують використовувати десяткові коди для подання інформації у комп'ютерах. Однак саме двійкове кодування дало змогу створити апаратну частину комп'ютера з найменшими витратами.

Другий принцип архітектури фон Неймана означає, що дані й команди зберігаються в спільній пам'яті. Коли закодована двійковим кодом інформація знаходиться в пам'яті комп'ютера, неможливо відрізнити дані від команд, як і неможливо визначити тип даних, які вже зберігаються у пам'яті.

Слід зазначити, що дані й команди у розумінні комп'ютера відрізняються: команди є інструкціями для їх виконання, тоді як дані обробляються командами процесора і являють собою своєрідну "їжу" для нього. Тому комп'ютер повинен відрізняти дані від команд, і для цього в архітектуру процесора введено спеціальний засіб, який адресує команди процесора. Цим засобом є регістр, який називається *покажчиком команд* (IP – Instruction Pointer), або *лічильником команд* (PC – Program Counter). Покажчик команд завжди вказує на команду, яка буде виконуватися *наступною*. Термін "вказує на команду" означає, що покажчик команд

"зберігає адресу команди".

Команди процесора розташовані у пам'яті послідовно. Так само послідовно їх витягують із пам'яті, подають у процесор, який їх обробляє. Після вилучення чергового байта команди з пам'яті покажчик команд IP необхідно збільшити на одиницю для переходу до наступного байта команди:

```
// перевести покажчик команд на наступну команду
   IP = IP+1;
// перевести покажчик команд на наступну команду
   PC = PC+1;
```

Оскільки покажчик команд завжди збільшується на одиницю для адресації наступної команди, у багатьох процесорах його називають лічильником команд.

Хоча покажчик команд зазвичай збільшується на одиницю, процесор може адресувати довільну команду, розташовану в пам'яті. Такі операції називають операціями переходу. Для цього потрібно занести відповідну адресу в покажчик команд.

```
// налаштувати покажчик команд на команду з адресою addr
   IP = addr;
// налаштувати покажчик команд на команду з адресою addr
   PC = addr;
```

Принципи організації комп'ютерів фон Неймана лежать в основі організації майже всіх сучасних універсальних комп'ютерів, у тому числі й персональних, оскільки співвідношення між обсягом даних і команд у програмах користувача може змінюватися в дуже широких межах.

1.5. Гарвардська архітектура

Як зазначалося раніше, в прінстонській архітектурі для зберігання даних і команд використовується спільна пам'ять (рис.1.4, а), проте існує й інший принцип організації комп'ютерів, який називається "*гарвардська архітектура*". Її суть полягає у тому, що для зберігання даних і команд використовуються різні види пам'яті (рис. 1.4, б).

Гарвардська архітектура – архітектура комп'ютера, відмітними ознаками якої є роздільне зберігання й оброблення команд і даних. Ця архітектура була розроблена Говардом Ейкеном наприкінці 30-х років у Гарвардському університеті.

Така організація архітектури комп'ютерів використовується в однокристальних мікроконтролерах (МК) і сигнальних процесорах і дає можливість створювати більш ефективні пристрої, які використовуються, перш за все, для створення вбудованих систем керування. Гарвардська

архітектура ефективна тоді, коли обсяг оброблюваних даних набагато менше обсягу обробних програм, що характерно саме для МК і сигнальних процесорів. Гарвардська архітектура використовується у програмованих логічних контролерах (ПЛК) і МК, таких як Atmel AVR, Intel MCS-8051, ST Microelectronics STM32.

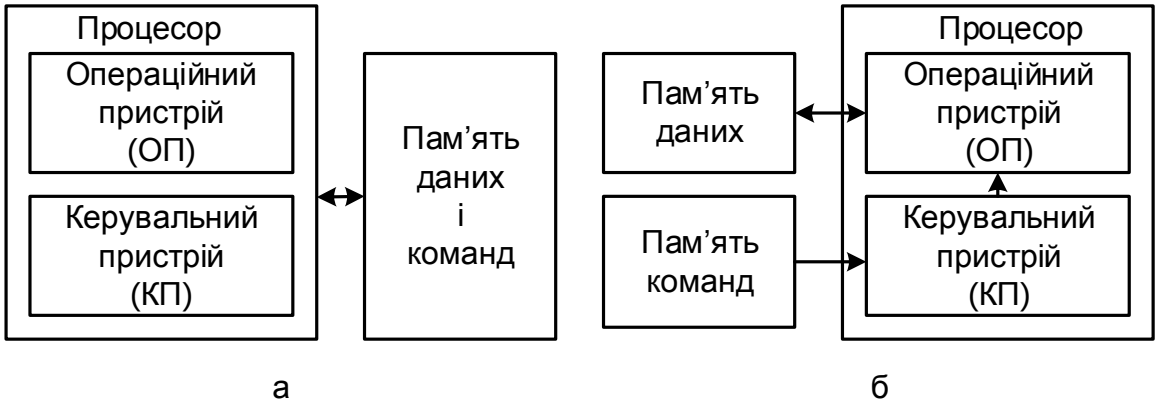


Рис. 1.4. Суть концепції архітектури фон Неймана (а)
і гарвардської архітектури (б)

1.6. Структура комп'ютера з архітектурою фон Неймана

Принципи архітектури фон Неймана визначають структуру комп'ютера:

– комп'ютер має єдину (лінійну) пам'ять, у якій адреси змінюються послідовно;

– комп'ютер має один пристрій оброблення даних – процесор;

– процесор виконує команди послідовно.

Сучасні комп'ютери не зовсім відповідають принципам організації комп'ютерів фон Неймана. Це викликано прагненням підвищити продуктивність комп'ютера, для чого в структуру центрального процесора вносять істотні зміни і застосовують новітні методи організації обчислювального процесу в комп'ютері. Наведемо деякі особливості сучасних комп'ютерів, які не вписуються у концепцію архітектури фон Неймана:

– випереджальна вибірка команд з пам'яті;

– виконання команд у довільному порядку (якщо це можливо) для максимального завантаження обробних пристроїв;

– багаторівнева організація пам'яті (використання кеш-пам'яті);

– використання декількох пристроїв оброблення даних;

– використання елементів мультипроцесорного оброблення даних.

Однак зазвичай для користувача ці особливості організації сучасних комп'ютерів є прозорими і впливають скоріше на продуктивність комп'ютерів, ніж на їх програмування.

Запитання до розділу 1

1. Якою є головна риса комп'ютера з архітектурою фон Неймана?
2. Якою є головна риса комп'ютера з гарвардською архітектурою?
3. Назвіть принципи архітектури фон Неймана.
4. Назвіть принципи гарвардської архітектури.
5. У чому полягає відмінність прінстонської архітектури від гарвардської?
6. Що розуміють під архітектурою комп'ютера?
7. Що розуміють під архітектурою будь-якої системи?
8. Виберіть будь-яку технічну систему і опишіть її архітектуру.
9. Назвіть характерні риси комп'ютера з архітектурою фон Неймана.
10. Назвіть характерні риси комп'ютера з гарвардською архітектурою.
11. Що розуміють під структурою процесора?
12. У чому полягає відмінність архітектури від структури процесора?

2. АРХІТЕКТУРА ПІДСИСТЕМИ ПАМ'ЯТІ

Пам'ять – це підсистема комп'ютера, яка призначена для зберігання даних і команд. Хоча комп'ютер може обробляти дані, які зберігаються на зовнішніх носіях або надходять з пристроїв введення, процесор виконує тільки ті команди і обробляє тільки ті дані, які розташовані в основній пам'яті комп'ютера. Тому для виконання програми вона має бути розміщена в основній пам'яті комп'ютера.

Пам'ять являє собою послідовність комірок, які мають однакову довжину (рис. 2.1, а). Усі комірки пронумеровані, і кожна має унікальний номер, який називається *адресою комірки пам'яті*. У кожній комірці пам'яті зберігається двійковий код певної довжини. Кількість біт, які зберігаються в одній комірці пам'яті, називається *організацією пам'яті*. У більшості комп'ютерів одна комірка пам'яті зберігає один байт, який має унікальну адресу. Однак для підвищення швидкодії процесор може оперувати даними, які складаються з декількох байтів, і в такому випадку за одне звернення до пам'яті він може зчитувати або записувати кілька байтів, необхідних для роботи.

Комірки пам'яті нумерують від 0 до N-1. У літературі прийнято зображати пам'ять у вигляді прямокутника, в якому молодші адреси

розташовуються у верхній частині пам'яті, а старші – у нижній. Кажуть, що пам'ять збільшується у напрямку старших адрес. Такий спосіб зображення адрес є особливо наочним при використанні стеку, який у більшості процесорів збільшується у напрямку молодших адрес. Прийнятий порядок зображення гарантує, що останній елемент, завантажений у стек, дійсно розташовується на "вершині стеку".

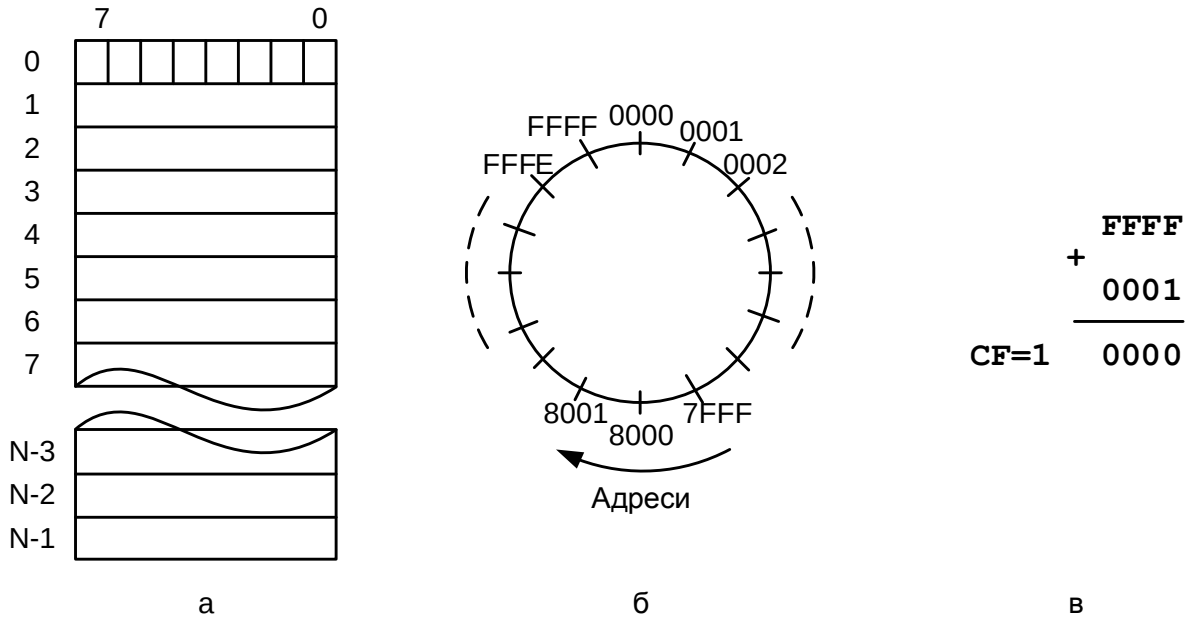


Рис. 2.1. Організація пам’яті: а – адресація пам’яті; б – замкнутість адрес пам’яті; в – арифметична операція, яка пояснює замкнутість адрес пам'яті

Слід зазначити, що хоча на рисунку пам'ять зображена лінійною, її межі замкнуті у кільце (рис. 2.1, б). Ця властивість не є особливістю організації пам'яті, а повністю визначається особливостями арифметики чисел з фіксованою розрядністю. Наприклад, якщо до максимального числа додати 1, то в результаті отримаємо 0 і одиницю перенесення із старшого розряду (CF).

Тому при послідовному переміщенні по комірках пам'яті від менших адрес до великих після комірки з максимальною адресою потрапимо у комірку пам'яті з адресою, яка дорівнює нулю (рис. 2.1, в).

2.1. Характеристики підсистеми пам'яті

Підсистема пам'яті має такі архітектурні характеристики:
– організація пам'яті;
– місткість пам'яті, установленої в комп'ютері;

– тип пам'яті;
– порядок запису байтів у мультибайтових числах;
– розподіл адресного простору процесора (карта пам'яті).

2.1.1. Організація пам'яті

Мінімальною одиницею зберігання інформації є *біт* (bit). Це означає, що будь-яка інформація може бути подана у вигляді цілого числа бітів, яке записується у вигляді послідовності бітів. Не існує можливості подавати інформацію у вигляді дробової величини одного біта, наприклад, 1/2 або 1/3 біта. Один біт позначають 1 b. Не слід плутати один біт (1 біт, 1b) з одним байтом (1 байт, 1 B), який, як відомо, складається з восьми бітів.

Організація пам'яті – це найменша кількість бітів, яку процесор може отримати із пам'яті для подальшого оброблення командою процесора. Організація пам'яті – одна з найважливіших її характеристик. Існують такі види організації пам'яті: байтова, 16-розрядна (словна) і бітова.

Майже всі сучасні комп'ютери мають *байтову організацію* пам'яті (рис. 2.2, б).

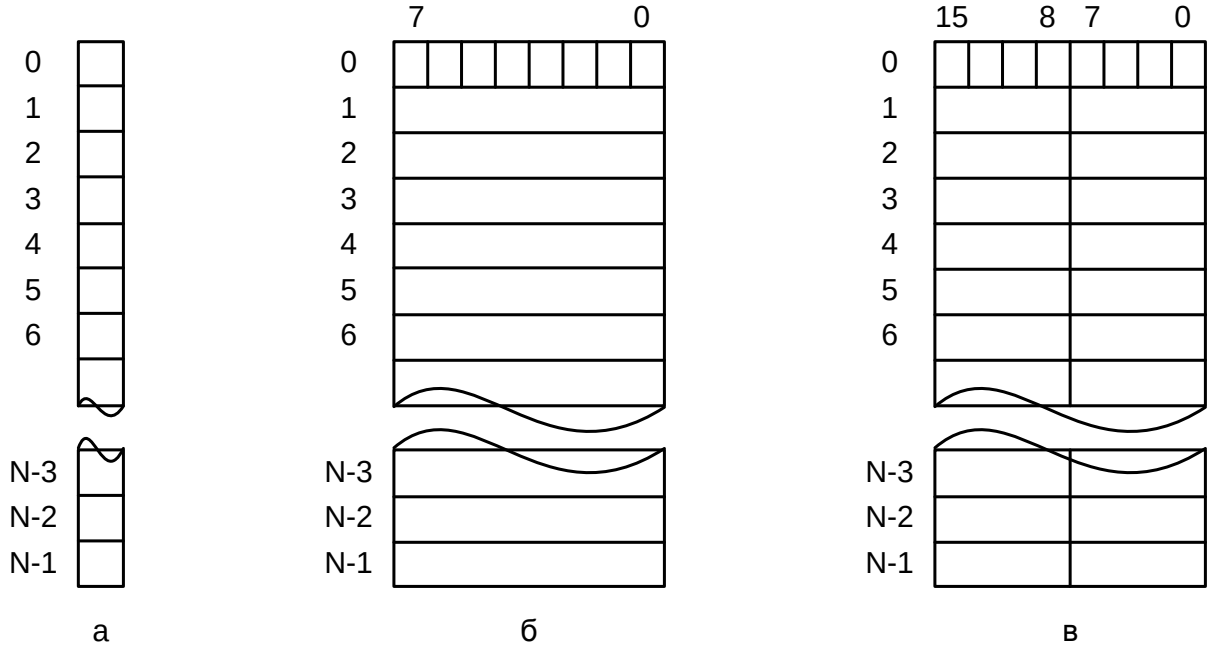


Рис. 2.2. Різні види організації пам'яті: а – бітова; б – байтова; в – 16-розрядна (словна)

Це означає, що за одне звернення до пам'яті не можна отримати *менш ніж один байт* (більше – можна). Якщо за одне звернення до пам'яті можна отримати не менше 16 бітів (одного слова), то говорять, що пам'ять має

16-розрядну, *або словну*, *організацію* (рис. 2.2, в). Така організація пам'яті характерна для сигнальних МП, вироблених фірмами Analog Devices, Motorola і Texas Instrument. Деякі процесори, що виготовляють ці фірми, мають 24-розрядну та навіть 32-розрядну організацію пам'яті. Це сигнальний МП DSP560xx (Motorola), який має 24-бітову організацію пам'яті даних, і сигнальний МП ADSP210xx (Analog Devices), який має 32-розрядну організацію пам'яті даних.

У спеціальних випадках багато сучасних процесорів використовують *бітову організацію пам'яті* (рис. 2.2, а). Бітова організація пам'яті являє собою послідовність нумерованих бітів. Це дає можливість програмісту при виконанні операцій адресувати окремий біт: скидати, встановлювати, інвертувати біт із зазначеним номером, перевіряти або копіювати біт куди-небудь. Бітова організація пам'яті використовується як у потужних МП, наприклад МП Intel x86, так і в однокристальних МК, наприклад Intel MCS-48 і MCS-51.

В однокристальних МК бітова організація пам'яті дає можливість ефективно використовувати пам'ять і зручно реалізувати функції операційної системи. Відомо, що операційна система працює з великою кількістю різноманітних об'єктів. Одним з таких об'єктів є потік. Кожен потік можна подати у вигляді виконуваного коду, який може перебувати в декількох станах: виконання, блокування виконання і завершений стан. Операційна система зберігає інформацію про кожний потік у відповідних таблицях. Якщо для зберігання інформації про кожен стан потоку використовувати один біт, а не байт, то можна отримати значну економію пам'яті.

Пряма адресація бітів полегшує програмну реалізацію алгоритмів шифрування і дешифрування.

В однокристальних МК бітова організація дає змогу зручно оперувати з окремими бітами керувальних регістрів і регістрів процесора, адресувати біти портів введення-виведення, реалізуючи той чи інший протокол обміну, а також виконувати бітове оброблення логічних сигналів.

Адресація окремих бітів пам'яті у командах процесора є зручною можливістю, але на практиці використовується для вирішення обмеженого кола завдань, до яких належать насамперед операції з периферійними пристроями. З огляду на відносно невелику кількість периферійних пристроїв, які можуть бути підключені до МК, немає необхідності забезпечувати адресацію кожного біта всієї пам'яті.

Крім того, оскільки один байт складається з восьми бітів, то для адресації кожного біта всіх комірок пам'яті будуть потрібні довгі команди.

Тому розробники МК обмежують можливість адресації бітів, дозволяючи адресувати окремі біти тільки у деякій кількості елементів

пам'яті. Саме так зробили розробники однокристальних МК Intel MCS-51, в яких для бітової адресації використовується пам'ять з адресами 0x20-0x2F і 0x80-0xFF.

2.1.2. Місткість пам'яті комп'ютера

Місткість пам'яті комп'ютера – це кількість елементів пам'яті, встановленої в даному комп'ютері. Місткість пам'яті з байтовою організацією вимірюється в байтах. Але один байт – занадто мала величина, тому на практиці використовуються більші одиниці вимірювання: кілобайт (KB), мегабайт (MB), гігабайт (GB) і терабайт (TB). Значення цих одиниць виміру пам'яті наведено нижче:

1 KB = 2^{10} = 1 024 B ≈ 10^3 байт;

1 MB = 2^{20} = 1 048 576 B ≈ 10^6 байт;

1 GB = 2^{30} = 1 073 741 824 B ≈ 10^9 байт;

1 TB = 2^{40} = 1 099 511 627 776 B ≈ 10^{12} байт.

Як видно, двійкова приставка "кіло" в одиницях виміру пам'яті відрізняється від десяткової приставки з тією ж назвою. У комп'ютерах використовується двійкова приставка "кіло". Із наведених прикладів видно, що десяткове значення приставки "кіло" становить меншу величину, ніж двійкове значення "кіло".

Це призводить до непорозумінь між розробниками певного обладнання і розробниками операційних систем і комп'ютерів. Так розробники накопичувачів на жорстких магнітних дисках при визначенні характеристик своїх виробів використовують десяткові приставки "кіло", "мега" і т.д. Це призводить до того, що при установленні таких пристроїв у комп'ютер їх місткість (вимірювана у двійкових приставках "кіло") відрізняється від указаної виробником і становить меншу величину, ніж та, яка зазначена у їх характеристиках. Наприклад, накопичувач, який має місткість 80 GB, виробник позиціонує як накопичувач ємністю 80 млрд байтів. При встановленні його у комп'ютер місткість накопичувача буде визначена у двійкових одиницях виміру і становитиме 74.5 GB:

80 GB = 80 000 000 000 B = 80·10^9 B → 80·10^9 / 1073741824 = = 74.5 GB;

500 GB = 500 000 000 000 B = 500·10^9 B → 500·10^9 / 1073741824 = = 465.66 GB.

Місткість пам'яті – важлива, хоч і не визначальна характеристика підсистеми пам'яті. Вона має суттєвий вплив, перш за все, на ціну і швидкодію комп'ютера. Для кожного покоління комп'ютерів існує певне типове значення місткості пам'яті комп'ютера (табл. 2.1).

Таблиця 2.1

Тип комп'ютера	Тип процесора	Місткість пам'яті	Адресний простір	Максимальна адреса
PC	8088	640 KB	1 MB	0xF FFFF
PC/XT	8088	1 MB	1 MB	0xF FFFF
PC/AT	80286	2 MB	16 MB	0xFF FFFF
PC/386	80386	2 - 4 MB	4 GB	0xFFFF FFFF
PC	80486	4 - 8 MB	4 GB	0xFFFF FFFF
PC	Pentium	8 - 16 MB	4 GB	0xFFFF FFFF
PC	Pentium 4	1 - 4 GB	4 GB	0xFFFF FFFF

Місткість пам'яті комп'ютера не перевищує певної величини, яка називається розміром *адресного простору процесора*. Адресний простір – це характеристика процесора, що показує максимальну місткість пам'яті, яка може бути встановлена у комп'ютер з даним процесором. Однак цю характеристику більш детально буде розглянуто пізніше.

2.1.3. Одиниці виміру місткості пам'яті МЕК і IEEE

У березні 1999 року Міжнародна електротехнічна комісія (МЕК) ввела новий стандарт МЕК 60027-2, в якому описано іменування двійкових чисел. Аналогічний стандарт IEEE 1541-2002 було введено в 2008 р.

Приставки МЕК схожі з системою позначень, прийнятою в інформаційних технологіях та індустрії (табл. 2.2): вони починаються на ті самі склади, але другий склад у всіх двійкових префіксах – бі (від англ. binary – «двійковий»). Слід зазначити, що цей міжнародний стандарт у практичній діяльності фактично не використовується, на думку деяких, через їх неблагозвучність: вони вважають, що кілобайт звучить приємніше, ніж Кібібі.

Таблиця 2.2

Значення	Префікс	Скорочення МЕК	Некоректне скорочення
2^{10} = 1024	кібі	Кібіт, КіБ	Кбайт, КВ
2^{20} = 1 048 576	мебі	Мібіт, МіБ	Мбайт, МВ
2^{30} = 1 073 741 824	гібі	Гібіт, ГіБ	Гбайт, GB
2^{40} = 1 099 511 627 776	тебі	Тібіт, ТіБ	Тбайт, ТВ

2.1.4. Більш крупні одиниці виміру місткості пам'яті

Удосконалення технологічних процесів виготовлення напівпровідників привело до істотного зниження вартості зберігання одного біта інформації. З цієї причини вартість напівпровідникової пам'яті і пам'яті на жорстких магнітних дисках постійно знижується, і для користувача стає доступною більша місткість пам'яті для зберігання інформації. Тому слід розглянути більші одиниці виміру місткості пам'яті:

1 петабайт = 1 PB = $2^{50} \approx 10^{15}$ байт;

1 ексабайт = 1 EB = $2^{60} \approx 10^{18}$ байт;

1 зетабайт = 1 ZB = $2^{70} \approx 10^{21}$ байт;

1 йотабайт = 1 YB = $2^{80} \approx 10^{24}$ байт.

2.1.5. Тип пам'яті

Як основна пам'ять в комп'ютерах використовуються різні типи *запам'ятовувальних пристроїв* (ЗП), які можна поділити на три великі групи: оперативні ЗП (ОЗП), постійні ЗП (ПЗП) і перепрограмовані ЗП (ППЗП).

ОЗП, які в зарубіжній літературі називаються RAM (Random Access Memory – пам'ять з довільною вибіркою), використовуються як основна пам'ять комп'ютера для зберігання даних і команд. Ця пам'ять допускає вибірку даних і команд, розташованих за довільною адресою, і виконання операцій читання і запису. Після вимкнення живлення інформація, що знаходиться в ОЗП, втрачається, а при ввімкненні – вміст комірок пам'яті ОЗП встановлюється довільно.

ПЗП, які в зарубіжній літературі іменуються як ROM (Read Only Memory – пам'ять тільки для читання), дають можливість виконувати тільки операцію читання. Вони застосовуються для зберігання системних таблиць, а також важливих системних і керувальних програм. Після вимкнення живлення інформація, що знаходиться в ПЗП, зберігається і відразу після ввімкнення комп'ютера може бути використана без завантаження операційної системи. Як приклад даних, які розміщуються у ПЗП, можна навести таблиці знакогенераторів у принтерах і моніторах. Крім того, у ПЗП зберігаються керувальні програми, такі як BIOS комп'ютера (Basic Input Output System – базова система введення-виведення). У BIOS комп'ютера розміщуються завантажувач операційної системи і примітиви, котрі дають можливість виконувати основні операції введення-виведення, поки не завантажена операційна система.

Більшість сучасних периферійних пристроїв є дуже складними і

підключаються до комп'ютера через адаптери, до складу яких входить спеціалізований процесор. Для керування периферійним пристроєм використовують керувальні програми, які розміщені у ПЗП адаптера і надають користувачеві набір базових функцій введення-виведення, специфічних для даного пристрою. Таке ПЗП разом з керувальними програмами називається BIOS адаптера. Свій BIOS мають відеоадаптер, звукова та мережева карти, модем та інші контролери.

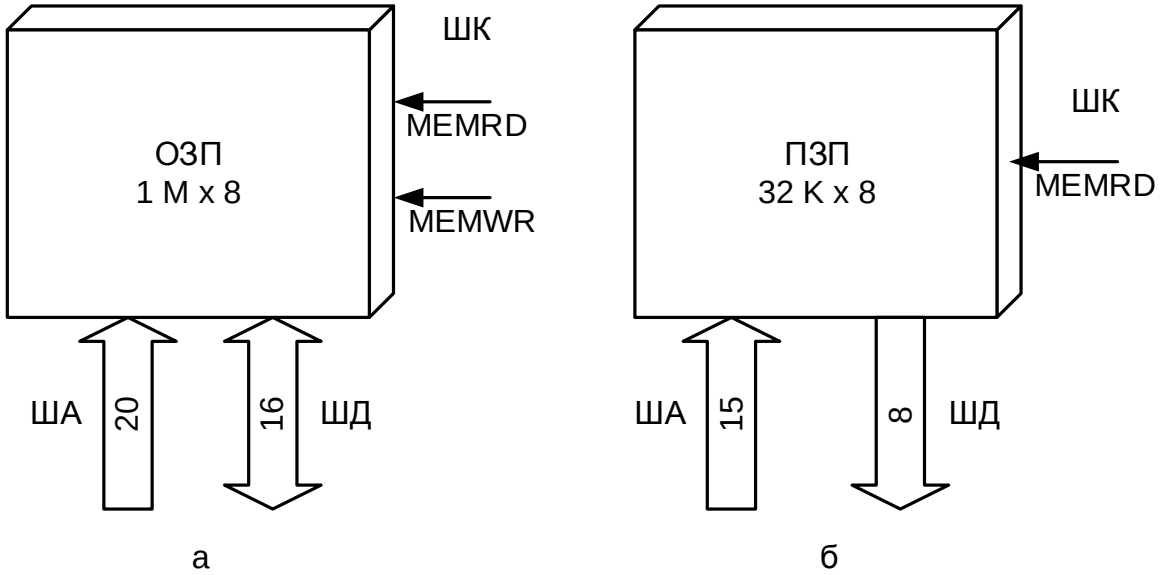
ППЗП – перепрограмовані ЗП, які називаються EPROM (Erasable Programmable ROM – перепрограмовані ПЗП), у звичайному стані функціонують як ПЗП, тобто зберігають записану в них інформацію, і допускають тільки операцію читання. Але в певному режимі – режимі програмування – вміст ППЗП може бути стерто і записано по-новому. Зазвичай для програмування необхідно витягти ПЗП із робочої плати, вставити в спеціальний пристрій, який називається програматором, і запустити спеціальну програму, що записує дані в ППЗП. Іноді перепрограмування ПЗП можна виконати і без вилучення його з пристрою. У цей час поширені два типи ППЗП: з ультрафіолетовим стиранням інформації (УСППЗП) і електрично програмовані (ЕППЗП).

Для перепрограмування ППЗП з ультрафіолетовим стиранням необхідно спочатку стерти попередні дані шляхом опромінення кристала мікросхеми за допомогою ультрафіолетової лампи протягом певного часу, а вже потім вставити мікросхему в програматор і запрограмувати її, запустивши спеціальну програму. Програмування ЕППЗП можна виконати, не виймаючи його з плати, але потрібно використовувати спеціальний режим програмування і відповідну програму. Крім того, на платі, у якій встановлено ЕППЗП, має бути попередньо встановлено програматор. Слід зазначити, що ППЗП допускає значну, але все ж обмежену кількість циклів перепрограмування.

Позначення підсистеми пам'яті на функціональних схемах наведено на рис. 2.3. Підсистема пам'яті у вигляді ОЗП або ПЗП підключається до системної магістралі за допомогою трьох шин: шини адреси (ША), шини даних (ШД) і шини керування (ШК). По шині адреси передається адреса комірки пам'яті, по шині даних – дані в пам'ять або з неї, а по шині керування – сигнали, які ініціюють процес запису або читання даних у модулі пам'яті. На ОЗП подаються два керувальних сигнали – сигнал читання MEMRD і сигнал запису MEMWR, оскільки допустимі обидві операції (рис. 2.3, а). На ПЗП досить подати один сигнал – сигнал читання MEMRD, тому що можна виконувати тільки читання даних (рис. 2.3, б).

На рис. 2.3, а видно, що модуль ОЗП має байтову організацію пам'яті, її місткість становить 1 МВ, а для передачі даних використовується 16-розрядна шина. Для цього процесор спочатку виконує читання байта, який

передається по восьми молодших лініях ШД, а потім – наступного байта, який передається по старших лініях ШД. Для користувача така процедура читання є абсолютно прозорою, оскільки являє собою читання 16-розрядного числа за один цикл звернення до пам'яті.



а б

Рис. 2.3. Позначення модулів пам'яті на функціональних схемах:
а – ОЗП; б – ПЗП

2.1.6. Порядок запису байтів мультибайтових чисел

У загальному випадку, коли потрібно компактно записати число, яке більше 255, використовують кілька байтів. Така декомпозиція числа у добуток інших множників, які після перемноження дадуть вхідне число, називається факторизацією, або розкладанням на множники.

Число А, що складається з n байтів, факторизується по основі 256 у такий спосіб:

$$M = \sum_{k=0}^{n-1} A_i \cdot 256^i = A_{n-1} \cdot 256^{n-1} + \ldots + A_1 \cdot 256^1 + A_0 \cdot 256^0 .$$

Набір чисел A_{n-1}, …, A_1, A_0 і є послідовністю байтів для запису числа А. Існує три порядки запису байтів:

1. *Від молодшого до старшого* (від англ. *little-endian* – "гострокінцевий") – A_0, A_1,…, A_{n-1} (запис починається з молодшого байта і закінчується старшим). Цей порядок запису байтів вперше застосувала компанія Intel для зберігання байтів у пам'яті персональних комп'ютерів з МП x86, у зв'язку з чим перевернутий спосіб іноді називають "*інтеловський порядок запису байтів*".

2. *Від старшого до молодшого* (від англ. *big-endian* – "тупокінцевий") – $A_{n-1},\ldots, A_1, A_0$ (запис починається зі старшого байта і закінчується молодшим). Цей порядок запису байтів є стандартним для протоколів TCP / IP. Він використовується у заголовках пакетів даних і в багатьох протоколах більш високого рівня, розроблених для використання поверх TCP / IP. Тому його називають мережевим (англ. *network byte order*). Він використовується процесорами IBM 360/370/390, Motorola 68000, SPARC, Texas Instruments 9900 (звідси третя назва – "*порядок запису байтів Motorola*, *Motorola byte order*"). Цей порядок запису байтів застосовується в багатьох форматах файлів (наприклад, PNG).

Багато процесорів можуть використовувати обидва порядки запису байтів – або один, або другий, наприклад: ARM, PowerPC, DEC Alpha, MIPS, PA-RISC і IA-64. Зазвичай порядок запису байтів вибирається програмно під час ініціалізації операційної системи, але може бути вибраний і апаратно перемичками на материнській платі. В цьому випадку правильніше говорити про порядок запису байтів в операційній системі. Такий порядок запису байтів іноді називають перемикальним (англ. *big-endian*).

3. *Змішаний* (англ. *middle-endian*). Він використовується при роботі з числами, довжина яких перевищує машинне слово. В цьому випадку виконується поділ числа на машинні слова, які записуються у форматі, природному для даної архітектури, але самі слова записуються в зворотному порядку. Класичний приклад middle-endian – подання 4-байтових цілих чисел на 16-бітових процесорах лінійки PDP-11 (відомий як PDP-endian). Для подання двобайтових значень (слів) використовувався порядок little-endian, але 4-байтове подвійне слово записувалося від старшого слова до молодшого.

2.1.7. Спосіб розміщення в пам'яті мультибайтових чисел

Часто мультибайтові числа, які складаються із більш ніж одного байта, називають *довгими числами*. Сучасні комп'ютери дають змогу обробляти числа великої розрядності, тоді як пам'ять має байтову організацію. Щоб програміст міг правильно звертатися до окремих частин довгих чисел і правильно інтерпретувати отримані результати, необхідно визначити набір правил для розміщення їх у пам'яті та правила адресації всього числа і окремих його частин. Найбільш популярні два способи розміщення і адресації довгих чисел: перевернутий (рис. 2.4, а) і неперевернутий (рис. 2.4, б). Перший спосіб застосовується в МП Intel x86, а другий – в процесорі Motorola 68000.

При використанні *перевернутого* способу розміщення чисел довгі

числа розташовуються в пам'яті побайтно, починаючи з молодшого байта, і адресою всього числа є адреса його молодшого байта. Цей принцип можна сформулювати так: "більш старше – за більш старшою адресою". Для доступу до більш старших фрагментів числа необхідно адресу його молодшого байта послідовно *збільшувати* на одиницю.

При використанні неперевернутого способу розміщення чисел довгі числа розташовуються в пам'яті побайтно, починаючи зі старшого байта, і адресою всього числа є адреса його старшого байта. Цей принцип можна сформулювати так: "більш молодше – за більш молодшою адресою". Для доступу до більш старших фрагментів числа необхідно адресу його молодшого байта послідовно *зменшувати* на одиницю.

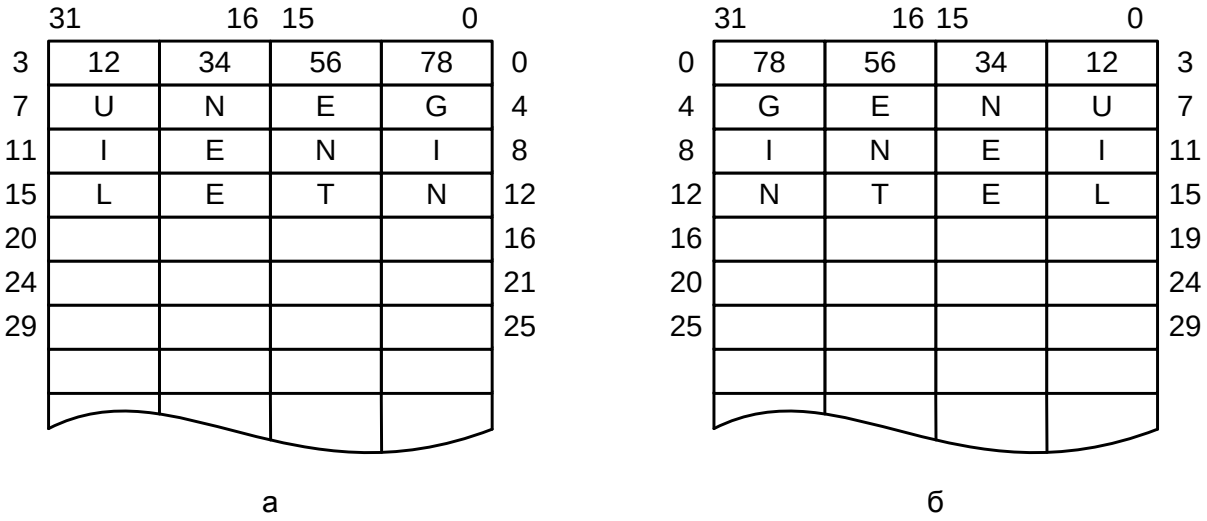


Рис. 2.4. Розміщення 32-розрядних чисел у пам'яті з байтовою організацією: а – перевернутий спосіб; б – неперевернутий спосіб

На рис. 2.4 показано приклад розміщення в пам'яті 32-розрядного числа 0x12345678 і символьного рядка "GENUINEINTEL" з використанням обох способів розміщення. Перевернутий спосіб є більш природним для розміщення в пам'яті чисел і виконання операцій з ними, оскільки арифметичні операції над довгими числами починаються від молодших розрядів до старших.

Іншою перевагою перевернутого способу розміщення чисел є можливість "неявної типізації" цілих чисел при читанні меншого обсягу байтів. Наприклад, якщо в комірці пам'яті міститься число 0x00000022, то, прочитавши його як int16 (два байти), отримаємо число 0x0022, прочитавши один байт, – число 0x22.

Символьні рядки, розміщені в пам'яті перевернутим способом, мають не зовсім природний вигляд. Вони розташовані справа наліво (звідси і назва

способу – "перевернутий"), тому виконання лексикографічних операцій не можна прискорити шляхом одночасного оброблення декількох символів.

Неперевернутий спосіб розміщення чисел у пам'яті є більш зручним для зберігання символьних рядків і ефективної реалізації їх лексикографічного аналізу шляхом оброблення декількох символів однією командою, тому що порівняння рядків виконують починаючи зі старшого символу.

2.1.8. Розподіл адресного простору комп'ютера

Визначення областей адресного простору, який комп'ютер використовує спеціальним чином, називається **розподілом адресного простору комп'ютера**, або **картою пам'яті**.

У загальному випадку всю пам'ять комп'ютера можна використовувати для зберігання будь-яких даних і команд. Однак на практиці не вся пам'ять, встановлена в комп'ютері, доступна для програм і даних користувача. У будь-якому комп'ютері є області пам'яті, які процесор, адаптери периферійних пристроїв або операційна система використовують для своїх цілей або на свій розсуд. Тому необхідно знати розподіл адресного простору комп'ютера для того, щоб:

– використовувати ці фіксовані адреси при програмуванні (наприклад, вектори переривання повинні містити адреси обробників переривань);

– не розміщувати дані і команди у зарезервованих комірках пам'яті, оскільки вони можуть бути змінені не Вашою програмою.

2.1.9. Розподіл адресного простору МП x86

На рис. 2.5 показано розподіл адресного простору для різних моделей МП, які мають архітектуру Intel x86. Як випливає із рисунка, МП Intel x86 в реальному режимі має дві зарезервовані області пам'яті: в одній з них повинен розміщуватися апаратний завантажувач, а в іншій – вектори переривань (див. рис. 2.5). Перший кілобайт адресного простору пам'яті МП x86 використовує для розміщення таблиці векторів переривань у реальному режимі. Всього існує 256 векторів переривань, кожен з яких має довжину 4 байти (256•4 = 1024).

Після апаратного скидання або ввімкнення живлення МП з архітектурою x86 починає виконувати програму, яка має довжину 16 байтів і розташована в останніх комірках їх адресного простору. Адресні простори зазначених процесорів різні (див. табл. 2.1). Тому завантажувач розташовують у кінці першого мегабайта (для МП 8086), у кінці перших 16 мегабайтів (для МП 800286) або в кінці 4-гігабайтового адресного

простору (для МП 80386). Початковий завантажувач розташовують у ПЗП. Такий спосіб його розміщення гарантує, що після скидання процесора і його переходу в захищений режим ПЗП початкового завантаження не опиниться в середині адресного простору процесора. Очевидно, що 16 байтів пам'яті недостатньо для розміщення повноцінного завантажувача, тому, починаючи з адреси FFFF0, розташовують кілька команд переходу, які і викликають первинний завантажувач операційної системи.

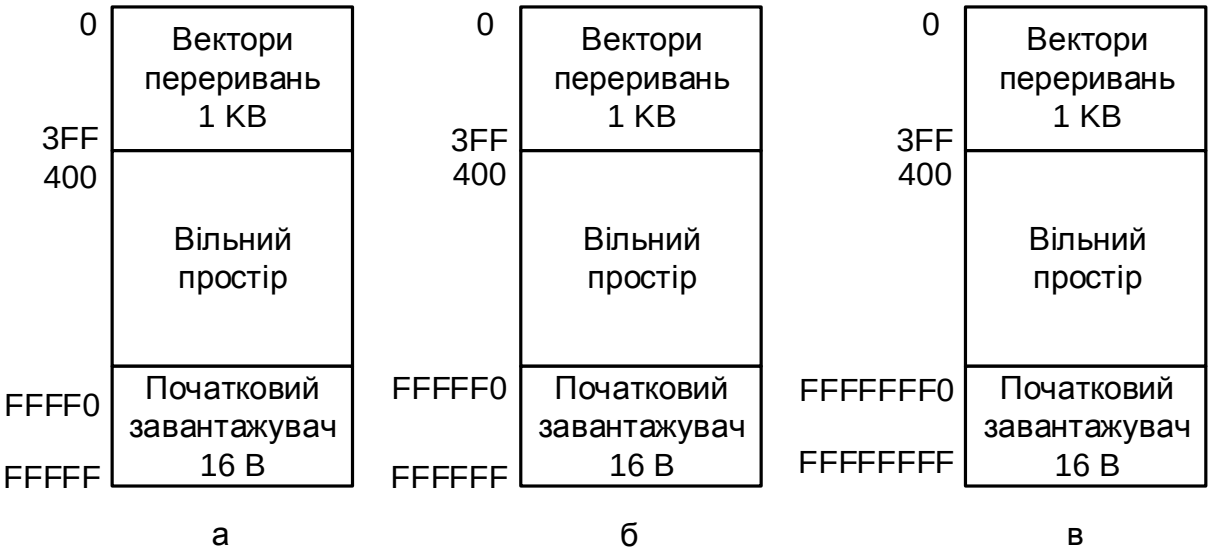


Рис. 2.5. Розподіл адресного простору МП Intel x86:
а – МП 8086; б – МП 80286; в – МП 80386

Слід зазначити, що при проектуванні комп'ютера на адресний простір процесора відображається адресний простір комп'ютера. При цьому, якщо розробники комп'ютера будуть виконувати розроблення недбало, можливі колізії. Так сталося при розробленні ПК IBM PC, в якому адресний простір векторів апаратних переривань наклався на адресний простір виключень (внутрішніх переривань при виникненні помилки).

Для оброблення виключень процесора призначені фіксовані номери переривань у діапазоні від 0 до 31. Процесор 8086 обробляє всього п'ять виключень, а процесор 80486 – значно більше і т.д. Передача керування потрібному оброблювачу виключення здійснюється через таблицю векторів переривань, розташовану на початку адресного простору процесора, – з адреси 0 до адреси 1023. Усього існує 256 типів переривань (INT0 – INT255). Кожен вектор переривання займає 4 байти (рис. 2.6).

Комп'ютер IBM PC має 16 апаратних переривань IRQ0 – IRQ15, вектори яких розташовані на місці переривань INT8 – INT15. Таким чином, апаратні переривання і частина внутрішніх переривань перекриваються. Більш того, через переривання $INT10_{16}$ та інші викликають певні функції BIOS. У даному

випадку знову має місце перекриття адресного простору. Це є недоліком, оскільки BIOS важко обробляти відповідні переривання через те, що деякі із них можуть викликатися через внутрішні або апаратні переривання.

При розробленні ПК IBM PC в кінці першого мегабайта пам'яті був розміщений BIOS місткістю 64 KB (рис. 2.6, а). Таким чином, початковий завантажувач МП x86 опинився в адресному просторі BIOS. А оскільки для переходу на первинний завантажувач достатньо всього кілька байтів, то інші комірки початкового завантажувача використовують для зберігання деякої інформації BIOS (рис. 2.6, б, в).

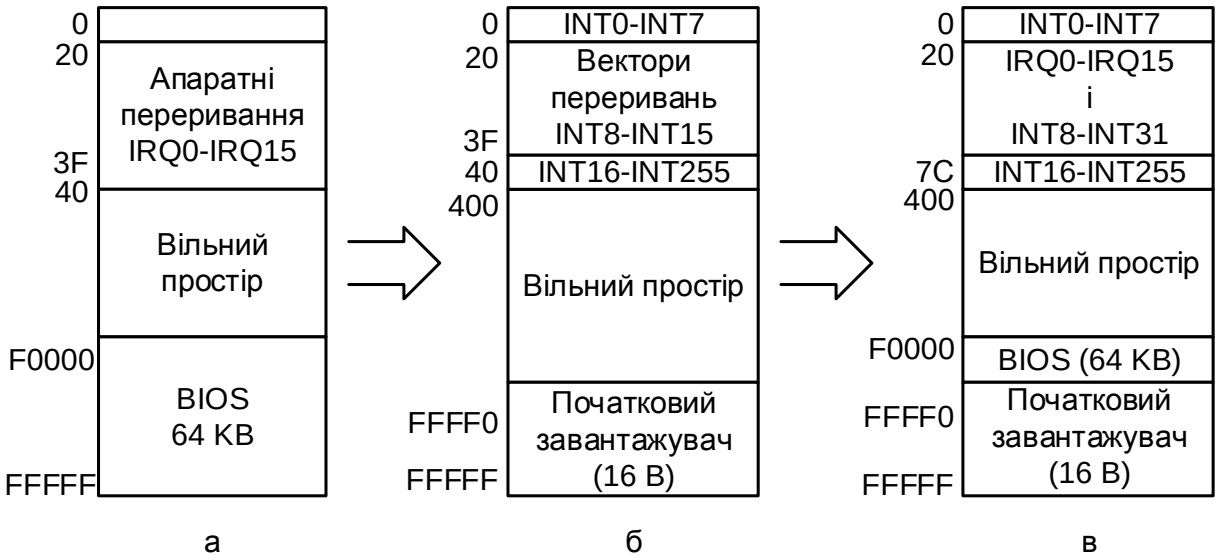


Рис. 2.6. Формування адресного простору комп'ютера IBM PC відображенням на адресний простір процесора

Частина адресного простору процесора розробники ПК резервують для обслуговування адаптерів периферійних пристроїв і операційної системи. Таким чином, у розпорядженні користувача залишається менша місткість пам'яті, ніж установлена в комп'ютері.

2.1.10. Розподіл адресного простору МК MCS-51

Однокристальний МК MCS-51 має три типи пам'яті: внутрішню пам'ять даних, внутрішню пам'ять команд, а також зовнішню пам'ять даних і команд (рис. 2.7). Тому користувач повинен знати розподіл адресного простору для кожного типу пам'яті.

Найбільш фрагментована внутрішня пам'ять даних, яка має місткість 256 байтів (рис. 2.7, а). Перші 32 байти внутрішнього ОЗП займають чотири банки регістрів загального призначення (РЗП). Кожен банк складається з восьми регістрів, а перший банк РЗП за замовчуванням використовується

як стек. 16 байтів адресного простору, що йдуть після банків внутрішнього ОЗП (адреси 0x20 – 0x2F), утворюють область комірок пам'яті, до яких можна застосовувати побітову адресацію. У МК є велика кількість інструкцій, що дають можливість працювати з окремими бітами. Дані, розташовані у цій області пам'яті, можна адресувати як байти з адресами від 0x20 до 0x2F (всього 16 байтів) або як біти з адресами бітів від 0 до 127 (всього 128 бітів). Друга половина адресного простору, розташована за адресами 0x80 – 0xFF, використовується для розміщення регістрів спеціальних функцій, до яких належать як регістри процесора (регістри ACC, B, PSW, IP), так і регістри пристроїв введення-виведення (регістри IE, P1, P2 , P3, SCON, TCON). Таким чином, для розміщення даних можна без побоювань використовувати область пам'яті з адресами 0x30 – 0x7F.

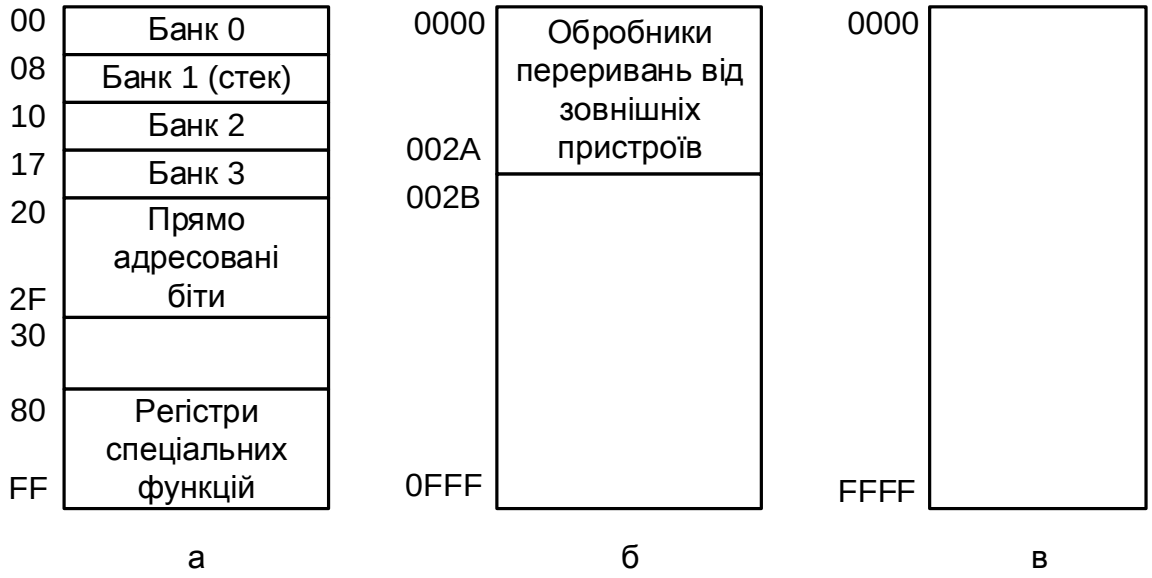


Рис. 2.7. Розподіл адресного простору МК MCS-51:
а – внутрішня пам'ять даних; б – внутрішня пам'ять команд;
в – зовнішня пам'ять даних і команд

Внутрішня пам'ять команд має адресний простір 4 KB (рис. 2.7, б). Для спеціального використання зарезервовано 44 комірки пам'яті на початку адресного простору команд з адресами 0 – 0x2A. У цьому адресному просторі повинні розміщуватися початковий завантажувач (4 байти) і п'ять обробників переривань від периферійних пристроїв (по вісім байтів кожний). Іншу частину пам'яті команд розробники можуть використовувати на свій розсуд.

Зовнішня пам'ять даних і команд є факультативною (рис. 2.7, в), і тому ніякі їхні області процесор спеціальним чином не використовує (рис. 2.8).

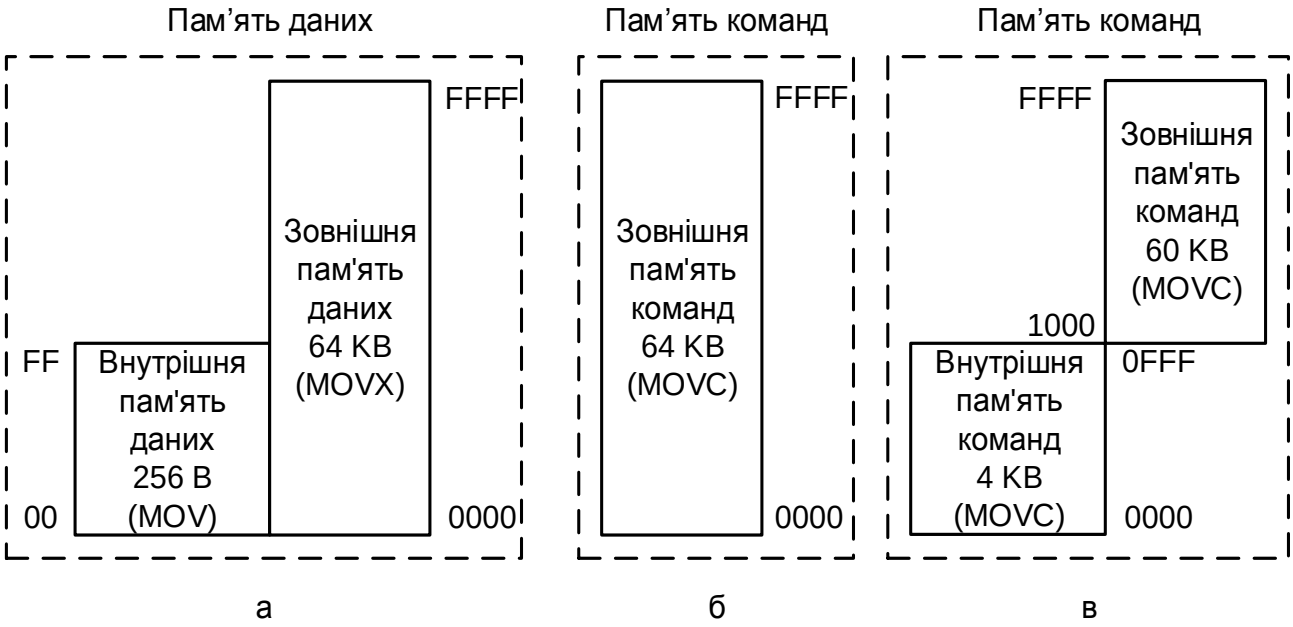


Рис. 2.8. Розподіл адресного простору МК MCS-51:
а – додатково встановлено зовнішню пам'ять даних; б – встановлено тільки зовнішню пам'ять команд, а внутрішньої пам'яті команд немає;
в – є внутрішня і зовнішня пам'ять команд

Залежно від конфігурації МК можливі різні варіанти підключення і використання пам'яті (див. рис. 2.8). Внутрішня пам'ять даних завжди є у МП, її адресний простір починається з нульової адреси. До даних, які там розташовані, слід звертатися за допомогою команди MOV (рис. 2.8, а). Якщо зовнішня пам'ять є у мікропроцесорній системі, то її адресний простір також починається з нульової адреси, але звертатися до даних, розташованих у цій пам'яті, потрібно за допомогою команди MOVX.

Підключення зовнішньої пам'яті команд можливе у двох конфігураціях: при підключеній внутрішній пам'яті команд і відключеній. В обох випадках звернення до даних, розташованих у пам'яті команд, виконується за допомогою команди MOVC, проте розподіл адресного простору відрізняється. Якщо внутрішня пам'ять заблокована, то адресний простір починається з нульової адреси і продовжується до максимально можливої (рис. 2.8, б). Якщо ж внутрішня пам'ять команд підключена, то починаючи з нульової адреси розташовується адресний простір внутрішньої пам'яті команд, а адресний простір зовнішньої пам'яті продовжується там, де закінчується адресний простір внутрішньої пам'яті (рис. 2.8, в). Це дуже важлива обставина, розуміння якої дасть змогу уникнути багатьох помилок при програмуванні МК.

2.2. Моделювання пам'яті

Моделювання основної пам'яті комп'ютера можна виконати на іншому, більш потужному комп'ютері, використовуючи програму мовою С. Основна пам'ять модельованого комп'ютера являє собою масив, тому її можна визначити як масив, використовуючи відповідний опис мовою С.

```
// Програма моделює роботу пам'яті з байтовою або словною
// організацією
// Визначити адресний простір для різних типів процесорів
const unsigned int SIZE_PDP11 =      65536; // 64 KB
const unsigned int SIZE_8080  =      65536; // 64 KB
const unsigned int SIZE_8086  =    1048576; //  1 MB
const unsigned int SIZE_80286 =   16777216; // 16 MB
const unsigned long SIZE_x86  = 2147483647; //  2 GB
// Визначити псевдоніми типів даних для роботи із пам'яттю
typedef unsigned int    ADDRESS; // адреса комірки пам'яті
typedef unsigned char   BYTE;   // байт
typedef unsigned short  WORD;  // слово

int main()
{
  ADDRESS addr;
  BYTE data8;
  WORD data16;
  // Моделюємо пам'ять МП 8080
  // Організація пам'яті - байтова, місткість пам'яті - 64 KB
  {
    BYTE RAM_8080[SIZE_8080];
    // Читати із пам'яті байт за адресою 1000
    data8 = RAM_8080[1000];
    // Записати в пам'ять байт за адресою 2000
    data8 = 0x55;
    RAM_8080[2000] = data8;
  }
  // Моделюємо пам'ять комп'ютера PDP-11
  // Організація пам'яті - словна, місткість пам'яті - 64 KB
  {
    WORD RAM_PDP11[SIZE_PDP11];
    // Читати із пам'яті слово за адресою 10 000
    data16 = RAM_PDP11[10000];
    // Записати в пам'ять слово за адресою 20 000
    data16 = 0xAAAA;
    RAM_PDP11[20000] = data16;
  }
  // Моделюємо пам'ять МП з архітектурою x86
```

```
  // Організація пам'яті - байтова, місткість пам'яті - 2 GB
  {
    BYTE RAM_x86[SIZE_x86];
  // Читати із пам'яті байт за адресою 10 000 000
    data8 = RAM_x86[10000000];
  // Записати в пам'ять байт за адресою 20 000 000
    data8 = 0xFF;
    RAM_x86[20000000] = data8;
  }
}
```

На початку програми оголошено константи, які задають розмір адресного простору для процесорів різних типів. Після цього визначено псевдоніми наявних типів даних, що дає змогу при моделюванні пам'яті звертатися до змінних природним чином. Далі йдуть три блоки, всередині кожного з яких моделюється пам'ять для трьох процесорів: МП 8080, комп'ютера PDP-11 і лінійки МП x86. Основна пам'ять подана як масив байтів або слів. Оскільки для МП 8080 і x86 пам'ять має байтову організацію, то у програмі пам'ять являє собою масив байтів. Для комп'ютера PDP-11 пам'ять має словну організацію, тому в програмі пам'ять являє собою масив слів. Далі для кожного типу пам'яті виконано операції запису даних у пам'ять і читання їх із пам'яті.

Запитання до розділу 2

1. Які функції виконує підсистема пам'яті?
2. Укажіть архітектурні характеристики пам'яті?
3. Що розуміють під організацією пам'яті?
4. Які одиниці виміру місткості пам'яті застосовують?
5. Чим відрізняються одиниці виміру пам'яті 1b і 1B?
6. Якого роду інформація зберігається в пам'яті?
7. Які типи пам'яті існують?
8. Що показує місткість пам'яті?
9. Навіщо програмісту знати розподіл адресного простору процесора або комп'ютера?
10. Навіщо розробнику апаратного забезпечення знати розподіл адресного простору процесора або комп'ютера?
11. Яка характеристика задає порядок розташування байтів у пам'яті?
12. В якому порядку розташовуються в пам'яті байти числа 0xABCD1234?
13. Виконайте моделювання пам'яті відомою Вам мовою програмування.

3. АРХІТЕКТУРА ЦЕНТРАЛЬНИХ ПРОЦЕСОРІВ

Архітектура процесора – це набір його властивостей і компонентів, доступних програмісту. До архітектури процесора входять:

– регістри і їх функціональне призначення;

– система команд процесора.

Програмна модель процесора – це кількість регістрів, їх іменування та функціональне призначення.

Центральний процесор – підсистема комп'ютера, призначена для виконання операцій з числами і керування іншими підсистемами комп'ютера. *Центральний процесор* – основна підсистема комп'ютера, його мозок. Оскільки головними функціями процесора є оброблення і керування, то до його складу входять операційний пристрій (ОП) і керувальний пристрій (КП). Іноді ОП називають операційним автоматом (ОА), а КП – керувальним автоматом (КА) (рис. 3.1).

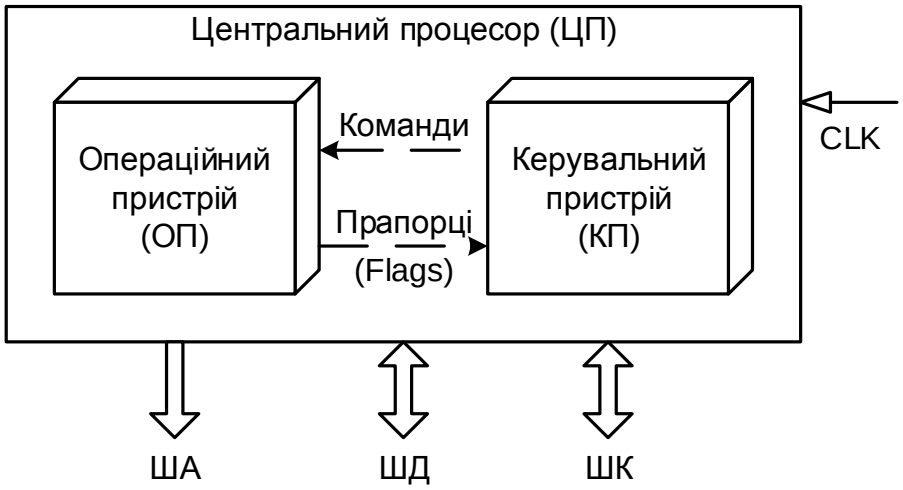


Рис. 3.1. Склад типового процесора

Операційний пристрій – підсистема процесора, призначена для оброблення даних. Він виконує набір базових арифметичних і логічних операцій. До простих базових арифметичних операцій належать операції додавання, віднімання, порівняння, збільшення і зменшення на одиницю та деякі інші. Базові логічні операції містять операції НІ, І, АБО, а також виключне АБО. Пристрій процесора, який виконує найпростіші арифметичні й логічні операції, називається *арифметико-логічним пристроєм* (АЛП). Більш складні арифметичні операції – множення і ділення – до базових операцій не належать і реалізуються за допомогою вбудованого програмного або апаратного забезпечення всередині ОП.

Керувальний пристрій – підсистема комп'ютера, яка виробляє задану *послідовність* керувальних сигналів, які подаються на різні функціональні вузли комп'ютера для правильного виконання кожної команди ЦП. Керувальні послідовності сигналів унікальні для кожної команди. Вони виробляються не в довільні моменти часу, а у певні моменти, які визначаються сигналами тактової частоти CLK тактового генератора. Чим вище частота тактового генератора, тим швидше виконуються команди процесора. Пристрій керування, мабуть, найбільш складна частина процесора.

Операційний і керувальний пристрої взаємодіють, обмінюючись необхідною інформацією: ОП передає ознаки результату (прапорці) в КП, а від нього отримує код виконуваної операції. Обмін даними між ОП і КП на рис. 3.1 показано пунктирними лініями.

Процесор підключається до системної магістралі за допомогою трьох шин: шини адреси (ША), шини даних (ШД) і шини керування (ШК). По односпрямованій ША процесор видає адреси комірок пам'яті і портів введення-виведення. По двобічній шині відбувається обмін даними між процесором, з одного боку, і підсистемою пам'яті або підсистемою введення-виведення – з іншого. ШК являє собою набір ліній, по яких передаються сигнали до підсистем пам'яті і введення-виведення. Мінімальний набір сигналів, які складають цю шину, такі: читання і запис у пам'ять (MEMRD і MEMWR), читання і запис у порти введення-виведення (I / ORD і I / OWR), запит на переривання (INTR) і підтвердження переривання (INTA). Деякі сигнали, наприклад MEMRD і MEMWR, вже розглядалися в підсистемах пам'яті, з іншими – ознайомимося пізніше.

Для підвищення продуктивності процесора його поділяють на функціональні пристрої, які паралельно працюють. Так, МП 80286 поділений на чотири пристрої, які можуть працювати незалежно один від одного. *Шинний пристрій* послідовно зчитує байти команд і поміщає їх у чергу команд. *Пристрій керування* витягує команди із черги команд, декодує їх у внутрішню форму і поміщає їх ще в одну чергу. *Операційний пристрій* виконує внутрішні команди. *Адресний пристрій* перетворює логічні адреси на фізичні, які надходять у шинний пристрій для читання із пам'яті або запису в пам'ять (рис. 3.2) [3].

3.1. Класифікація процесорів

Існує кілька способів класифікації МП. Усі МП можна поділити на універсальні й спеціалізовані. Крім того, МП можна поділити на три великі групи: однокристальні МП, однокристальні МК і секціоновані МП.

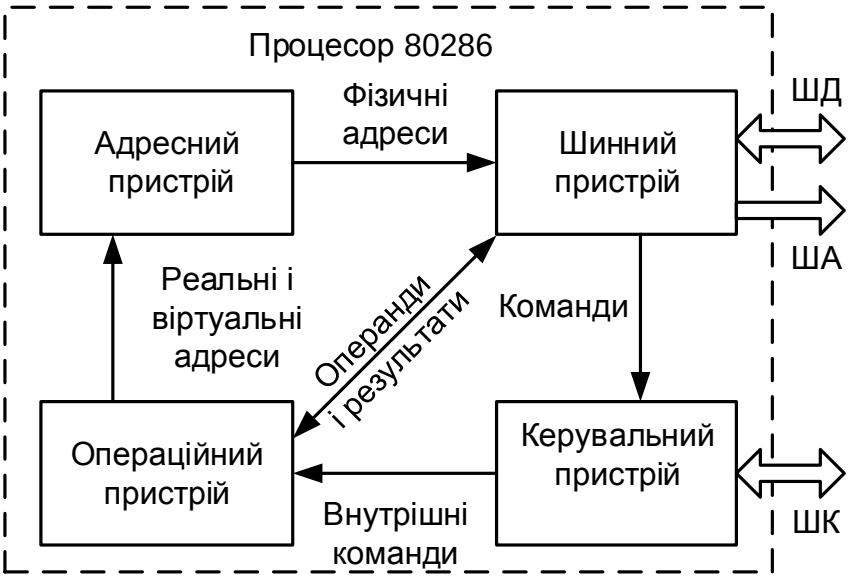


Рис. 3.2. Функціональна схема МП 80286

Універсальними називаються МП, які здатні вирішувати широке коло завдань (обчислювальні, керувальні, які моделюють), обробляти великі масиви нечислових даних тощо. Усі МП, які встановлюють у персональні комп'ютери, робочі станції і сервери, є універсальними. Приклади процесорів наведено в табл. 3.1 [2].

Таблиця 3.1

Фірма-розробник	Моделі МП
Intel	Pentium, Celeron, Core (i3, i5, i7, i9), Xeon
AMD	Sempron, Athlon, Phenom, Ryzen (3, 5, 7, 9)
Sun Microsystems	Ultra-SPARC
Hewlett Packard	PA-8000
IBM, Apple, Motorola	PowerPC
MIPS	R10000

Спеціалізовані МП призначені для вирішення певного класу завдань, тому є дешевшими порівняно з універсальними МП і мають більш високу продуктивність для даного класу завдань. До таких МП належать сигнальні, мультимедійні, графічні процесори, а також процесори введення-виведення (SCSI- або RAID-контролери). Сигнальні МП призначені для цифрового оброблення сигналів у реальному часі. Приклади таких МП наведено в табл. 3.2. Графічні процесори призначені для обчислень тривимірної графіки в реальному часі й виведення на екран дисплея тривимірних зображень.

Однокристальний МП – процесор, який виконано у вигляді однієї мікросхеми і має *фіксовану* розрядність, адресний простір і систему команд. Одного МП не достатньо для побудови комп'ютера, потрібні додаткові мікросхеми пам'яті і периферійних адаптерів для підключення зовнішніх пристроїв. Сучасні технології дають змогу більшість стандартних контролерів розмістити на кристалі однієї мікросхеми, яка називається чіпсетом (chipset). Таким чином, *чіпсет* – спеціалізований МП, який виконує функції введення-виведення даних зі стандартних пристроїв, а також реалізує функції керування пам'яттю.

Конструктивно МП, чіпсет, модулі пам'яті і додаткові периферійні адаптери розміщуються на *материнській платі*, яка інтегрує у собі по суті весь персональний комп'ютер без периферійних пристроїв.

Таблиця 3.2

Фірма-розробник	Моделі МП
Analog Devices	ADSP21xx, ADSP210xx
Motorola	DSP560xx, DSP563xx
Texas Instrument	TMS320C1x, TMS320C2xx, TMS320C5xx

Однокристальний МК – це комп'ютер, виконаний на одному кристалі. МК містить всі компоненти, що є в комп'ютері: процесор, пам'ять даних (ОЗП), пам'ять команд (ПЗП), основні периферійні адаптери (таймери, паралельний і послідовні адаптери, контролер переривання і т.ін.). Однокристальні МК зазвичай мають гарвардську архітектуру, невисоку продуктивність, низьке енергоспоживання і є значно дешевшими, ніж універсальні МП. Побудова обчислювального пристрою на їх основі потребує мінімальної кількості зовнішніх схем, тому такий пристрій має невисоку вартість. Невисока продуктивність МК порівняно з універсальними МП обумовлена тим, що для розміщення великої кількості функціональних вузлів на одному кристалі мікросхеми потрібне застосування технологій з високим ступенем інтеграції. Такі технології зазвичай мають невисоку швидкодію і низьке енергоспоживання. Застосовувані при їх виготовленні технології не є лідерами за швидкодією. Область застосування МК обумовлена їх дешевизною, невеликим енергоспоживанням і мініатюрними розмірами. Це вбудована побутова техніка, бортові комп'ютери в автомобілях та інші області, які вирішують завдання керування і не виконують обчислень у великих обсягах.

Розглянуті групи МП мають одну загальну властивість – незмінність. Вони мають фіксовані розрядність, адресний простір, систему команд. Для

зміни системи команд, збільшення розрядності або адресного простору слід використовувати спеціальну програму або вибрати інший процесор. Використання секціонованих МП дає можливість вирішити зазначені проблеми. *Секціоновані МП* – це набори ВІС, які містять функціональні вузли певної розрядності і мають засоби для їх об'єднання з метою побудови процесора із заданою розрядністю та іншими користувацькими характеристиками. Система команд проектованого пристрою, режими адресації та інші можливості реалізуються за допомогою набору мікропрограм, які знаходяться у ПЗП. Затримки, пов'язані з передачею інформації між секціями, не дали змоги реалізувати високопродуктивні системи на базі секціонованих МП, що призвело до зникнення цього класу компонентів.

Прикладом таких пристроїв були секціоновані МП 2900, вироблені фірмою AMD.

3.2. Структура найпростішого процесора

Найпростіший процесор складається з операційного пристрою і пристрою керування. Іноді в окремий вузол виділяють шинний пристрій. До складу процесора входять вузли, які здійснюють вибірку і виконують команди. Структурна схема, наведена на рис. 3.3, дає можливість скласти досить точне уявлення про структуру та принципи функціонування процесорів.

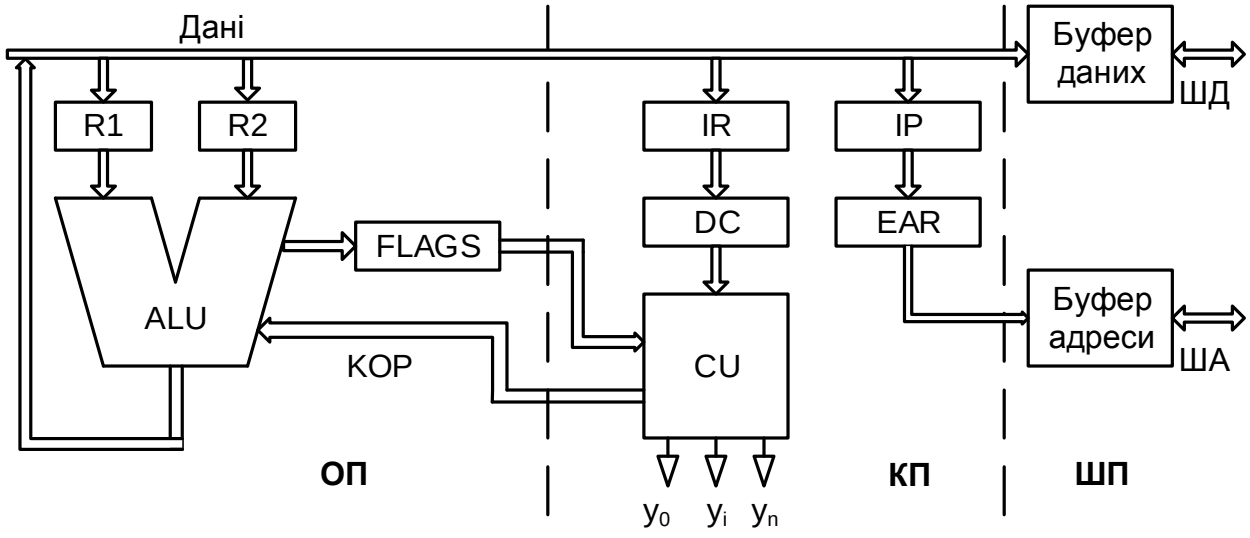


Рис. 3.3. Структурна схема процесора

Процесор складається з двох підсистем: операційного пристрою (ОП) і

керувального пристрою (КП). ОП виконує арифметичні й логічні операції над даними. Він складається з арифметико-логічного пристрою (АЛП, ALU), буферних регістрів R1 і R2, а також регістра прапорців FLAGS. Основний вузол ОП – АЛП, який виконує найпростіші арифметичні й логічні операції над тими даними, що надходять на два його входи. АЛП – комбінаційна схема, тому при змінюванні даних на входах дані на виході також будуть змінюватися. Якщо зафіксувати дані на входах АЛП, то через деякий час на виході буде отримано результат, значення якого не буде змінюватися до наступної зміни вхідних даних. Роль фіксаторів відіграють регістри R1 і R2. Розрядність цих регістрів і АЛП визначається розрядністю процесора. Характер операції, що виконує АЛП, залежить від команди, яка у вигляді коду операції (КОП, КОР) надходить з КП.

Крім результату операції АЛП формує його *ознаки* – бітові прапорці, що характеризують деякі суттєві властивості результату. Прапорці позначаються однією або двома буквами, які відображають суть даної властивості. У МП Intel x86 другою буквою ознаки результату є F – Flag. Наприклад, прапорець нульового результату називається ZF (Zero Flag), прапорець перенесення – CF (Carry Flag), прапорець знака – SF (Sign Flag), а прапорець переповнення – OF (Overflow Flag).

Керувальний пристрій призначений для вибірки команд із пам'яті, їх декодування і видачі керувальних сигналів на ОП та інші підсистеми комп'ютера. До складу КП входять такі вузли: регістр команд (IR – Instruction Register), регістр адреси (EAR – Executive Address Register), покажчик команд (IP – Instruction Pointer), дешифратор команди (DC – Decoder) і безпосередньо керувальний пристрій КП.

Покажчик команд IP – регістр, який адресує (зберігає адресу) *наступну* команду, яка підлягає виконанню. Як тільки із пам'яті вибрана поточна команда, IP необхідно перевести на наступну команду.

Регістр адреси EAR призначений для адресації байта, який необхідно прочитати або записати в пам'ять. Всі звернення до пам'яті процесор виконує через цей регістр. Для обчислення адреси операнда або команди, заданих у вигляді складного режиму адресації, можуть знадобитися додаткові обчислення. Розрядність регістрів EAR визначається розрядністю шини адреси.

Регістр команд IR призначений для зберігання першого байта команди, яка містить код операції. Регістр команд зберігає КОП у незмінному вигляді протягом всього періоду виконання команди.

З усіх розглянутих регістрів тільки покажчик команд IP і регістр прапорів FLAGS є програмно-доступними, тобто такими, які програміст може змінювати або читати за допомогою команд процесора. З цієї причини регістри IP і FLAGS є частиною архітектури процесора. Решта регістрів

забезпечують функціонування процесора і тому входять до його структури (рис. 3.4).

Рис. 3.4. Архітектура найпростішого процесора

Дешифратор команди DC – складна комбінаційна схема, на вхід якої надходить КОП із регістра команд. Завдання дешифратора команд – розшифрувати КОП, тобто виділити з неї корисну інформацію про те, яку операцію слід виконувати і де слід шукати операнди, необхідні для виконання команди. Частина декодованої команди КОП надходить на АЛП для задання арифметичної або логічної операції, яка буде в ньому виконуватися.

Керувальний пристрій КП використовує декодовану команду для видачі послідовності керувальних сигналів Yi (i = 1 ... n), які надходять на потрібні функціональні вузли для виконання команди.

Іноді у складі процесора виділяють окремий пристрій керування доступом до шині – шинний пристрій (ШП). У найпростішому випадку достатньо двох буферних регістрів – буфера даних і буфера адреси, які відіграють роль підсилювачів. Буфери даних і адреси збільшують навантажувальну здатність шин, що дає змогу підключити до процесора декілька модулів пам'яті і портів введення-виведення.

Функціональне призначення вузлів процесора не дає повного уявлення про його функціонування, для цього необхідно розглянути взаємодію вузлів у процесі виконання команди в динаміці. Але спочатку потрібно ознайомитися з процесом кодування команд процесора.

3.3. Формати машинних команд

Машинна команда – це двійковий код, який сприймається процесором і змушує його виконувати потрібні дії. Усі можливі команди деякого процесора становлять його *систему команд*. Кожен процесор має унікальну, притаманну лише йому, систему команд, і кожна команда кодується двійковим кодом. Для полегшення кодування і декодування команд процесора двійковим кодом окремі біти і бітові групи мають різне функціональне призначення.

Кожна команда у закодованому вигляді зберігає двояку інформацію: про те, яку операцію повинен виконати процесор (що робити) і як визначити місцезнаходження операндів (де знаходяться операнди). Характер

виконуваної операції визначається основною частиною команди, яка називається *кодом операції команди* (рис. 3.5). КОП є обов'язковою частиною будь-якої команди. Зазвичай він має довжину 1 байт (рис. 3.5, а) і крім виконуваної операції може задавати кількість операндів і їх довжину. Як відомо, за допомогою одного байта можна закодувати лише 256 кодів операцій команд. Якщо система команд процесора містить більшу кількість команд, то використовують два байти для КОП. Тоді перший байт КОП містить ознаку розширеного коду операції (наприклад, число 0F у КОП_1), а другий – безпосередньо КОП (у КОП_2) (рис. 3.5, б).

Формат команди – умовне графічне позначення команди, яке відображає функціональне призначення окремих її бітів.

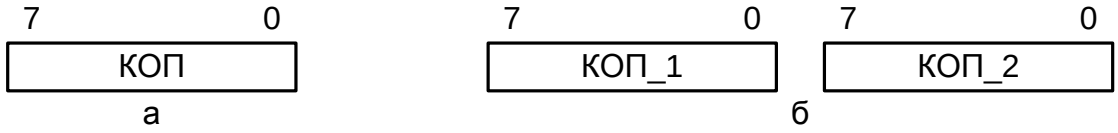


Рис. 3.5. Способи задання коду операції команди: а – звичайний код операції довжиною один байт; б – розширений код операції, що складається з двох байтів

Режим адресації операнда (РА) – спосіб задання операнда, який дає змогу визначити його місце розташування в пам'яті або процесорі. Зазвичай процесори мають кілька режимів адресації. Частина команди, яка задає режим адресації операнда, розташовується після КОП і є необов'язковою. Режим адресації задається за допомогою коду, який називається специфікатором операнда (СО). СО розташовується відразу після КОП і подається в певному форматі (рис. 3.6, в).

Деякі РА потребують для свого задання додаткової інформації у вигляді розширення СО і / або безпосередніх даних, які розташовуються одразу після основного СО (рис. 3.6, г). Для простих режимів адресації КОП визначає не тільки виконувану операцію, але й режим адресації операнда. Описані вище варіанти кодування команд на прикладі форматів деяких команд МП Intel x86 показано на рис. 3.6.

3.4. Основний машинний цикл процесора

Робота процесора полягає у безперервній вибірці команд із пам'яті і їх виконанні. Ці дві операції процесор здійснює нескінченно. Процесор починає працювати відразу після ввімкнення живлення або початкового скидання і завершує після вимкнення живлення. **Основний машинний цикл процесора** – нескінченний цикл, в якому команди витягуються із пам'яті, декодуються і виконуються.

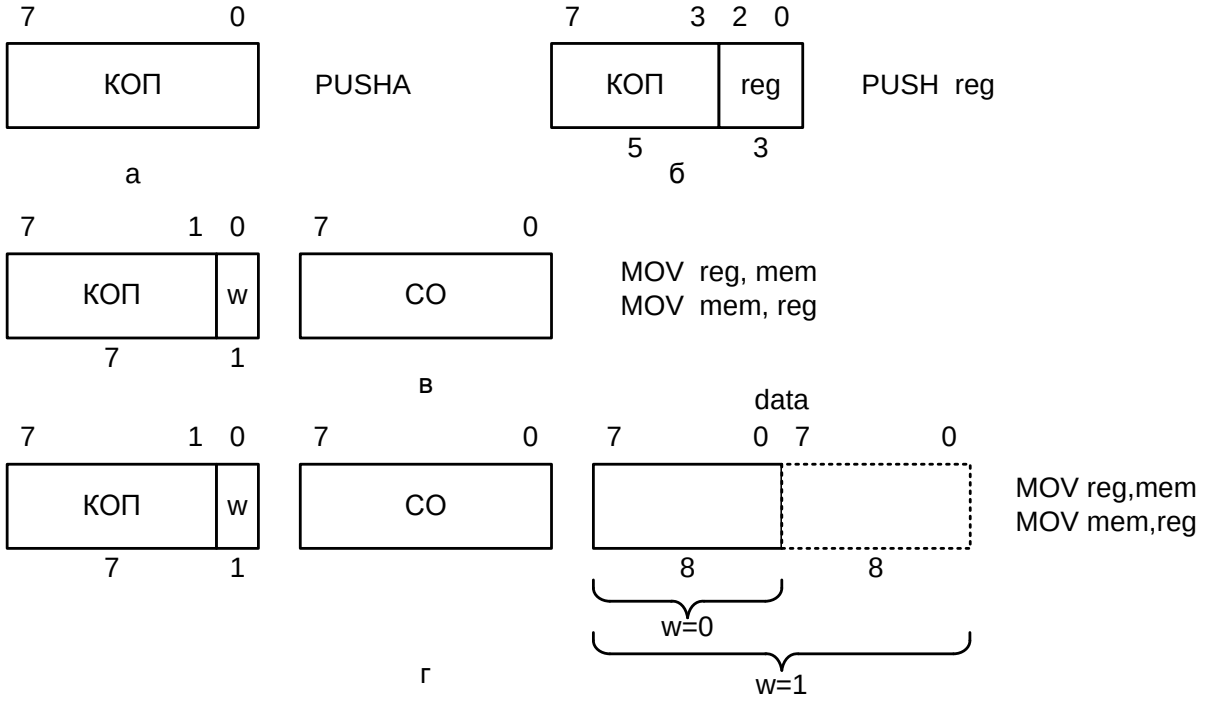


Рис. 3.6. Формати машинних команд МП Intel x86: а – команда складається тільки з КОП; б – КОП і РА задані у вигляді одного байта; в – РА операнда, в якому СО заданий у вигляді окремого байта; г – РА операнда, в якому крім СО міститься додатково один або два байти даних

3.4.1. Етапи виконання команди

Основний машинний цикл повторюється доти, доки не буде вимкнено живлення або процесор не отримає команду *зупинки* (HLT – HaLT). Основний машинний цикл процесора складається з таких етапів: цикл вибірки команди (ЦВиб) і цикл виконання команди (ЦВик). У зарубіжній літературі ЦВиб називається Fetch, ЦВик – Execute (або Exec).

Під час вибірки команди процесор повинен:

– прочитати із пам'яті перший байт команди, який містить КОП, і помістити його в регістр команд IR, IR = Memory [IP];

– перевести покажчик команд IP на наступну команду шляхом додавання до нього одиниці, IP = IP + 1.

Під час виконання команди процесор повинен:

– декодувати команду, яка знаходиться у регістрі команд IR – цикл Decode (після декодування будуть визначені довжина команди, операція АЛП, кількість і довжина операндів);

– отримати вхідні операнди, розташовані в пам'яті або всередині процесора, і розмістити їх у регістрах R1 і R2 – цикл Operands;

– виконати в АЛП необхідну операцію, код якої отримано від КП, – цикл Execute;

– зберегти результат операції де-небудь у пам'яті або всередині процесора – цикл Save.

Таким чином, ЦВик у найбільш повному вигляді складається з чотирьох етапів, а основний машинний цикл – з п'яти етапів: Fetch, Decode, Operands, Execute, Save. Слід зазначити, що всі етапи ЦВик необхідні не всім командам, тому обов'язковим є тільки декодування команди в КП. Наприклад, при виконанні команди зупинки HLT не потрібні вихідні операнди і збереження результату, а при виконанні команди інкремента вмісту комірки пам'яті необхідний тільки один операнд – джерело даних. Якщо результат виконання команди повинен зберігатися в процесорі, то його не потрібно зберігати в пам'яті, тобто етап Save не потрібен.

3.4.2. Моделювання основного машинного циклу

Моделювання основного машинного циклу процесора можна виконати на іншому комп'ютері відомою Вам мовою програмування високого рівня. Програму, яка моделює основний машинний цикл МП Intel 8086 мовою С, наведено нижче:

```
// Програма моделює роботу процесора
// Визначити псевдоніми типів даних для роботи з
// процесором
typedef unsigned int ADDRESS;// адреса комірки пам'яті
typedef unsigned char BYTE;  // байт
typedef unsigned short WORD; // слово
// Визначити адресний простір для процесора 8086
const unsigned int SIZE_8086 = 1048576; // 1 MB
const unsigned char HLT = 0xF4; // КОП команди HLT ("Зупинити")
// Визначити пам'ять з байтовою організацією
BYTE Memory [SIZE_8086];
// Визначити змінні, які описують регістри процесора
ADDRESS IP, EAR; // покажчик команд і регістр адреси
BYTE IR;         // регістр команди, містить КОП
// Процедура вибірки першого байта команди з пам'яті
void Fetch ()
{
  EAR = IP;          // Подати адресу команди на ША
  IR = Memory [EAR];// Отримати КОП із пам'яті
                     // і зберегти його
  IP ++; // Перевести покажчик на наступну команду
}
void Decode () {};  // Процедура декодування КОП команди
void Operands () {};// Процедура вибірки операндів
void Execute () {}; // Процедура виконання команди
```

```
void Save () {};    // Процедура збереження результату
int main ()
{
// Виконати ініціалізацію покажчика команд
// Після скидання процесор повинен стартувати
// за фізичною адресою 0xFFFF0
  IP = 0xFFFF0;
  do
  {
    Fetch ();   // вибрати КОП з пам'яті
    Decode ();  // декодувати КОП
    Operands ();// отримати операнд або операнди
    Execute (); // виконати команду
    Save ();    // зберегти результат у пам'яті
  } While (IR! = HLT);
}
```

На початку моделювальної програми визначено нові імена як псевдоніми наявних типів даних. Далі оголошено константи, які визначають характерні риси для МП 8086: місткість пам'яті становить 1 МВ, а код операції команди *зупинки* HLT – 0xF4. Пам'ять оголошено як масив байтів.

Найбільш важливим у цій програмі є визначення функцій, які моделюють основні етапи виконання команди. Тіла всіх функцій, за винятком Fetch (), залишено порожніми, оскільки необхідно змоделювати тільки основний машинний цикл. Функція Fetch () виконує ті дії, які є характерними для всіх процесорів. Вона звертається до пам'яті за адресою, переданою в регістр адреси EAR з покажчика команди IP. Отриманий із пам'яті байт зберігається у регістрі команди IR, після чого за допомогою операції інкремента покажчик команд IP переводиться на наступну комірку пам'яті. Якщо команда містить додаткові байти, то вміст покажчика команд IP під час виконання команди буде збільшуватися, послідовно вказуючи на наступні байти команди для вилучення їх із пам'яті. Ці дії будуть повторюватися доти, доки з пам'яті не буде прочитано всю команду.

Цикл виконання команди буде повторюватися безперервно доти, доки не буде прочитано команду, яка має КОП, що дорівнює 0xF4 (код команди зупинки).

Початкове значення покажчика команд IP встановлюється таким, що дорівнює 0xFFFF0, як це прийнято при ініціалізації МП 8086.

3.4.3. Інші машинні цикли процесора

Справжній процесор використовує кілька видів машинних циклів: читання з пам'яті і запис у пам'ять, читання з пристрою введення і запис у пристрій виведення, оброблення переривань та ін. Як приклад наведемо

перелік машинних циклів процесора Intel 8080:

M1 – цикл вибірки команди;

M2 – цикл читання з пам'яті;

M3 – цикл запису в пам'ять;

M4 – цикл читання зі стеку;

M5 – цикл запису у стек;

M6 – цикл введення з пристрою введення;

M7 – цикл виведення в пристрій виведення;

M8 – цикл оброблення запиту на переривання;

M9 – цикл зупинки;

M10 – цикл оброблення переривання при зупинці.

Для виконання команди може знадобитися від трьох до п'яти машинних циклів M1 – M10, при цьому першим машинним циклом кожної команди є цикл M1. Кожен машинний цикл виконується протягом 3 – 5 періодів тактової частоти.

Видно, що серед машинних циклів немає циклу виконання команди. Це не помилка – виконання команди процесором (додавання, віднімання, логічні операції та ін.) відбувається в останньому четвертому або п'ятому такті останнього машинного циклу.

Запитання до розділу 3

1. Що розуміють під архітектурою процесора?
2. Що розуміють під структурою процесора?
3. Із яких двох підсистем складається процесор? Яке призначення кожної підсистеми?
4. Який пристрій виконує арифметичні обчислення?
5. Де зберігається адреса наступної команди?
6. Де зберігається команда, яка виконується процесором у поточний момент?
7. Який пристрій декодує команду?
8. Який пристрій виробляє керувальні сигнали? На які підсистеми процесора і комп'ютера надходять ці сигнали?
9. Яка інформація зберігається в машинній команді процесора?
10. Що описує формат команди?
11. Що відбувається під час основного машинного циклу процесора?
12. З яких етапів складається основний машинний цикл процесора?
13. З яких етапів обов'язково має складатися основний машинний цикл процесора?
14. Які дії виконує процесор під час циклу вибірки команди?

4. ХАРАКТЕРИСТИКИ ПРОЦЕСОРА

Характеристики процесора можна поділити на дві групи: архітектурні і неархітектурні. **Архітектурними** називаються характеристики, які впливають на сумісність програм на рівні машинних кодів, **неархітектурними** – характеристики, які не впливають на програмну сумісність, а визначають його продуктивність та вартість.

Можна виділити такі архітектурні характеристики:

– тип організації комп'ютера;
– ідеологія побудови системи команд процесора;
– розрядність процесора;
– адресний простір процесора;
– архітектура процесора;
– кількість адрес пам'яті, які задаються у команді.

До неархітектурних характеристик належать тактова частота процесора, його продуктивність, споживана потужність, ціна та ін.

4.1. Тип організації комп'ютера

Тип організації комп'ютера – базові принципи його організації. Розрізняють архітектуру комп'ютера фон Неймана (або прінстонську) і гарвардську архітектуру. Тип організації комп'ютера впливає на систему команд процесора. Якщо комп'ютер для зберігання команд і даних використовує *спільну пам'ять*, то процесор має архітектуру фон Неймана. При такій організації процесор має команди, які дають можливість обробляти будь-які дані незалежно від того, де вони розташовані – в пам'яті даних або в пам'яті команд. Більшість універсальних процесорів мають архітектуру фон Неймана.

Якщо комп'ютер для зберігання даних і команд використовує різні типи пам'яті – *пам'ять даних і пам'ять команд*, то він має гарвардську архітектуру. Комп'ютери з гарвардською архітектурою мають окремі шини для передачі даних і команд. Як правило, комп'ютери з такою архітектурою випускаються у вигляді однокристальних МК, які мають два види пам'яті даних: внутрішню (невеликої місткості), розташовану на кристалі, і зовнішню, розташовану поза МК. При використанні гарвардської організації комп'ютера процесор має окремі набори команд для роботи з кожним типом пам'яті: з внутрішньою пам'яттю даних, пам'яттю команд і зовнішньою пам'яттю даних.

4.2. Ідеологія формування системи команд процесора

Ідеологія формування системи команд – спосіб формування системи команд процесора, який визначає кількість команд і їх складність. Іншими словами, ідеологія формування системи команд показує, наскільки великою є система команд процесора і наскільки складними можуть бути самі команди. Розрізняють два види ідеології побудови системи команд: процесори зі складною системою команд – CISC-процесори (Complex Instruction Set Command) і процесори зі скороченою системою команд – RISC-процесори (Reduced Instruction Set Command).

CISC-процесори мають велику кількість *складних* команд (зазвичай сотні), режимів адресації і використовуваних типів даних. Оскільки команди процесора складні, потрібні значні ресурси процесора для декодування і виконання таких команд. Тому команди CISC-процесора мають велику довжину, і кожна команда виконується *протягом декількох періодів тактової* частоти CLK, а середній час виконання команди становить кілька тактів (а часто-густо кілька десятків тактів). Прикладами CISC-процесорів є комп'ютер VAX-11 фірми DEC і процесори з архітектурою x86, розробленою фірмою Intel. Процесор комп'ютера VAX-11 мав більше 300 команд, а процесори з архітектурою x86 у цей час мають більше 400 команд (табл. 4.1).

Таблиця 4.1

Фірма-розробник	Моделі процесорів
DEC	VAX-11
Intel	x86, x64
AMD	x86, x64

RISC-процесори мають невелику кількість *простих* команд (зазвичай не більше сотні), режимів адресації і використовуваних типів даних. Прості й короткі команди процесора декодуються і виконуються швидко. Кожна команда RISC-процесора виконується не більше одного періоду тактової частоти CLK, а відтак в одному такті можуть виконуватися кілька команд. Індустрія випускає велику кількість RISC-процесорів. Прикладами RISC-процесорів є як потужні 64-розрядні процесори, які використовують у серверах і робочих станціях, так і прості 8- і 16-розрядні МК (табл. 4.2). Наприклад, МП PIC 16Cx має 34 команди.

Потужні RISC-процесори виконують 3 – 4 команди за один такт. Наприклад, МП SPARC може виконати до чотирьох команд за один такт, а

МП PA RISC – до чотирьох команд, які обробляють дійсні числа.

Таблиця 4.2

Фірма-розробник	Моделі процесорів	Розрядність
DEC	Alpha	64 біти
Sun Microsystems	SPARC	64 біти
Hewlett Packard	PA RISC	64 біти
IBM, Apple, Motorola	PowerPC	64 біти
MIPS	R10000	64 біти
Intel	80860, 80960	32 біти
PIC	16Cx	16 бітів
Ліцензія ARM Limited	ARM (Cortex)	32 біти

В останні роки спостерігається тенденція комбінування організації процесорів. Так, у МП з архітектурою Intel x86, починаючи з МП 80486, CISC-система команд реалізується RISC-ядром процесора. Це дало змогу довести середній час виконання команди майже до одного такту (точніше, 1,25 у середньому).

4.3. Розрядність процесора

Розрядність процесора – це типова кількість бітів, яка може оброблятися *більшістю* команд процесора. Розрядність процесора – основна його характеристика, яка безпосередньо впливає на інші його характеристики. Розрядність процесора визначається розрядністю *внутрішніх регістрів*, розрядністю АЛП і шириною внутрішніх шин процесора. Таким чином, якщо внутрішні регістри процесора мають розрядність 8 бітів, то його команди можуть обробляти тільки 8-розрядні дані. Такий процесор є 8-розрядним (байтовим). Розрізняють 8-, 16-, 32- і 64-розрядні процесори. Іноді спеціальні процесори мають розрядність, яка не кратна цілому степеню числа 2. Так, сигнальні процесори Motorola DSP 560xx і DSP 563xx є 24-розрядними.

16-розрядний процесор має 16-розрядні внутрішні регістри. Більшість його команд можуть обробляти 8- і 16-розрядні дані, а деякі з них – і 32-розрядні. Аналогічно 32-розрядний процесор має 32-розрядні внутрішні регістри, і більшість його команд можуть обробляти 8-, 16- і 32-розрядні дані. 64-розрядні процесори мають внутрішні регістри такої самої розрядності. Деякі типи процесорів і їх розрядність наведено в табл. 4.3.

Таблиця 4.3

Розрядність	Моделі процесорів
8 бітів	8080/85, Z80, 6809, MCS-48, MCS-51, 16Cx
16 бітів	8086/88, 80186, 80286
24 біти	DSP 560xx, DSP 563xx
32 біти	80386, 80486, Pentium, PowerPC, Athlon, 80860, 80960
64 біти	Alpha, IA-64, SPARC, PA RISC, PowerPC, R10000

Слід зазначити, що розрядність зовнішньої ШД процесора не впливає на його розрядність, а визначає його продуктивність і вартість. Так, 16-розрядні процесори 8086 і 8088 мають відповідно 16-розрядну та 8-розрядну ШД. Неважко здогадатися, що процесор 8086 дорожчий і продуктивніший, ніж архітектурно ідентичний йому процесор 8088. МП 80386 випускався в двох варіантах – з 16-розрядною шиною даних (80386SX) і 32-розрядною (80386DX). Дешевший процесор дає можливість використовувати більш дешеву пам'ять і периферійні мікросхеми при створенні комп'ютера.

Розрядність процесора впливає не тільки на оброблення даних, а й визначає інші його характеристики, зокрема адресний простір процесора. Процесори з більшою розрядністю мають більший адресний простір і більш складні способи керування пам'яттю. Тому докладніше розглянемо цю характеристику процесора.

4.4. Адресний простір процесора

Адресний простір процесора – це максимальна місткість фізичної пам'яті, яка може бути підключена до процесора без додаткових апаратних засобів. Величина адресного простору визначається розрядністю зовнішньої шини адреси:

$$\mathbf{АП = 2^{ША}},$$

де **ША** – розрядність шини адреси процесора.

Так, якщо шина адреси процесора 8080 має розрядність 16 бітів, то його адресний простір становить 64 КВ (2^{16}). Процесор 80386 і всі наступні моделі, аж до МП Pentium, мають 32-розрядну шину адреси і, відповідно, адресний простір 4 GB (2^{32}). Приклади адресного простору для різних процесорів наведено в табл. 4.4.

Адресний простір процесора показує потенційні можливості комп'ютера при адресації даних, але місткість встановленої пам'яті зазвичай менше і

визначається такою характеристикою, як "місткість пам'яті". На практиці можна встановити в комп'ютері пам'ять, місткість якої перевищує його адресний простір, однак для цього будуть потрібні додаткові апаратні й програмні засоби.

Таблиця 4.4

Моделі процесорів	Розрядність ША	Адресний простір
MCS-51 (внутрішня пам'ять)	8 бітів	256 B
MCS-51 (зовнішня пам'ять)	16 бітів	64 KB
PDP-11, 8080/85, Z80	16 бітів	64 KB
8086	20 бітів	1 MB
80286	24 біти	16 MB
80386, 80486, Pentium	32 біти	4 GB
Alpha 21164	40 бітів	1 TB
Alpha 21264	44 біти	16 TB
Power PC	40 бітів	1 TB

4.5. Архітектура процесора

Архітектура процесора визначає наявність і використання внутрішніх програмно-доступних регістрів процесора для оброблення даних. Розрізняють три види архітектури процесорів: з акумулятором, з регістрами загального призначення (РЗП) і зі стеком.

Спочатку розглянемо позначення операндів, які беруть участь у командах оброблення даних. Деяка абстрактна команда може бути подана у вигляді

Destination = Source1 op Source2,

де **op** – виконувана операція (+, -, •, /, or, and і т. ін.);

Destination – приймач результату;

Source – джерело результату.

Операнди, розташовані праворуч від знака рівності, задають вхідні дані для виконання операції, тобто є *джерелами* операції. Вони визначають, де знаходяться операнди – у пам'яті або регістрах процесора. Під час виконання операції операнди-джерела не змінюються. Операнд, розташований зліва від знака операції, визначає місце розміщення результату операції і тому називається *приймачем* результату. Після виконання операції *op* вміст приймача результату замінюється новим значенням.

4.5.1. Архітектура з акумулятором

Процесор має **архітектуру з акумулятором**, якщо для зберігання даних він використовує один або два програмно-доступних регістра, які називають *акумуляторами*. Один акумулятор у процесорі називається "регістр А", два акумулятори – "регістри А і В".

При виконанні арифметичних і логічних операцій зазвичай потрібні два операнди. У процесорі з акумулятором *один* з операндів і *результат* операції знаходяться в акумуляторі:

`Acc = Acc op Source,`

де **`Acc`** – акумулятор;

`Source` – комірка пам'яті або інший регістр, який зберігає операнд;

`op` – певна операція оброблення даних.

Якщо місце знаходження одного операнда і приймача результату незмінне і однакове для всіх команд оброблення даних, то його можна не вказувати у команді. Досить вказати в команді тільки один операнд-джерело. Тому команди процесора з акумуляторною архітектурою короткі й виконуються швидко. З іншого боку, оскільки місце знаходження одного з операндів фіксоване тільки в акумуляторі, перед тим як виконати певну двохоперандну команду оброблення даних, один із операндів необхідно переслати в акумулятор. Для виконання наступної операції акумулятор потрібно звільнити від результату попередньої команди і знову помістити в акумулятор новий операнд. Таким чином, програми для процесорів з акумуляторною архітектурою мають велику кількість команд пересилання операндів в акумулятор або з нього.

В основному машинному циклі процесора з акумулятором не потрібне пересилання результату в пам'ять. На етапі вибірки з пам'яті операндів Operands() необхідно отримати тільки один операнд. Декодування команд виконується відносно просто. Тому можна стверджувати, що процесори з акумуляторної архітектурою – прості, недорогі, зазвичай байтові і ненайшвидкодійні. Прикладами процесорів, які мають архітектуру з акумулятором, є МП Intel MCS-51 і 8080/85, МП Zilog Z80 та ін.

4.5.2. Архітектура з регістрами загального призначення

Процесор має **архітектуру з регістрами загального призначення**, якщо для зберігання даних він використовує більше двох програмно-доступних регістрів, які називаються *регістрами загального призначення* (РЗП). Для роботи процесора, який має архітектуру з РЗП, потрібно, щоб один з операндів знаходився у регістрах. Другий операнд і результат

виконання операції можуть знаходиться або в регістрі, або в пам'яті:

Reg = Reg op Memory,

Memory = Reg op Memory,

де **Reg** – регістр процесора;

Memory – комірка пам'яті, яка зберігає операнд;

op – деяка операція оброблення даних.

Регістри процесора являють собою невелику швидкодійну внутрішню пам'ять, розташовану всередині процесора, де кожна комірка має власне ім'я. Регістри процесора можна порівняти з кишенями на одязі: їх небагато, до них швидкий доступ, вони дорогі, Ви за них вже заплатили кошти, але вони завжди під рукою. Тому їх потрібно використовувати якомога частіше.

У командах оброблення даних необхідно вказувати, де розташований кожен операнд – у пам'яті або в регістрах. Тому процесор з РЗП має довші і складніші команди, ніж процесор з акумулятором. У зв'язку з цим процесор з РЗП має і більш складну структуру для виконання таких команд, ніж процесор з акумулятором. Однак програми, написані для таких процесорів, коротші від програм, написаних для процесорів з акумулятором, тому що для них не потрібно великої кількості операцій пересилань.

Основний машинний цикл процесора з РЗП потребує використання етапу отримання одного або двох операндів Operands() і етапу зберігання результату операції Save(). Етап декодування команди Decode() досить складний через складність самих команд і різноманіття розташування операндів, тому для виконання декодування необхідна додаткова апаратура. У зв'язку з цим процесори з РЗП – зазвичай складні і дорогі пристрої. Усі сучасні потужні і швидкодійні процесори мають архітектуру з РЗП. Приклади процесорів з РЗП наведено в табл. 4.5.

Таблиця 4.5

Моделі процесорів	Тип даних	Всього регістрів	Доступно регістрів
VAX-11	Цілі	16	14
x86	Цілі	12	6
x86	Дійсні	8	8
Alpha	Цілі	80	80
Alpha	Дійсні	72	72
SPARC	Цілі	32	32
SPARC	Дійсні	144	32
PA8000	Цілі	32	32
PA8000	Дійсні	32	32
R10000	Цілі	64	33
R10000	Дійсні	64	32

Кількість регістрів у процесорах з РЗП може змінюватися в широких межах: від шести регістрів МП Intel x86, які фактично може використовувати програміст для зберігання даних, до 80 цілочислових регістрів у процесорі Alpha (див. табл. 4.5). Як випливає з табл. 4.5, RISC-процесори мають велику кількість РЗП. Типова кількість доступних у кожен момент часу регістрів становить 32. CISC-процесори мають меншу кількість РЗП, тому зазвичай програміст може використовувати 8 – 16 регістрів.

Багато процесорів з акумуляторами мають у своєму складі РЗП. Наприклад, МП Intel 8080/85 має 6 РЗП, а МП Intel MCS-51 – 4 банки по 8 регістрів кожен. Це не означає, що зазначені процесори, які були віднесені до так званих процесорів з акумуляторами, мають архітектуру з РЗП. У цих процесорах регістри можна використовувати тільки як другий операнд-джерело, тоді як першим операндом і приймачем результату обов'язково має бути акумулятор.

4.5.3. Архітектура зі стеком

Процесор зі стеком – це процесор, який вхідні операнди витягує з вершини стеку і результат виконання операції поміщає назад у вершину стеку. Цей процесор не має програмно-доступних регістрів для зберігання даних, а працює з *апаратним стеком*. Стек як структура даних відомий давно, але до початку 60-х років використовували його програмну реалізацію. Поява апаратного стеку, тобто введення в систему команд процесора інструкцій, які оперують зі стеком, є найбільш важливим удосконаленням організації комп'ютера фон Неймана. Поява апаратного стеку в архітектурі комп'ютерів дає можливість ефективно організовувати підпрограми і виконувати введення-виведення через переривання.

Для виконання операцій процесор зі стеком використовує дані, які зберігаються в пам'яті з особливою організацією, яка називається стеком. *Стек* – пам'ять з послідовним доступом, звернення до елементів якої виконується за принципом LIFO (Last In Fist Out – першим прийшов – останнім вийшов), причому в кожен момент часу програмісту доступні один або два елементи, які завантажені останніми і знаходяться у вершині стеку TOS (Top Of Stack). Верхній елемент стеку адресується спеціальним програмно-доступним регістром, який називається *покажчиком стеку SP* (Stack Pointer). Для того щоб виконати операцію з певними даними, розташованими у пам'яті, їх необхідно завантажити в стек. При роботі зі стеком доступні дві операції: проштовхування в стек Push і виштовхування із стеку Pop.

Покажчик стеку SP адресує операнд, завантажений у пам'ять останнім. Унарні операції виконуються з одним верхнім елементом стеку, бінарні – з

двома верхніми елементами (рис. 4.1, а), причому верхній елемент втрачається, а результат поміщається на місце другого від вершини елемента стеку (рис. 4.1, б):

`Push (Pop op Pop),`

де **Pop** – операція виштовхування зі стеку;

Push – операція проштовхування у стек;

op – деяка операція оброблення даних.

Процесор оперує з даними, розташованими в стеку і адресованими неявно за допомогою покажчика стеку SP, тому адреси операндів у командах вказувати не потрібно. Команди оброблення даних не містять адрес, вони максимально короткі і прості, але виконуються повільно. Це пояснюється тим, що всі дані обробляються всередині процесора – процесор читає обидва операнди зі стеку в основній пам'яті, виконує необхідну операцію і надсилає результат назад у стек, розташований в основній пам'яті. Така кількість передач даних між пам'яттю і процесором різко знижує його продуктивність.

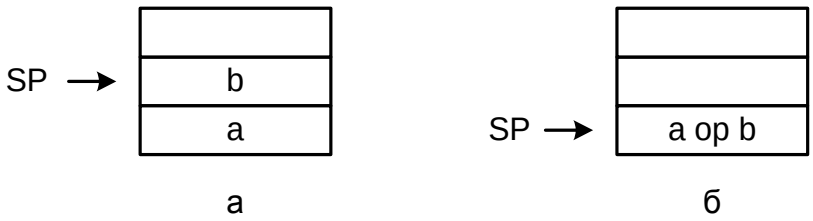


Рис. 4.1. Стан стеку при виконанні команд оброблення даних:
а – до виконання операції; б – після виконання операції

Процесор зі стеком має просту структуру, але програми, написані для нього, довші від аналогічних програм, написаних для процесорів з РЗП, тому що потребують великої кількості пересилань для завантаження даних з основної пам'яті у стек і збереження даних зі стеку в основній пам'яті.

Основний машинний цикл процесора зі стеком потребує виконання всіх етапів, які, однак, виконуються швидко. Вони могли б виконуватися ще швидше, якби не часті звернення до пам'яті. Тому в деяких стекових процесорах як стек використовують блок регістрів, організованих у вигляді стеку.

Приклади стекових процесорів наведено в табл. 4.6.

Сучасні процесори зі стеком використовують як числові співпроцесори, що працюють паралельно з основними процесорами і виконують спеціальні обчислення з високою продуктивністю. Це числові співпроцесори Intel x87, які в цей час входять до складу МП Intel x86. Крім того, стек – це вкрай корисна структура, яка використовується для зберігання адрес повернення,

оброблення вкладених переривань, передачі параметрів у підпрограму та ін. Тому апаратно реалізований стек використовується у кожному процесорі, і, відповідно, кожен процесор має набір команд для роботи зі стеком.

Таблиця 4.6

Архітектура	Моделі процесорів	Розрядність
З акумулятором	8080/85, Z80, 6809, MCS-48, MCS-51	8 бітів
З РЗП	x86, Alpha, SPARC, PA 8000, R10000	32–64 біти
Зі стеком	x87	32, 64, 80 бітів

4.6. Кількість адрес пам'яті, які задаються в команді

Кількість адрес пам'яті, що задаються в команді, – це характеристика процесора, яка показує найбільшу кількість комірок пам'яті, адреси яких можна вказувати в команді. Слід зазначити, що за цією характеристикою визначають не кількість операндів у команді, а те, скільки операндів команди може бути розташовано у пам'яті. Цей показник безпосередньо впливає на довжину команд процесора і їх швидкодію.

N-адресним називається процесор, якщо більшість його команд може використовувати N операндів, які є комірками пам'яті. За цим показником усі процесори можна поділити на такі групи (табл. 4.7):

– триадресні;
– двохадресні;
– півтораадресні;
– одноадресні;
– безадресні.

Таблиця 4.7

Вид команд	Архітектура	Моделі процесорів
Триадресні	з РЗП	VAX-11
Двохадресні	з РЗП	VAX-11, PDP-11
Півтораадресні	з РЗП	x86
Одноадресні	з акумулятором	8080, MCS-51
Безадресні	зі стеком	x87

Триадресні команди дають можливість визначити три адреси комірок пам'яті, заданих у вигляді деякого режиму адресації, розміщуючи два операнди-джерела і операнд-приймач результату в окремих комірках пам'яті. Загальна форма запису команди для такого процесора має вигляд

Memory3 = Memory2 op Memory1,

де **Memory3, Memory2, Memory1** – адреси комірок пам'яті;

op – деяка операція оброблення даних.

Триадресні команди – довгі й складні, тому виконуються порівняно повільно. Наприклад, для триадресних команд 32-розрядного процесора тільки для задання адреси може знадобитися 12 байтів (три адреси по чотири байти кожен).

Триадресні процесори – це процесори, в яких більшість команд є триадресними. Такі команди зазвичай мають CISC-процесори (див. табл. 4.7) – дорогі й продуктивні.

Двохадресні команди дають можливість визначити дві адреси комірок пам'яті, заданих за допомогою деякого режиму адресації, розміщуючи в окремих комірках два операнди-джерела, один з яких буде операндом-приймачем результату. Загальна форма запису команди для такого процесора має вигляд

Memory2 = Memory2 op Memory1,

де **Memory2, Memory1** – адреси комірок пам'яті;

op – деяка операція оброблення даних.

Двохадресні процесори – це процесори, в яких більшість команд є двохадресними. Такі команди також мають зазвичай потужні CISC-процесори (див. табл. 4.7).

Одноадресні команди дають можливість визначити одну адресу комірок пам'яті, які задають за допомогою деякого режиму адресації, а місце знаходження другого операнда має задаватися за замовчуванням, наприклад, в акумуляторі. Загальна форма запису команди для такого процесора має вигляд

Acc = Acc op Memory1,

де **Memory1** – адреса комірки пам'яті;

op – деяка операція оброблення даних.

Одноадресні команди – короткі, прості, швидко виконувані.

Одноадресні процесори – це процесори, в яких більшість команд є одноадресними. Одноадресні процесори мають архітектуру з акумулятором (див. табл. 4.7).

Півтораадресні команди дають можливість визначити два операнди, але тільки один з них може зберігатися в комірці пам'яті. В такому процесорі кожна команда може містити *не більше однієї* адреси пам'яті. Півтораадресний процесор дає змогу виконувати команди типу "пам'ять-регістр" і "регістр-пам'ять", проте виконання операцій типу "пам'ять-пам'ять"

не є допустимим. Загальна форма запису команди для такого процесора має вигляд

Reg = Reg op Reg1,

Reg = Reg op Memory1,

Memory1 = Reg op Memory1,

де **Memory1** – адреса комірки пам'яті;

Reg, Reg1 – регістр загального призначення;

op – деяка операція оброблення даних.

Півтораадресні команди є оптимальними як за довжиною, так і за швидкодією.

Півтораадресні процесори – це процесори, в яких більшість команд є півтораадресними. Такі процесори мають архітектуру з РЗП. Прикладом півтораадресного процесора є МП Intel x86 (див. табл. 4.7).

Безадресні команди взагалі не містять операндів, оскільки дані, необхідні для команди, адресуються неявно. Загальна форма запису такої команди має вигляд

Op,

де **Op** – деяка операція оброблення даних.

Оскільки у безадресних командах операнд не вказують, то вони обробляють дані, які зберігаються у найбільш часто використовуваних ресурсах процесора – регістрах або прапорцях. Які саме регістри або прапорці обробляють такі команди залежить від конкретного процесора. У МП Intel x86 безадресні команди використовують прапорці CF, DF, IF, а серед регістрів найбільш популярними є такі: акумулятор (EAX / AX / AL), регістр даних (EDX / DX), регістр-лічильник (ECX / CL), адресні регістри (індекс джерела ESI / SI) та індекс приймача (EDI / DI). Таких команд доволі багато, і якщо їх часто використовувати, можна отримати значну економію пам'яті.

Безадресні команди – найкоротші, прості і дуже швидко виконуються, але вони зовсім не гнучкі – не можна вказати інший операнд.

Безадресні процесори. Кожний процесор використовує безадресні команди, але існують повноцінні процесори, більшість команд яких є безадресними. Саме до таких належать стекові процесори, в яких дані адресують за допомогою покажчика стеку SP. Загальна форма запису команди для такого процесора має вигляд

Push (Pop op Pop),

де **Pop** – операція виштовхування зі стеку;

Push – операція проштовхування у стек;

op – деяка операція оброблення даних.

Прикладом безадресного процесора є арифметичний співпроцесор (СП) Intel x87 (див. табл. 4.7). У стекових процесорах команди короткі, але вони виконуються порівняно повільно, оскільки дані для них розташовані в пам'яті. Цей недолік можна виправити, якщо використовувати РЗП СП, які організують у вигляді стеку.

Коротко підсумуємо розглянуті вище можливості процесорів різних типів. Найбільш потужні команди мають триадресні процесори. Програмісту дуже зручно використовувати триадресні команди, хоча іноді вони надлишкові. Програми виходять компактними, оскільки вони мають мінімальну кількість команд пересилання. Це пов'язано з тим, що будь-яка триадресна команда оброблення даних неявно містить команди пересилання. Однак для реалізації таких процесорів потрібні великі витрати обладнання, тому вони складні й дорогі.

Оптимальним вирішенням компромісу "зручність програміста – складність процесора" є двох- і півтораадресні процесори. Такі процесори – помірно складні й дорогі, а програми – все ще досить компактні.

Одноадресні команди використовують процесори з акумулятором, які призначені, перш за все, для розв'язання керувальних, а не обчислювальних задач. Вони прості, дешеві, хоча програми для них мають велику кількість команд передачі даних, і тому довгі. Але в цьому випадку головне – низька вартість МП.

Таким чином, мета цієї класифікації – спробувати у кожному реальному процесорі знайти риси тієї чи іншої архітектури, визначити її вплив на програмно-доступні регістри, систему команд, режими адресації, організацію введення-виведення, щоб правильно їх використовувати при програмуванні мовою Асемблер будь-якого процесора.

Запитання до розділу 4

1. Що розуміють під архітектурою процесора?
2. Які архітектурні характеристики процесора Вам відомі?
3. Навіщо потрібно знати архітектурні характеристики процесора?
4. Назвіть характерні риси RISC-процесорів.
5. Назвіть характерні риси CISC-процесорів.
6. Що розуміють під ідеологією побудови системи команд процесора?
7. Що розуміють під розрядністю процесора?

8. Навіщо програмісту знати розрядність процесора?

9. Які властивості процесора описує характеристика "адресний простір"?

10. Які властивості процесора описує характеристика "архітектура процесора"?

11. Що означає характеристика процесора "кількість адрес пам'яті, які задаються в команді"?

12. Якби потрібно було розробити архітектуру простого недорого процесора, яку архітектуру Ви б вибрали?

13. Якби потрібно було розробити архітектуру потужного швидкодійного процесора, яку архітектуру Ви б вибрали?

14. Чому в смартфонах використовують RISC-процесори?

5. АРХІТЕКТУРА ПРОЦЕСОРІВ

Зробимо короткий огляд архітектур гіпотетичних процесорів, які мають прототипи у вигляді реально існуючих МП, розглянемо їх програмні моделі, системи команд і формати команд для кожного процесора, а також визначимо, які зміни необхідно внести у структуру найпростішого процесора для реалізації архітектури кожного процесора. Такий підхід використовується в роботі [4].

Порівняємо приклади програм мовою Асемблер для трьох типів архітектур процесорів, які розв'язують одну й ту саму задачу. Виберемо таке завдання: обчислити значення виразу

$$\boldsymbol{result = x^2 - 2 \cdot x \cdot y + y^2}.$$

При розгляді архітектури процесора вивчимо лише ті команди реального МП, які є необхідними для вирішення цього завдання.

Оцінимо кожну архітектуру, для чого порівняємо програми для трьох процесорів за швидкодією, кількістю команд у програмі й місткістю займаної пам'яті. Час виконання програми будемо визначати як суму часу виконання всіх команд. Час виконання кожної команди наближено можна оцінити за кількістю звернень до пам'яті при виконанні команди – це найповільніша операція при виконанні кожної команди.

Оскільки для програмування мовою Асемблер необхідні певні знання, програми забезпечені коментарями і детально описані.

5.1. Процесор, який має архітектуру з акумулятором

Архітектуру процесора з акумулятором розглянемо на прикладі гіпотетичного процесора H51, прототипом якого є МП Intel MCS-51.

Процесор H51 – однокристальний МК, який має гарвардську архітектуру. До його складу входить 8-розрядний процесор, який має архітектуру з акумулятором. Процесор H51 має два акумулятори A і B для зберігання даних і результатів. Процесор використовує одноадресні команди.

На кристалі МК розташована пам'ять даних місткістю 256 байтів і пам'ять команд місткістю 4 КВ. Тому адресний простір даних цього процесора становить 256 байтів, а для адресації даних досить одного байта.

Структуру процесора H51 показано на рис. 5.1, а програмну модель процесора – на рис. 5.2.

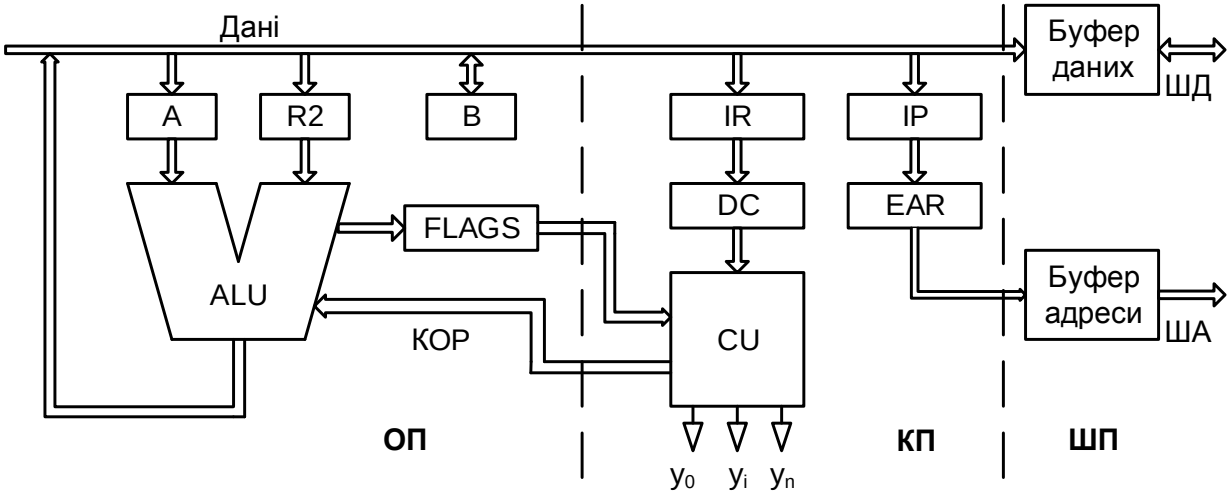


Рис. 5.1. Структура процесора H51

На рис. 5.1 видно, що до структури найпростішого процесора додано два програмно-доступних регістри-акумулятори A і B. Перший акумулятор є основним, другий – додатковим. Акумулятор B використовується при виконанні деяких складних команд. Таким чином, програмна модель процесора H51 містить чотири програмно-доступних регістри: два акумулятори A і B, регістр прапорців Flags і покажчик команд IP.

Рис. 5.2. Архітектура процесора H51

Акумулятор A призначений для зберігання одного операнда при виконанні всіх арифметичних операцій. Результат операції також буде збережений у ньому. Акумулятор B використовується при виконанні операції множення, оскільки команда множення дає змогу множити тільки вміст двох акумуляторів. Регістр прапорів містить ознаки результату – *прапорці*. Прикладом такого прапорця є прапорець перенесення-позики CF. У цьому розділі прапорці використовуються тільки в команді віднімання з урахуванням позики (SUBB). Зазначимо, що для простого віднімання двох чисел значення прапорця перенесення CF має дорівнювати нулю.

Система команд процесора H51, наведена в табл. 5.1, містить тільки ті команди, які використовуються для обчислення заданого виразу. Процесор використовує п'ять команд: передачі даних (MOV); складання (ADD); віднімання з урахуванням позики (SUBB); множення (MUL) і очищення прапорця перенесення CF (CLR).

Таблиця 5.1

Мне-мокод	Операнди	Коментар	Опис	Формат, рис. 5.3	Довжи-на, байти	Кількість пересилань
mov	acc, mem	acc←mem	*Переслати* вміст комірки пам'яті в акумулятор	б	2	3
mov	mem, acc	mem←acc	*Переслати* вміст акумулятора в комірку пам'яті	б	2	3
mov	acc,#const	acc←const	*Переслати* константу в акумулятор	в	2	2
add	A, mem	A← A+mem	*Скласти* вміст акумуля-тора і комірки пам'яті	б	2	3
subb	A, mem	A←A-mem--CF	*Відняти* від акумулятора вміст комірки пам'яті і біт переносу	б	2	3
mul	ab	A ←A•B	*Помножити* вміст обох акумуляторів	а	1	1
clr	c	CF←0	*Очистити* біт переносу	а	1	1

Примітка. acc – один із акумуляторів (A або B), A – акумулятор, mem – комірка пам'яті.

Команда передачі даних MOV призначена для пересилання байта в акумулятор і з нього. Другим операндом при передачі даних може бути або вміст комірки пам'яті, або константа. Таким чином, перші три рядки в табл. 5.1 – це команда передачі даних з різними комбінаціями операндів.

Команди складання ADD і віднімання SUBB призначені відповідно для додавання і віднімання з основного акумулятора вмісту комірки пам'яті.

Команда віднімання крім операнда віднімає з акумулятора ще і значення прапорця CF. Для того щоб значення прапорця CF не впливало на результат віднімання двох чисел, його потрібно обнулити за допомогою команди CLR C.

Команда множення двох чисел MUL дає змогу перемножити вміст двох акумуляторів і помістити результат в основний акумулятор A. Оскільки місце знаходження операндів і результату зумовлене і незмінне, то вказівка операнда AB суто декоративна і є лише підказкою для програміста.

Розглянемо програму, за допомогою якої можна обчислити значення виразу.

```
; Обчислити арифметичний вираз
; RESULT = X^2 - 2XY + Y^2
;
X     DATA    30h ; за адресою 30h має зберігатися операнд X
Y     DATA    31h ; за адресою 31h має зберігатися операнд y
RESULT  DATA  32h ; за адресою 32h буде зберігатися результат
TEMP1   DATA  33H ; комірка пам'яті для тимчасового зберігання даних
TEMP2   DATA  34H ; комірка пам'яті для тимчасового зберігання даних
START:
;                                   Довжина/Тривалість
; Обчислити x^2 і зберегти у temp1
        MOV     A,X     ; A <- X        (2)[3]
        MOV     B,A     ; B <- X        (2)[3]
        MUL     AB      ; A <- X*X      (1)[1]
        MOV     TEMP1,A ; TEMP1 <- X^2  (2)[3]
; Обчислити 2xy
        MOV     A,X     ; A <- X        (2)[3]
        MOV     B,Y     ; A <- Y        (2)[3]
        MUL     AB      ; A <- (X*Y)    (1)[1]
        MOV     B,#2    ; A <- 2        (2)[2]
        MUL     AB      ; A <- 2*(X*Y)  (1)[1]
        MOV     TEMP2,A ; TEMP2<- 2*X*Y (2)[3]
; Обчислити y^2
        MOV     A,Y     ; A <- Y        (2)[3]
        MOV     B,A     ; A <- Y        (2)[3]
        MUL     AB      ; A <- Y^2      (1)[1]
; Скласти три часткових добутки, які зберігаються в
; акумуляторі, і змінних temp1 й temp2
        ADD     A,TEMP2 ; A = (X^2+Y^2) (2)[3]
        CLR     C       ;               (1)[1]
        SUBB    A,TEMP1 ; A = () - 2XY  (2)[3]
        MOV     RESULT,A; зберегти результат (2)[3]
        END     START
```

Програма для процесора H51 являє собою послідовність рядків, кожен з яких є оператором мовою Асемблер. **Оператором** мовою Асемблер може бути або команда процесора, або директива мови Асемблер. Кожна мова Асемблер має список зарезервованих імен директив і команд, які розташовують у полі мнемокода. Існує набір певних імен, які є директивами, і набір імен – команд даного процесора. Решта імен, які вказані у полі мнемокода, є помилковими. Наприклад, в даній програмі зустрічається ім'я DATA, що є ім'ям директиви, а імена MOV, ADD та інші – імена команд процесора. За допомогою відповідних директив мови Асемблер користувач може визначити власні імена.

Кожен оператор мови Асемблер складається з чотирьох стовпців:

```
Мітка:  Мнемокод  Операнди     ;Коментар
```

У першому стовпці зазвичай розташовуються визначені користувачем імена або мітки. Якщо цей рядок – команда процесора, то в першому стовпці знаходиться мітка.

Мітка – символічна адреса комірки пам'яті, в якій розташована ця команда. Якщо в рядку розміщується директива, то в першому стовпці знаходиться визначене користувачем ім'я.

У наведеній вище програмі використовується тільки одна директива мови Асемблер DATA, призначення якої – присвоїти символічне ім'я числовим значенням адреси комірки пам'яті. Надалі у програмі замість адреси можна використовувати це символічне ім'я. Наприклад, ім'я X є ім'ям адреси комірки пам'яті з адресою 0x30, а ім'я Y будемо використовувати замість числового значення адреси 0x31.

У разі необхідності після мнемокода розташовується поле з одним або двома операндами. Наприклад, у команді MOV A, X – два операнди: операнд-приймач результату A і операнд-джерело даних X. Команда CLR C має тільки один операнд – прапорець CF, а для команди MUL AB взагалі операнд не потрібен.

Будь-яка інформація, указана після знака крапки з комою (;), призначена для програміста і є коментарем тих чи інших дій у програмі для реалізації алгоритму.

Розглянемо, як працює програма. Перші кілька рядків коментарів пояснюють її призначення. Далі у програмі визначені імена комірок пам'яті, в яких повинні розташовуватися операнди, що використовуються в програмі. Видно, що імена комірок пам'яті збігаються з іменами змінних у постановці завдання. Таким чином, у програмі визначені імена для двох комірок пам'яті (X і Y), в які перед початком виконання програми мають бути поміщені вихідні дані, і комірка пам'яті RESULT, в якій після виконання

програми буде зберігатися результат. Комірки TEMP1 і TEMP2 використовуються для зберігання проміжних даних.

Команди, за допомогою яких можна обчислити значення виразу, розташовані після мітки Start. У програмі можна виділити чотири фрагменти: в трьох з них обчислюють часткові добутки X^2, Y^2, $2 \bullet X \bullet Y$, а в четвертому – їх складають. Для обчислення значення X^2 операнд X поміщають в акумулятор A, створюють його копію в додатковому акумуляторі B і за допомогою команди MUL перемножують вміст обох акумуляторів $X \bullet X$. Після цього зберігають результат у комірці пам'яті TEMP1.

Далі обчислюють значення виразу $2 \bullet X \bullet Y$: поміщають операнди X і Y в акумулятори A і B, перемножують їх за допомогою команди MUL і отримують добуток в акумуляторі A. Після цього поміщають константу 2 в акумулятор B, множать її на добуток XY, розташований в акумуляторі A. Отриманий добуток зберігають у комірці пам'яті TEMP2, звільняючи акумулятор A.

Добуток Y^2 можна отримати так само, як X^2, з тією лише різницею, що зберігати результат не потрібно. Далі за допомогою команди ADD складають значення двох квадратів X^2 і Y^2, які зберігаються в комірці пам'яті TEMP1 і акумуляторі відповідно. Від отриманої суми з допомогою команди SUBB віднімають частковий додаток $2 \bullet X \bullet Y$, розташований у комірці пам'яті TEMP2. Перед виконанням команди віднімання значення прапорця перенесення CF обнуляють за допомогою команди CLR C. Отриманий результат зберігають у комірці пам'яті з адресою RESULT.

Робота першої програми була розглянута так докладно для того, щоб було зрозуміло головне – для програмування мовою Асемблер необхідно детально вказувати процесору послідовність дій. Вказівками для процесора є машинні команди. Команди процесора працюють з трьома видами об'єктів – комірками пам'яті (X, Y і т. ін.), константами (у цьому випадку 2) і програмно-доступними регістрами (A, B).

Оцінимо довжину програми. Для визначення місткості пам'яті, займаної програмою, потрібно розглянути формати всіх команд із системи команд процесора H51 (рис. 5.3).

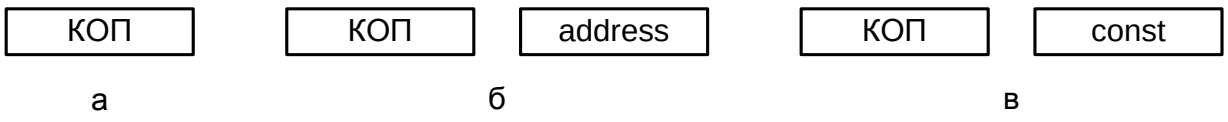


Рис. 5.3. Формати команд процесора H51

Кілька команд мають довжину *один байт*, який є КОП (рис. 5.3, а). Це команди, в яких місце знаходження операнда визначено розробниками і не може бути змінено програмістом. Такими командами є команди множення

(MUL) і команда очищення прапорця CF (CLR). Більшість використовуваних команд має довжину два байти: один байт займає КОП, другий – або адреса комірки пам'яті (рис. 5.3, б), або константа (рис. 5.3, в). Довжина кожної команди в лістингу програми вказана у квадратних дужках у кінці рядка кожної команди.

Знаючи формат команди і розуміючи основний машинний цикл, можна визначити час виконання кожної команди і всієї програми в цілому. Час виконання команди досить точно можна визначити за кількістю звернень до пам'яті під час виконання основного машинного циклу, оскільки процесор працює значно швидше пам'яті. Спочатку команду необхідно вибрати із пам'яті (етап Fetch), а потім вибрати з пам'яті або зберегти в пам'яті *тільки один операнд*. Для вібірки всієї команди таким чином необхідно виконати стільки звернень до пам'яті, скільки байтів становить команда (два звернення для двобайтових команд і одне – для однобайтових). Під час виконання команд, які використовують як операнд комірку пам'яті, необхідно ще одне звернення до пам'яті. Таким чином, команда MOV A, X *тричі* звертається до пам'яті (два звернення для отримання команди і одне – для вилучення операнда X). Для команди, яка використовує як операнд константу, додаткових звернень до пам'яті після її вилучення не потрібно, оскільки число вже знаходиться у процесорі. Таким чином, команда MOV B, # 2 звертається до пам'яті *двічі*. І, нарешті, команди MULL і CLR працюють з даними всередині процесора, додаткових звернень до пам'яті не потрібно. Ці команди звертаються до пам'яті лише *один* раз і працюють максимально швидко. Час виконання кожної команди відповідно до вибраного критерію вказується в кінці кожного рядка програми в круглих дужках. Результати оцінювання тестової програми наведено в табл. 5.4.

Таким чином, програма для процесора з акумулятором складається з 17 команд, займає в пам'яті 27 байтів і звертається до пам'яті 37 разів.

5.2. Процесор, який має архітектуру з регістрами загального призначення

Архітектуру процесора з РЗП розглянемо на прикладі гіпотетичного процесора H86, прототипом якого є МП Intel 80386.

Процесор H86 – однокристальний МП з архітектурою фон Неймана. МП H86 – 32-розрядний процесор, в якому реалізована архітектура з РЗП. Він має шість РЗП для зберігання даних і результатів і виконує півтораадресні команди. Адресний простір цього МП становить 4 GB, і для адресації даних він оперує 32-розрядними логічними адресами. Процесор H86 за одне звернення до пам'яті може читати або записувати 8-, 16- або 32-розрядні числа.

Структуру процесора H86 показано на рис. 5.4, а програмну модель процесора – на рис. 5.5.

Структура МП H86 відрізняється від структури найпростішого процесора наявністю шести РЗП, що називаються EAX, EBX, ECX, EDX, ESI, EDI (див. рис. 5.4).

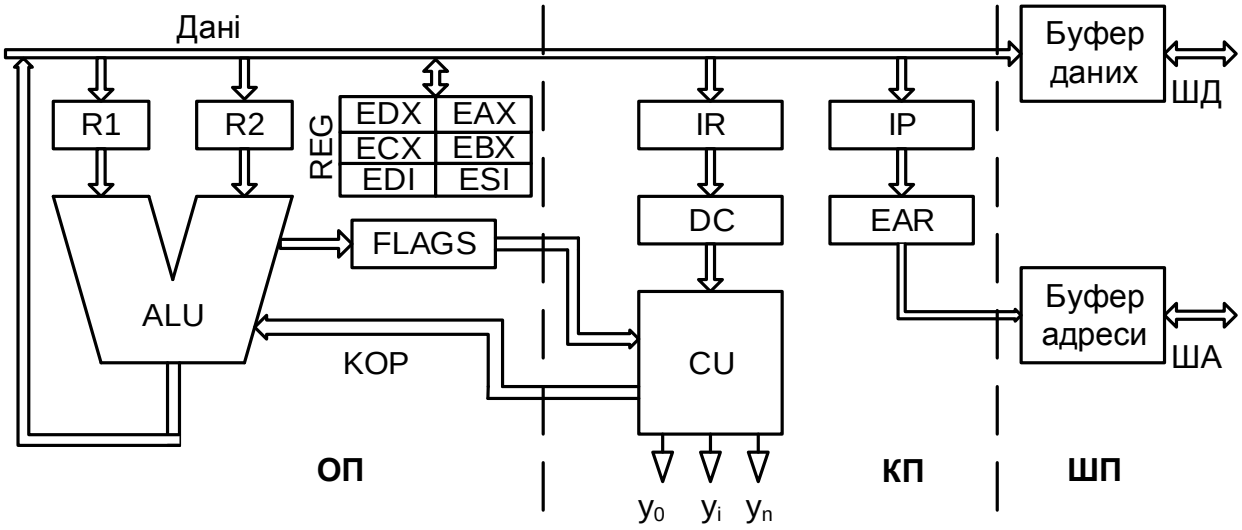


Рис. 5.4. Структура процесора H86

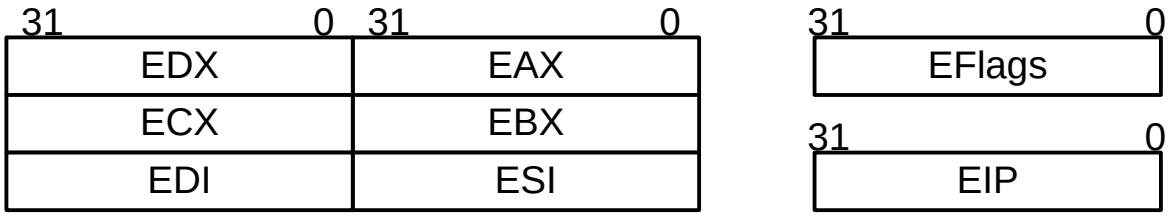


Рис. 5.5. Архітектура процесора H86

Усі РЗП рівноправні й дають змогу зберігати як дані, так і адреси, оскільки їх розрядності збігаються. При виконанні арифметичних операцій вхідні дані й результат можуть розміщуватися в будь-яких регістрах. Як операнд можна використовувати також адреси комірок пам'яті і константи. Єдина вимога до операндів – у команді можна вказувати тільки одну адресу комірки пам'яті (це властивість півтораадресних процесорів).

Програмна модель процесора H86 містить вісім програмно-доступних регістрів: шість РЗП, регістр прапорів EFlags і покажчик команд EIP. Регістри EFlags і EIP мають ті функції, що й регістри Flags і IP у найпростішому процесорі, але збільшену до 32 бітів кількість розрядів.

Команди процесора H86, які будемо використовувати для обчислення

тестового виразу, наведено в табл. 5.2. Процесор використовує чотири команди: передачі даних (MOV); складання (ADD); віднімання (SUB); знакового множення (IMUL). Призначення команд таке саме, як у процесорі H51.

Таблиця 5.2

Мнемо-код	Операн-ди	Коментар	Опис	Формат, рис. 5.6	Довжи-на, байти	Кількість пересилань
mov	dst, src	dst←src	*Переслати* вміст джерела у приймач	б	2	3
add	dst, src	dst←dst+src	*Скласти* вміст джерела і приймача	б	2	3
sub	dst, src	dst←dst-src	*Відняти* від приймача вміст джерела	б	2	3
imul	dst, src	dst←dst•src	*Перемножити* вміст джерела на вміст приймача	а	1	1

Примітка. src – операнд-джерело (адреса комірки пам'яті, константа або регістр), dst – операнд-приймач (адреса комірки пам'яті або регістр).

Кожна команда має два операнди: операнд-джерело src (SouRCe – джерело) і операнд-приймач результату dst (DeSTination – приймач). Операнд-джерело зберігає вихідні дані й може бути константою, регістром або адресою комірки пам'яті. Операнд-приймач до початку операції зберігає другий операнд, а після неї приймає результат операції. Операндом-приймачем можуть бути регістри або адреси комірок пам'яті. Слід зазначити, що неприпустимою є комбінація операндів з двох елементів пам'яті ("пам'ять-пам'ять").

Розглянемо програму, за допомогою якої можна обчислити значення виразу:

```
; Обчислити арифметичний вираз
; RESULT = X^2 - 2XY + Y^2
;
x:      .long   -47 ; за адресою x зберігається перший операнд
y:      .long    63 ; за адресою y зберігається другий операнд
result: .long     0 ; зберегти результат за цією адресою
start:
;                                                Довжина/Тривалість
; Обчислити x^2
```

```
        mov     eax,x       ; x -> eax              (6)[4]
        imul    eax,eax     ; x*x -> eax            (2)[2]
; Обчислити y^2
        mov     ebx,y       ; y -> ebx              (6)[4]
        imul    ebx,ebx     ; y*y -> ebx            (2)[2]
; Обчислити 2xy
        mov     ecx,x       ; x -> ecx              (6)[4]
        imul    ecx,y       ; x*y -> ecx            (6)[4]
        imul    ecx,2       ; 2*xy -> ecx           (3)[3]
; Скласти отримані часткові добутки
        add     eax,ebx     ; (x^2+y^2) -> eax      (2)[2]
        sub     eax,ecx     ; ()-2xy -> eax         (2)[2]
        mov     result,eax  ; зберегти результат    (6)[4]
        end     start
```

Оператори мовою Асемблер процесора H86 мають ту саму структуру, що й оператори процесора H51. Відзначимо відмінність мови Асемблер для цього процесора від мови Асемблер процесора H51. При заданні операндів у команді першим задається операнд-приймач dst, а другим – джерело src. У процесорі H86 усі директиви починаються з точки. Директива .long використовується для *присвоєння* комірці пам'яті символічного імені та *розміщення* в ній будь-якого числа. Для розміщення кожного числа відводиться пам'ять місткістю 32 біти (4 байти).

Перші три рядки програми, не ураховуючи необхідних коментарів на початку програми, містять директиви ".long", які виділяють три комірки пам'яті для трьох довгих чисел. Кожній комірці присвоюють відповідне ім'я, яке збігається з ім'ям змінних в обчислюваному виразі. Так, наприклад, комірка пам'яті із символічною адресою X зберігає число -47, яке має довжину 4 байти. Комірки пам'яті для зберігання проміжних даних не потрібні, оскільки для цього використовують регістри процесора.

Програма для процесора складається з тих же чотирьох фрагментів, у трьох з яких обчислюють часткові добутки, а у четвертому підсумовують отримані величини. Для зберігання кожного добутку використовують свій регістр: величина X^2 зберігається у регістрі EAX, значення Y^2 – у регістрі EBX, а значення $2 \bullet X \bullet Y$ – у регістрі ECX. Для обчислення значення X^2 число з цієї комірки пересилають у регістр EAX, а потім множать його на себе. Результат залишається у регістрі EAX. У такий же спосіб у регістрі EBX отримують значення Y^2, а в регістрі ECX – значення $2 \bullet X \bullet Y$. В останньому фрагменті обробляють отримані добутки. Спочатку складають значення двох квадратів, які зберігаються у регістрах EAX і EBX, а від отриманої суми віднімають значення добутку $2 \bullet X \bullet Y$, який знаходиться у регістрі ECX, і результат поміщають у регістр EAX. На закінчення результат зберігають у

комірці пам'яті з адресою RESULT. Як видно, програма стала коротшою і складається лише з 10 команд.

Розглянемо формати команд для процесора H86 (рис. 5.6).

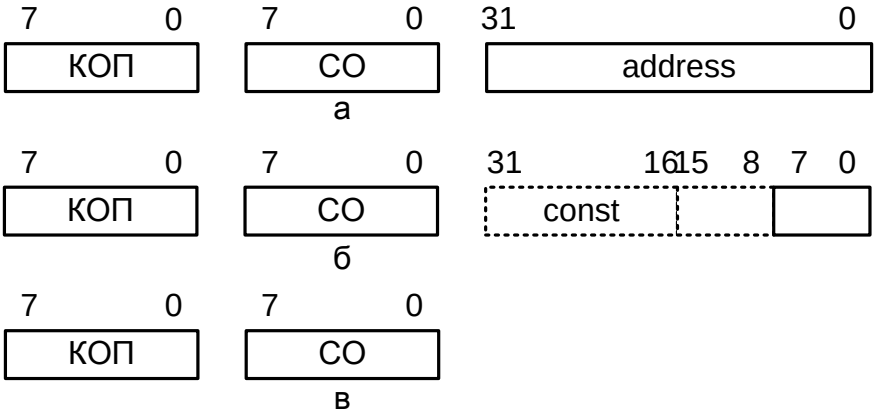


Рис. 5.6. Формати команд процесора H86

З програми видно, що можна отримати велику кількість комбінацій операндів у командах процесора. Тому для задання операндів використовують окремий байт специфікатора операнда (СО), в якому вказують кількість операндів, а також характер другого операнда, оскільки один операнд завжди зберігається в регістрі. Кілька бітів СО задають регістр, в якому розміщується один операнд. Решта бітів задають режим адресації другого операнда, який визначає, де зберігається операнд: у пам'яті, регістрі або в команді (якщо є константою). Якщо другий операнд – теж регістр, то в СО розміщуються номери обох регістрів. Таким чином, у випадку комбінації операндів "регістр-регістр" команда складається з коду операції і СО та має найменшу довжину – два байти (рис. 5.6, в).

Якщо другим операндом є комірка пам'яті, то одного СО недостатньо, – потрібна адреса операнда, який має довжину чотири байти і розташовується після СО (рис. 5.6, а). Команда, яка використовує операнд, що зберігається в пам'яті, завжди має довжину 6 байтів (1 + 1 + 4). Якщо другим операндом є константа, вона вказується в команді після СО (рис. 5.6, б) і може становити один, два або чотири байти. Для подання константи у команді вибирається мінімальна кількість байтів, достатня для подання числа. При використанні констант довжина команди може коливатися від трьох до шести байтів. Це зроблено для мінімізації довжини команди. У цій програмі використовується константа 2, для подання якої досить одного байта, і довжина всієї команди становить три байти. Довжина кожної команди вказана в кінці рядка кожної команди у круглих дужках.

Визначимо час виконання команд для процесора з РЗП, використовуючи наведену раніше методику. Команди, які використовують

операнди, розташовані у пам'яті (див. рис. 5.6, а), чотири рази звертаються до неї: для вилучення КОП, СО, 32-розрядної адреси, а також для читання операнда або запису у пам'ять результату. Процесор H86 дає можливість вибирати із пам'яті за одне звернення байт, 16-розрядне або 32-розрядне число. Команди, які використовують константу як операнд (див. рис. 5.6, а), три рази звертаються до пам'яті: для вибірки із пам'яті КОП і СО, а також для читання самої константи будь-якої довжини. Команди, які виконують операції типу "регістр-регістр", звертаються до пам'яті двічі: для читання КОП і СО. Час виконання кожної команди відповідно до вибраного показника вказується в кінці кожного рядка програми в квадратних дужках. Результати оцінювання тестової програми наведено в табл. 5.4.

Таким чином, програма для процесора з РЗП складається з 10 команд, займає в пам'яті 41 байт і звертається до пам'яті 31 раз.

5.3. Процесор, який має архітектуру зі стеком

Архітектуру процесора зі стеком розглянемо на прикладі гіпотетичного процесора H87, прототипом якого є арифметичний співпроцесор (СП) Intel 8087. Цей пристрій називається співпроцесором, оскільки може працювати тільки спільно з МП 8086. МП читає із пам'яті команду, яка надходить одночасно у МП і СП, після цього вона декодується кожним з них для того, щоб визначити, чия це команда. Якщо з'ясовується, що це команда СП, то він починає її виконувати, а МП – пропускає виконання цієї команди і переходить до наступної вибірки. Якщо ж прочитана команда належить МП, то він починає її виконувати, а СП – очікує наступну команду, яку прочитає із пам'яті МП.

Процесор H87 – однокристальний СП, який має архітектуру фон Неймана. СП H87 є 32-розрядним процесором, який має архітектуру зі стеком. У нього немає ні акумулятора, ні РЗП. Для зберігання даних і результатів СП використовує стек, розташований в основній пам'яті. Для адресації стеку СП використовує єдиний програмно-доступний регістр – 32-розрядний покажчик стеку ESP. Процесор H87 – безадресний, оскільки в його системі команд є тільки безадресні команди оброблення даних. Адресний простір цього СП становить 4 GB, і для адресації даних використовують 32-розрядні логічні адреси.

Структуру процесора H87 показано на рис. 5.7, а програмну модель процесора – на рис. 5.8.

На рис. 5.7 видно, що до структури найпростішого процесора додано регістр ESP – покажчик стеку (Stack Pointer). Покажчик стеку ESP

використовується для адресації даних, розташованих у стеку. ESP адресує вершину стеку, яка називається TOS (Top Of Stack). Другий зверху елемент стеку називається SOS (Second Of Stack). Стек знаходиться в основній пам'яті комп'ютера. Для того щоб виконати операції з даними в стеку, їх необхідно туди помістити.

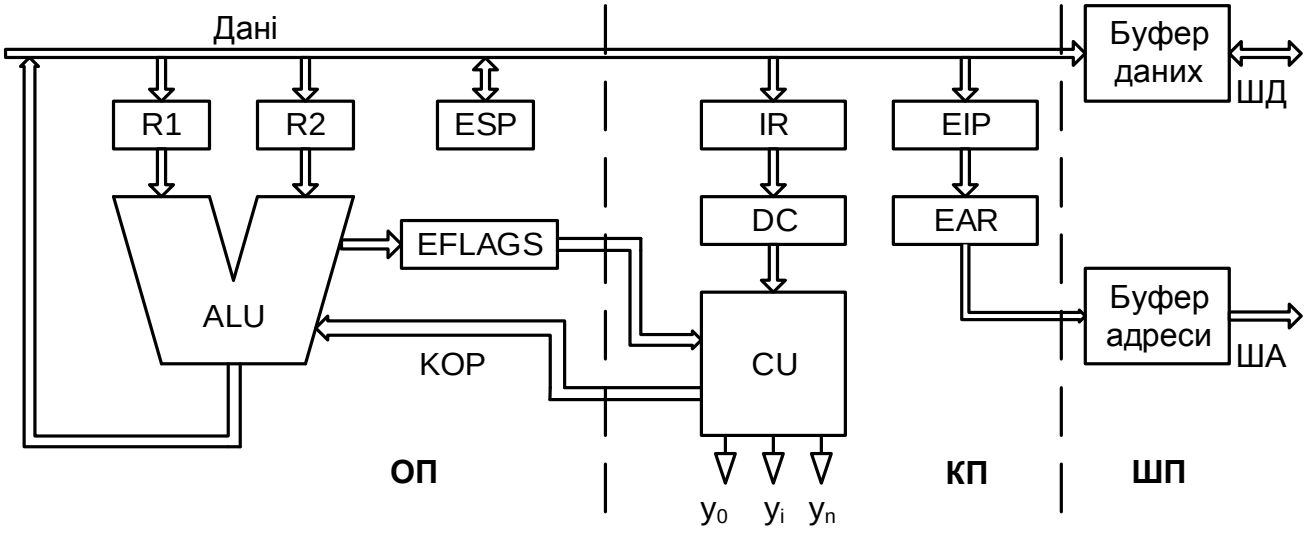


Рис. 5.7. Структура процесора H87

Програмна модель процесора H87 містить три програмно-доступних регістри (див. рис. 5.8): покажчик стеку ESP, регістр прапорців EFlags і покажчик команд EIP. Регістр EFlags містить ознаки результату, а покажчик команд EIP зберігає адресу наступної команди.

Процесор зі стеком складніший в програмуванні, ніж процесор з акумулятором або з РЗП. Він має досить незвичайну техніку програмування, яку можна освоїти, якщо зображати стан стеку для кожної команди.

Рис. 5.8. Архітектура процесора H87

Команди процесора H87, які використовують для обчислення тестового виразу, наведено в табл. 5.3. Усі команди процесора починаються з літери F, що є ознакою команд числового співпроцесора. Процесор застосовує

п'ять команд: завантаження у стек (FLD); видалення зі стеку (FSTP); складання (FADD); віднімання (FSUB); знакове множення (FMUL). Розглянемо виконання команд процесором зі стеком, зображуючи стан стеку до і після виконання кожної команди.

Таблиця 5.3

Мнемокод	Операнди	Коментар	Опис	Формат, рис. 5.15	Довжина, байти	Кількість пересилань
fld	src	sp←(sp-4) Mem[sp]←src	*Проштовхнути* вміст джерела у стек	а б	5 2-3-5	4 3
fstp	dst	dst← Mem[sp] sp ← (sp+4)	*Виштовхнути* вміст стеку у приймач	а	5	4
fadd		SOS←TOS+SOS	*Скласти* TOS і SOS. TOS виштовхнути	в	1	4
fsub		SOS←TOS-SOS	*Відняти* від вмісту TOS вміст SOS. TOS виштовхнути	в	1	4
fmul		SOS←TOS*SOS	*Помножити* TOS і SOS. TOS виштовхнути	в	1	4

Примітка. src – операнд-джерело (адреса комірки пам'яті або константа), dst – операнд-приймач (адреса комірки пам'яті), TOS – вершина стеку, SOS – другий зверху елемент стеку.

Команда завантаження вмісту приймача у стек FLD (LoaD – завантажити) спочатку зменшує на 4 значення покажчика стеку ESP, а потім поміщає вміст приймача в комірку пам'яті, на яку вказує ESP (рис. 5.9, а). Операндом, завантаженим у стек, може бути вміст комірки пам'яті або константа.

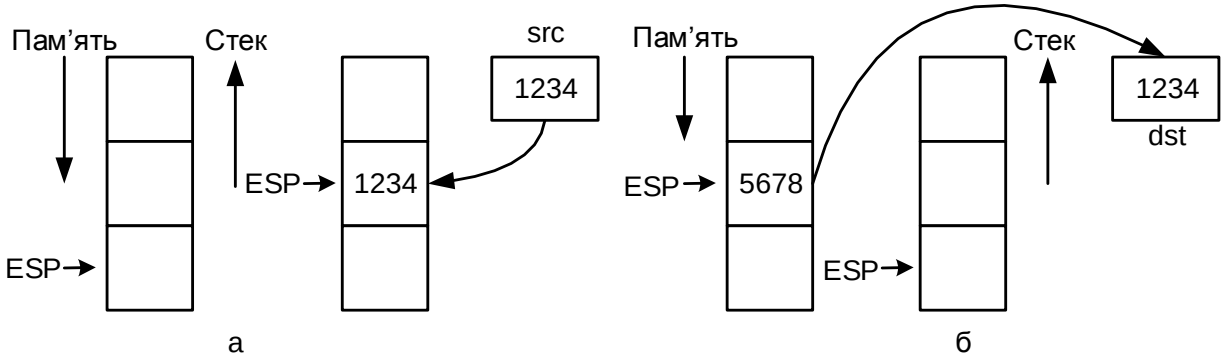


Рис. 5.9. Стан стеку до і після виконання команд процесора H87:
а – команда FLD dst; б – команда FSTP src

Команда збереження вмісту стеку FSTP (Store and Pop – зберегти і виштовхнути) спочатку зберігає вміст вершини стеку, на яку вказує ESP, у комірці пам'яті, а потім збільшує на 4 вміст покажчика стеку ESP (рис. 5.9, б). Таким чином, вміст вершини стеку очищується.

Розглянемо стан стеку після виконання таких команд (рис. 5.10):

```
fld  a    ; завантажити у стек вміст комірки A  (б)
fld  b    ; завантажити у стек вміст комірки B  (в)
fld  c    ; завантажити у стек вміст комірки C  (г)
fld  777  ; завантажити у стек значення константи (д)
```

Перед початком виконання команди завантаження FLD стек порожній (рис. 5.10, а), а комірки пам'яті містять деякі значення (рис. 5.10, е). Для завантаження першого операнда за допомогою команди FLD A значення покажчика стеку ESP *зменшується* на 4 і в стек *поміщається* 4-байтове (32-розрядне) число, розташоване у комірці пам'яті з символічною адресою A (див. рис. 5.10, а). Для виконання другої команди значення покажчика стеку ESP знову зменшується і в стек поміщається друге число з комірки пам'яті з адресою B. Після виконання третьої команди початковий стан стеку змінюється втретє, у стеку вже будуть зберігатися три числа, а покажчик стеку, як і раніше, адресує останнє завантажене у стек число, розташоване у вершині стеку TOS. Другим елементом є число B, яке було завантажено передостаннім. Оскільки перед кожним завантаженням значення покажчика стеку *зменшується*, то стек збільшується у напрямку молодших адрес пам'яті (вниз), а сама пам'ять – у напрямку старших адрес (вгору).

Команда вилучення вмісту стеку в приймач FSTP (STore and Popup – зберегти і виштовхнути) *спочатку поміщає* вміст верхнього елемента стеку TOS у комірку пам'яті, а *потім збільшує* на 4 значення покажчика стеку ESP (рис. 5.9, б). Як приймач операнда можна вказувати тільки вміст комірки пам'яті.

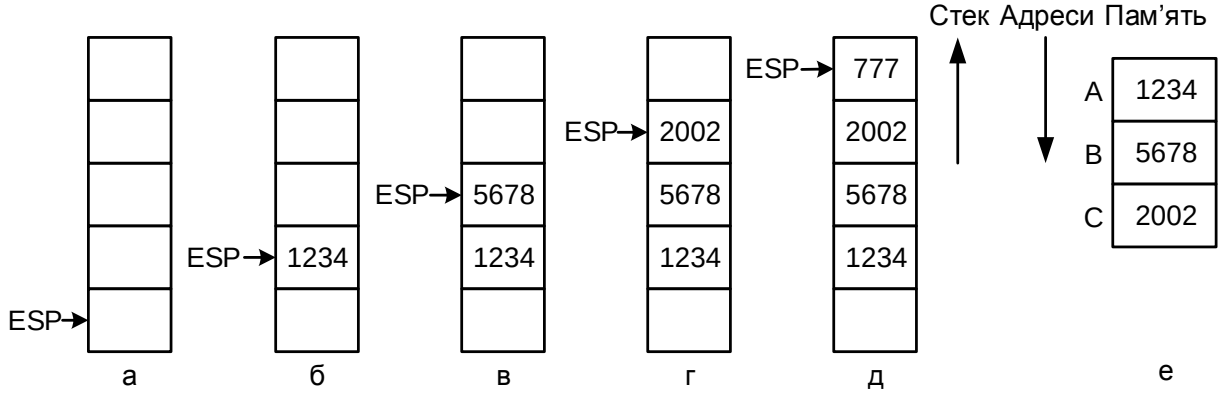


Рис. 5.10. Стан стеку після виконання серії команд FLD

Розглянемо стан стеку після виконання таких команд (рис. 5.11):

```
fstp a  ; виштовхнути число з вершини стеку в комірку A (б)
fstp b  ; виштовхнути число з вершини стеку в комірку B (в)
fstp c  ; виштовхнути число з вершини стеку в комірку C (г)
fstp d  ; виштовхнути число з вершини стеку в комірку D (д)
```

Перед початком виконання команд вилучення стек заповнений (рис. 5.11, а), а комірки пам'яті містять завантажені раніше значення (рис. 5.11, е). Вміст комірки пам'яті, в яку заноситься виштовхнуте зі стеку число, вказано під зображенням кожного стеку.

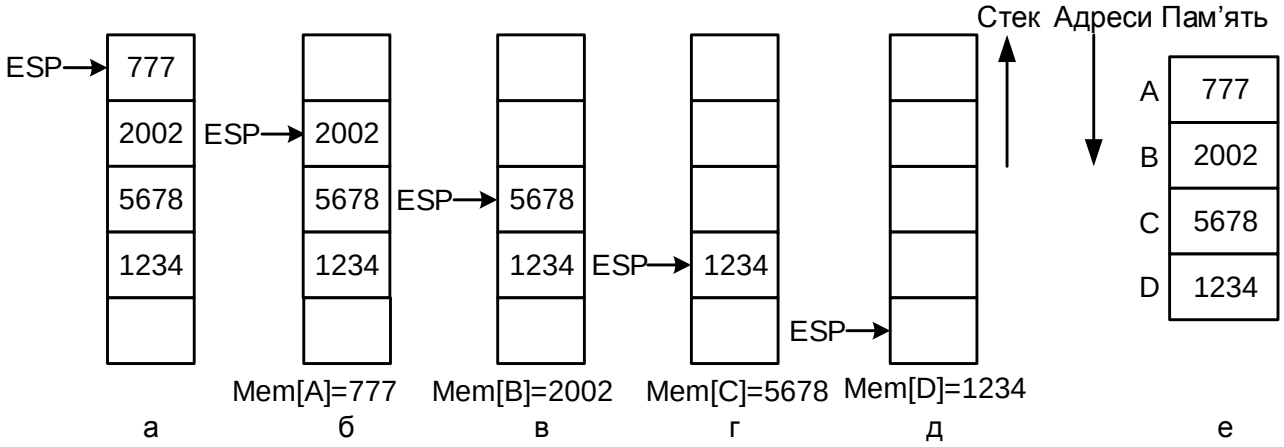


Рис. 5.11. Стан стеку після виконання серії команд FSTP

Для виштовхування із стеку першого операнда командою FSTP A число, яке зберігається у вершині стеку TOS і адресується регістром ESP, поміщається у комірку пам'яті із символічною адресою A. Після цього значення покажчика стеку ESP збільшується на 4 (див. рис. 5.11, а). Таким чином, унаслідок виконання першої команди у комірку пам'яті з адресою A буде занесено число 777. Після виконання другої команди FSTP B число 2002, яке знаходиться у вершині стеку, поміщається у комірку пам'яті з адресою B, а значення покажчика стеку ESP знову зменшується на 4, адресуючи таким чином нову вершину стеку (рис. 5.11, б). Після виконання третьої команди FSTP C виштовхнуте зі стеку число 5678 поміщається у комірку пам'яті з адресою C, а покажчик стеку ESP після зменшення його значення на 4 адресує тепер останнє число, яке зберігається в стеку (рис. 5.11, в). Після виштовхування із стеку останнього числа за допомогою команди FSTP D (рис. 5.11, г) стек очищається. Стан елементів пам'яті після виконання всіх команд виштовхування показано на рис. 5.11, е.

Арифметичні команди FADD, FSUB, FMUL процесора H87 є безадресними, оскільки вони не містять операндів. Як операнди ці команди використовують два останніх числа, завантажених у стек. Результатом операції буде одне число, поміщене на місце другого елемента стеку SOS, а колишнє значення, яке зберігалося у вершині стеку TOS, втрачається.

Виконання арифметичної команди FSUB процесора H87 показано на рис. 5.12.

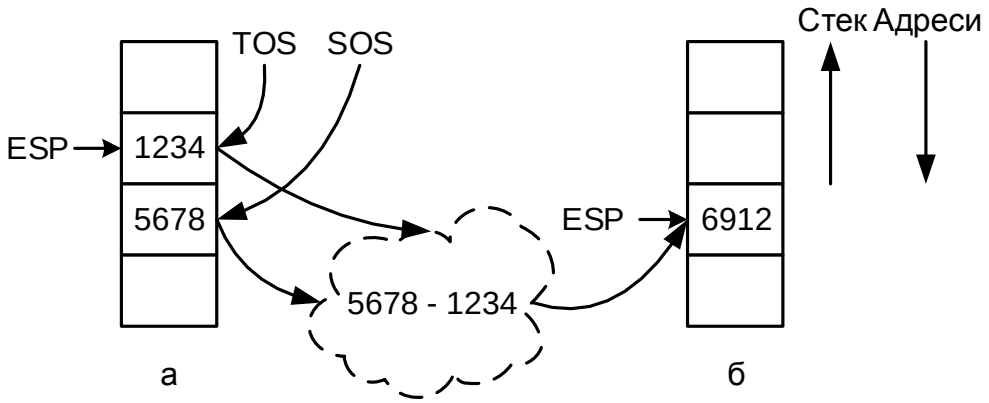


Рис. 5.12. Стан стеку при виконанні команди FSUB

Перед виконанням арифметичної команди FSUB у стек мають бути завантажені два операнди (рис. 5.12, а). Із стеку витягується другий елемент (число 5678), а потім – число, яке зберігається у вершині стеку (число 1234). Від першого числа віднімається друге і результат записується замість другого елемента стеку SOS, тобто замість числа 5678. Після цього значення покажчика стеку ESP збільшується на 4, пересуваючись у стеку на одну клітинку вниз (рис. 5.12, б). Команди множення FMUL і складання FADD виконуються аналогічно.

Мова Асемблер процесора H87 аналогічна мові Асемблер процесора H86. Для розуміння дій, виконуваних програмою для кожної команди, слід зображувати стан стеку після її виконання. Тому у кожному рядку програми, який містить команди процесора, у полі коментарів будемо зображати стек: спочатку – вміст вершині стеку TOS, потім – другий елемент стеку SOS та ін. Для більш наочного уявлення про динаміку виконання програми на рис. 5.13 показано стан стеку після виконання кожної команди.

Тепер розглянемо програму, за допомогою якої можна обчислити значення необхідного виразу:

```
; Обчислити арифметичний вираз
; RESULT = X^2 - 2XY + Y^2
;
x:      .long   -47 ; за адресою x зберігається перший операнд
y:      .long    63 ; за адресою y зберігається другий операнд
result: .long     0 ; зберегти результат за цією адресою
start:
;                       ; TOS          Рисунок  Довжина/Тривалість
; Обчислити x^2
        fld     x       ; x            {3.20а}       (5)[4]
        fld     x       ; x,x          {3.20б}       (5)[4]
```

```
        fmul            ; x*x          {3.20в}      (1)[4]
; Обчислити y^2
        fld     y       ; y,x^2        {3.20г}      (5)[4]
        fld     y       ; y,y,x^2      {3.20д}      (5)[4]
        fmul            ; y*y,x^2      {3.20е}      (1)[4]
        fadd            ; (y^2+x^2)    {3.20ж}      (1)[4]
; Обчислити 2xy
        fld     x       ; x,()         {3.21а}      (5)[4]
        fld     y       ; y,x,()       {3.21б}      (5)[4]
        fld     2       ; 2,y,x,()     {3.21в}      (2)[3]
        fmul            ; 2*y,x,()     {3.21г}      (1)[4]
        fmul            ; x*2y,()      {3.21д}      (1)[4]
        fsub            ; ()-2xy       {3.21е}      (1)[4]
; Зберегти результат
        fstp    result  ; зберегти результат        (5)[4]
        end     start
```

У перших трьох рядках програми призначено символічні адреси X, Y і RESULT. У перших двох комірках розміщено вихідні числа, а останню – зарезервовано для майбутнього результату. Комірки пам'яті для зберігання проміжних даних не потрібні, оскілки для цього використовують стек.

Частина програми, яка містить команди процесора і розташована після мітки START, складається з трьох фрагментів. У перших двох фрагментах обчислюють значення квадратів змінних X і Y, результати яких додають один до одного. Сума залишається у стеку. Третій фрагмент обчислює добуток 2•X•Y, який згодом віднімають від суми квадратів, яка раніше зберігалася у стеку.

Спочатку двічі проштовхують у стек вміст комірки пам'яті X, а потім за допомогою команди FMUL перемножують два верхні елементи стеку (рис. 5.13, а-в). Після цього отримане значення квадрата числа X залишаються в стеку. Наступні значення завантажують у стек поверх цього числа. Значення Y^2 отримують аналогічно (рис. 5.13, г-д). Після цього два верхні елементи стеку складають за допомогою команди FADD (рис. 5.13, е-ж). Отримана сума залишається у вершині стеку TOS.

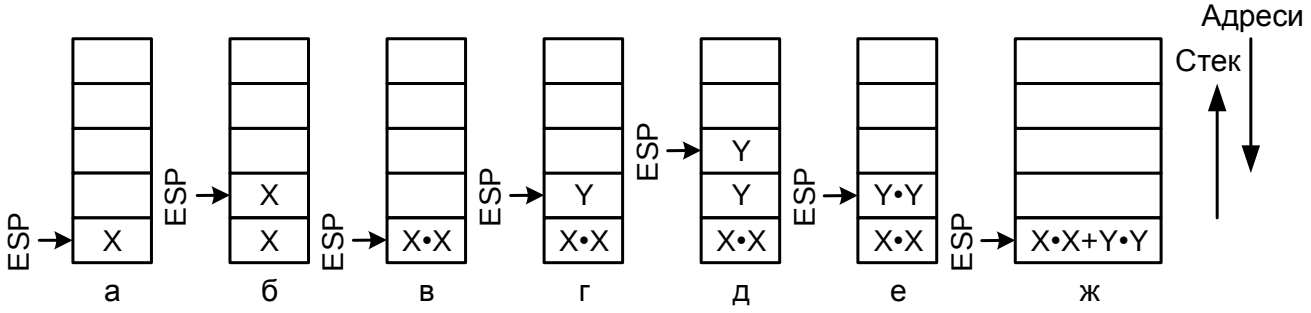


Рис. 5.13. Стан стеку при виконанні програми процесором H87

Значення добутку 2•X•Y обчислюють таким самим чином. Спочатку поміщають у стек операнд X, потім – операнд Y, а потім – константу 2 (порядок завантаження у стек у цьому випадку не має ніякого значення) (рис. 5.14, а-в). Слід зазначити, що в цій програмі після завантаження у стек константи 2 в ньому зберігаються чотири числа. Після цього перемножують три верхніх елемента у стеку: спочатку отримують добуток 2 • Y (рис. 5.14, г), а потім його множать на X (рис. 5.14, д). Тепер у стеку зберігається два числа: у вершині стеку TOS знаходиться добуток 2•X•Y, а другим елементом стеку є сума квадратів. Після використання команди FSUB у вершині стеку отримують таке значення виразу: X^2 - 2•X•Y + Y^2 (див. рис. 5.14, д). Віднімання виконують саме так: від другого елемента стеку SOS віднімають вміст вершини стеку TOS. Тепер має бути зрозуміло, що порядок виконання операцій було вибрано не випадково – значення, *яке мали відняти*, було отримано останнім, щоб воно знаходилося у вершині стеку TOS, як того потребує команда FSUB. Завдяки виштовхуванню із стеку результату і збереження його у пам'яті за адресою RESULT стек очищається. Правильна техніка програмування у стековому процесорі передбачає виконання важливого правила – після завершення програми необхідно очистити стек. Неухильне дотримання цієї вимоги ніколи не призведе до переповнення стеку і некоректної роботи інших програм через недбалість програміста.

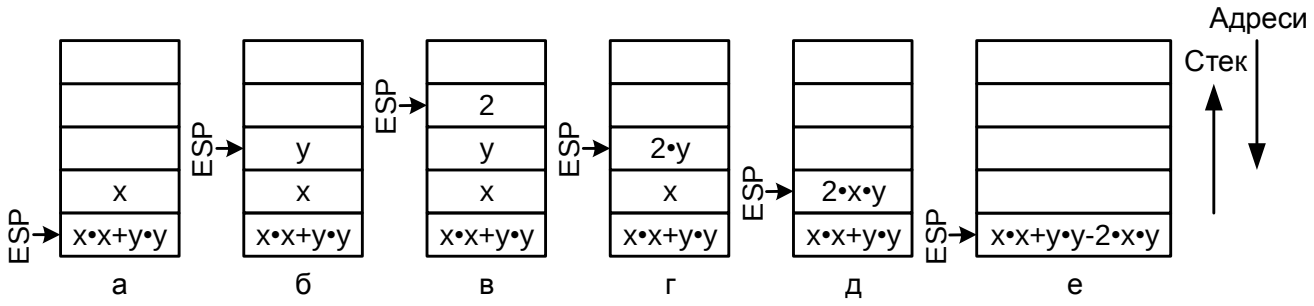


Рис. 5.14. Стан стеку при виконанні другої половини програми процесором H87

Розглянемо формати команд для процесора H87 (рис. 5.15). З аналізу програми видно, що команди процесора H87 мають набагато менше можливостей для задання операнда.

Команда виштовхування даних зі стеку FSTP дає змогу зберегти дані у пам'яті за вказаною адресою (див. рис. 5.15, а). Ця команда складається з коду операції КОП (один байт) і адреси комірки пам'яті SRC (чотири байти). Такий самий формат має команда завантаження операнда у стек із пам'яті FLD SRC.

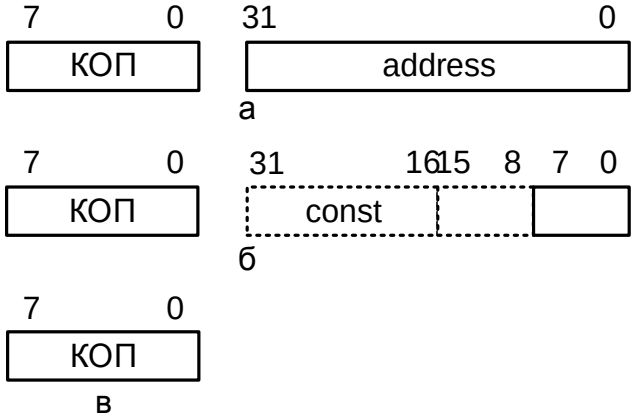


Рис. 5.15. Формати команд процесора H87

Якщо у команді завантаження FLD операнд є константою, то команда має формат, зображений на рис. 5.15, б, при цьому довжина команди залежить від довжини операнда. Якщо константа становить 1 байт, то команда має довжину два байти: один байт – КОП і один байт – сама константа. Якщо константа має довжину 16 бітів, то команда має довжину три байти, і при використанні 32-розрядної константи – п'ять байтів. У цій програмі використовується константа 2, для подання якої досить одного байта, і тому довжина команди становить два байти.

Усі арифметичні команди не містять операндів, оскільки оперують з даними, розташованими в стеку (безадресні команди). Такі команди мають простий однобайтовий формат (рис. 5.15, в). Довжина кожної команди вказана в кінці рядка кожної команди у круглих дужках.

Визначимо швидкодію команд для стекового процесора за кількістю звернень до пам'яті при виконанні кожної команди. При цьому вважаємо, що процесор здатний за одне звернення до пам'яті витягти 8, 16 або 32 біти. Час виконання кожної команди визначається часом, потрібним для отримання кожної команди з пам'яті (етап Fetch), і часом виконання кожної команди (етапи Operands і Execute). Час вибірки команди з пам'яті визначається її форматом, а час виконання – логікою її роботи. Найповільнішими командами є команди завантаження у стек FLD і виштовхування із стеку FSTP. Для вибірки команди з пам'яті (див. рис. 5.15, а) необхідно двічі звернутися до неї: один раз для отримання коду операції КОП і один раз для отримання адреси операнда, розташованого після КОП. Після дешифрування КОП процесор переходить до виконання команди, для чого читає вміст зазначеної комірки пам'яті (одне звернення до пам'яті) і записує це саме число у комірку пам'яті, яка адресується покажчиком стеку ESP (ще одне звернення до пам'яті). Таким чином, команди FLD і FSTP, які використовують як операнд адресу комірки пам'яті, працюють повільно, оскільки звертаються до пам'яті *чотири* рази.

Команда FLD може мати як операнд константу. Для вибірки команди із пам'яті слід двічі звернутися до неї. Після декодування КОП необхідно константу, яка є частиною команди і вже знаходиться у процесорі, занести в комірку пам'яті, яка адресується покажчиком стеку ESP, тобто у вершину стеку TOS. Таким чином, для виконання команди, яка використовує як операнд константу, потрібно *тричі* звернутися до пам'яті.

Найбільш швидко виконуються безадресні команди оброблення даних FADD, FSUB і FMUL. Для розуміння цих команд необхідно врахувати таке: для виконання деякої операції операнди повинні знаходитися в процесорі. Тому для виконання арифметичної операції потрібно прочитати із пам'яті перший байт команди, який містить КОП, і декодувати його (одне звернення до пам'яті). Для виконання арифметичної команди слід вибрати із пам'яті два операнди (етап Operands) за два звернення до пам'яті, виконати в АЛП відповідну арифметичну операцію і зберегти у пам'яті результат (етап Save), звертаючись один раз до пам'яті. Таким чином, для виконання коротких однобайтових безадресних команд необхідно звернутися до пам'яті *чотири* рази! Це пояснюється тим, що операнди і результат зберігаються у стеку, розташованому в основній пам'яті. Час виконання кожної команди відповідно до вибраного критерію вказаний у кінці кожного рядка програми у квадратних дужках.

Таким чином, програма для процесора зі стеком складається з 14 команд, займає у пам'яті 43 байти і звертається до пам'яті 55 разів.

5.4. Порівняльний аналіз архітектур процесорів

Раніше було розглянуто три архітектури процесорів. Оцінимо отримані результати (табл. 5.4).

Таблиця 5.4

Архітектура процесора	Кількість команд	Довжина програми, байти	Кількість пересилань
З акумулятором	17	27	37
З РЗП	10	41	31
Зі стеком	14	43	55

Найбільш коротку програму має процесор з РЗП (10 команд), мінімальну пам'ять займає програма для процесора з акумулятором (27 байтів), найбільш швидко працює програма для процесора з РЗП (31 звернення до пам'яті). Найбільшу кількість команд має програма для процесора з акумулятором (17 команд), найбільшу пам'ять займає програма

для процесора з РЗП (43 байти), найбільш повільно працює програма для процесора зі стеком (55 звернень до пам'яті). Не найкращим є процесор зі стеком, але у нього є перевага – короткі команди оброблення даних. Найкращим процесором можна вважати процесор з РЗП, оскільки він має найменшу кількість команд у програмі й працює найшвидше.

Характеристики процесора значною мірою визначаються особливостями роботи процесора з пам'яттю. Необхідність роботи з пам'яттю впливає на формат команд, їх довжину, а також кількість звернень до пам'яті. Швидкодія процесорів знижується насамперед через часті звернення до пам'яті. Велика місткість пам'яті, яка займає програма, пояснюється необхідністю вказувати у програмі довгі 32-розрядні адреси пам'яті. Слід звернути увагу на те, що при зменшенні розміру адрес пам'яті до 8 бітів для процесора з акумулятором місткість займаної пам'яті зменшилася майже в два рази (27 байтів у процесора з акумулятором і 41 байт у процесора з РЗП). Таким чином, зменшення кількості звернень до пам'яті підвищує швидкодію програми. Зменшення кількості адрес пам'яті, які задають у командах, дає змогу зменшити місткість зайнятої пам'яті, але при цьому збільшується кількість пересилань для доступу до даних.

Як видно, ідеальної архітектури не існує. Кожна архітектура має свої переваги і недоліки, а будь-яке інженерне рішення є компромісом між декількома часто-густо протилежними критеріями вибору: простота – гнучкість, зручність програмування – швидкодія, продуктивність – вартість. А при виборі певних інженерних рішень завжди доводиться віддавати перевагу одним показникам перед іншими для досягнення своєї мети, яка полягає у виявленні та виконанні вимог певного сегменту ринку.

Один із підходів задовольнити різним (часто протилежним) вимогам полягає у поєднанні в одному процесорі кількох архітектур в надії одержати нову якість процесора. Тому в одному процесорі розробники зазвичай намагаються поєднати переваги всіх архітектур. Оцінимо вдосконалення, які потрібно внести в архітектуру кожного розглянутого гіпотетичного процесора, щоб отримати архітектуру його реального прототипу.

Гіпотетичний процесор H51 має архітектуру з акумулятором, перевагою якої є короткі команди і мала місткість пам'яті, займаної програмою. Швидкодію процесора знижує велика кількість звернень до пам'яті, проте цей недолік можна компенсувати шляхом застосування гарвардської архітектури МП з окремими шинами для передачі даних і команд. Додавання до архітектури процесора H51 восьми РЗП і апаратного стеку дає змогу зменшити довжину команд, місткість займаної пам'яті й підвищити швидкодію програм. Для використання апаратного стеку до архітектури процесора потрібно додати покажчик стеку SP, при цьому стек буде складатися з восьми регістрів (за замовчуванням), розташованих на

кристалі процесора. Таким чином можна отримати програмну модель процесора однокристального МК Intel MCS-51 (рис. 5.16).

Реальний однокристальний МК Intel MCS-51 має чотири банки з восьми РЗП. Регістри в цьому процесорі є частиною пам'яті. Заміна адрес пам'яті іменами регістрів дає можливість зменшити довжину програми. Таким чином, доповнення архітектури процесора з акумулятором регістрами і апаратним стеком дало змогу створити процесор, який зручно програмувати. Його програми компактні й швидкі.

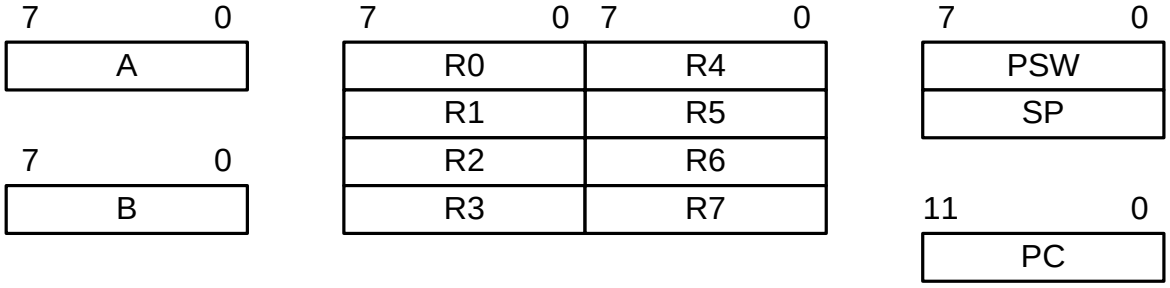


Рис. 5.16. Архітектура МК Intel MCS-51

У літературі, присвяченій процесору MCS-51, прийнято такі позначення програмно-доступних регістрів:

– PSW (Processor Status Word – слово стану процесора) – аналог регістра прапорців Flags;

– PC (Program Counter – лічильник команд) – 12-розрядний регістр, який адресує пам'ять команд місткістю 4KB, аналог регістра IP;

– R0 до R7 – регістри загального призначення.

Стековий процесор H87 має невисоку швидкодію через те, що його стек розташований в основній пам'яті, і навіть для виконання коротких безадресних арифметичних команд процесор змушений тричі звертатися до основної пам'яті. Водночас у цього процесора багато коротких безадресних команд, що дає змогу отримати невелику програму.

Як показала практика, при виконанні програми у стеку розташовується незначна кількість даних (у розглянутій програмі використовувалися чотири числа). Тому підвищити швидкодію стекового процесора можна шляхом розміщення стеку на його регістрах, організованих у вигляді стеку. У числовому співпроцесорі Intel x87, який є реальним прототипом гіпотетичного процесора H87, використовується саме таке рішення. Цей співпроцесор має вісім регістрів R0 – R7, які організовані в циклічний стек з елементами ST0 – ST7. Вершина стеку називається ST0, другий елемент стеку – ST1 та ін. Для адресації регістрів стеку використовується трирозрядний програмно-доступний регістр SP. Число, яке зберігається у

покажчику стеку, позначає номер регістра, який в даний момент є вершиною стеку. Передача даних між СП і пам'яттю здійснюється цілочисловим процесором, який є провідним, тоді як СП – веденим. Таким чином можна об'єднати короткі команди стекового процесора з високою швидкодією регістрового процесора. Внутрішній стек числового СП показано на рис. 5.17. На рисунку зображено стан покажчика стеку SP після ініціалізації СП.

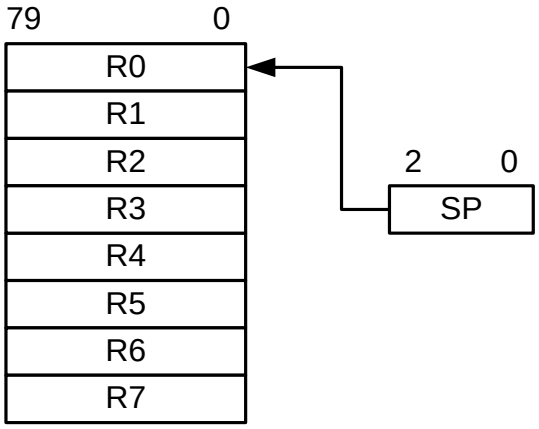


Рис. 5.17. Внутрішній стек співпроцесора Intel x87

Розглянемо зміни гіпотетичного процесора H86. Хоча процесор з РЗП є найкращим серед інших процесорів, його архітектуру також можна поліпшити (рис. 5.18).

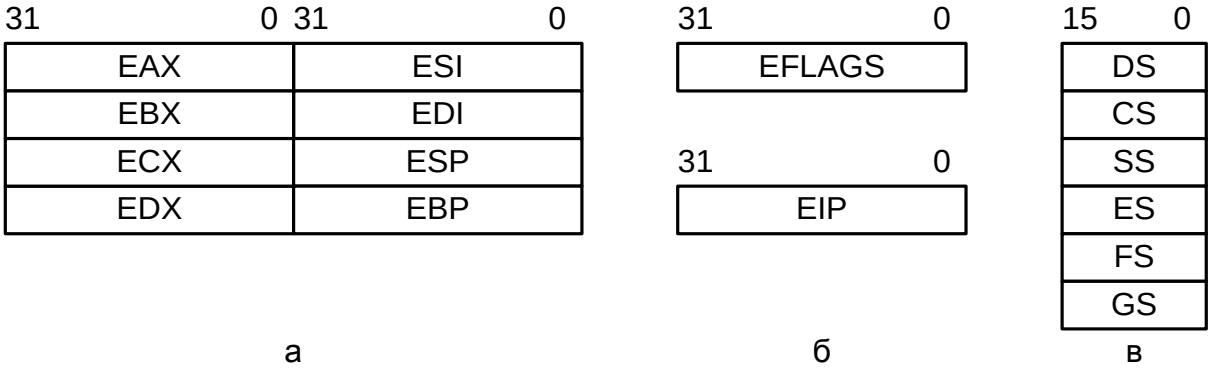


Рис. 5.18. Програмна модель МП Intel x86

Перш за все, до архітектури МП Intel x86 можна додати апаратний стек і регістри ESP і EBP для доступу до нього. Регістр EBP (Base Pointer – покажчик бази) буде використовуватися для адресації даних усередині стеку, регістр ESP – для адресації вершини стеку TOS. Унаслідок цього архітектура процесора матиме вісім РЗП: EAX, EBX, ECX, EDX, ESI, EDI,

ESP і EBP (рис. 5.18, а). МП, як і раніше, буде мати покажчик команд EIP і регістр прапорців EFLAGS (рис. 5.18, б). Імена всіх розглянутих регістрів будуть починатися з літери E (Extended – розширений), нагадуючи програмісту про те, що ці регістри є 32-розрядними. До програмної моделі буде входити також група з шести 16-розрядних сегментних регістрів DS, CS, SS та ін. (рис. 5.18, в), які будуть брати участь у формуванні адреси операнда.

Вище було розглянуто зміни програмної моделі гіпотетичного процесора H86. Однак його система команд має властивості, характерні для процесора з акумулятором. Реальний МП x86 має велику кількість безадресних команд, які використовуються порівняно рідко. Тому розробники процесора обмежили можливості задання операндів – дані повинні розташовуватися в акумуляторі або інших визначених регістрах. Майже всі такі команди складаються тільки з коду операції і виконуються швидко. До таких командам належать команди ділення DIV, IDIV, збільшення довжини числа CBW, CWD, CWDE, CDQ, десяткової корекції DAA, DAS, AAA, AAS, AAM, AAD, ланцюгові команди MOVS, LODS, STOS, CMPS, SCAS, INS, OUTS, введення і виведення IN, OUT. Таких команд досить багато. Їх використання дає змогу зменшити довжину програми шляхом деякого зниження продуктивності.

Загалом усі РЗП МП є рівноправними, і програміст може використовувати будь-які регістри у всіх командах без обмежень. Однак якщо як регістр використовується акумулятор, то довжина деяких команд зменшується на один байт або взагалі становить один байт.

Таким чином, МП Intel x86 мають архітектуру з РЗП, використовують апаратний стек і елементи архітектури з акумулятором.

5.5. Неархітектурні характеристики процесорів

Неархітектурними називаються характеристики, які не впливають на сумісність програм на рівні машинних кодів процесора, а визначають продуктивність і вартість. Головною неархітектурною характеристикою процесора є продуктивність.

Продуктивність – це кількість обчислювальних операцій, які виконуються за одиницю часу всіма наявними у процесорі обробними пристроями. Продуктивність залежить від того, які саме дані обробляє процесор – цілі або дійсні числа, і вимірюється у мільйонах операцій за секунду. Продуктивність процесора при обробленні цілих чисел вимірюється в MIPS, а продуктивність процесора при обробленні дійсних чисел – у MFLOPS.

1 MIPS = 1 Million Instructions Per Second = 1 мільйон операцій з цілими

числами за секунду.

1 MFLOPS = 1 Million Floating Point operation Per Second = 1 мільйон операцій з дійсними числами за секунду.

Продуктивність процесора залежить від архітектури і структури процесора. Вирізняють пікову, реальну та інтегральну теоретичну продуктивність.

Пікова продуктивність показує граничну швидкодію процесора за умови, що він обробляє нескінченну послідовність команд, а дані й команди розташовані у внутрішній кеш-пам'яті, тобто процесор не звертається до зовнішньої пам'яті. Пікова продуктивність абсолютно не залежить від характеру виконуваної програми і визначається довжиною розрядної сітки процесора, кількістю обробних пристроїв, тактовою частотою процесора.

Складова теоретична продуктивність (СТП) аналогічна піковій продуктивності й показує граничну швидкодію процесора, яка залежить тільки від характеристик його апаратних засобів. СТП визначається тактовою частотою процесора, набором функціональних пристроїв, пропускною здатністю внутрішніх шин, розрядністю регістрів і т.ін. Одиницею вимірювання СТП є MTOPS.

1 MTOPS = 1 Million Theoretical Operations Per Second = 1 мільйон теоретичних операцій за секунду.

Реальна продуктивність показує швидкодію процесора при виконанні програми у реальних умовах. При оцінюванні реальної продуктивності не робиться ніяких припущень щодо довжини програми, місця розташування даних і команд. Реальна продуктивність процесора *залежить* від характеру програми і оброблюваних даних і може бути істотно нижчою від пікової продуктивності й СТП.

Для оцінювання продуктивності різних комп'ютерів використовують завдання, характерні для тієї чи іншої області застосування обчислювальної техніки. За результатами виконання програм обчислюють *індекс продуктивності* досліджуваного комп'ютера, який є відносною оцінкою для порівняння двох комп'ютерів або обчислювальних систем.

Підвищення продуктивності процесорів досягають шляхом збільшення кількості обробних пристроїв, забезпечення їх максимального завантаження, підвищення тактової частоти процесора. Для підвищення продуктивності всього комп'ютера збільшують пропускну здатність системної магістралі, зменшують час доступу до пам'яті, збільшують швидкодію зовнішніх накопичувачів інформації. Сучасні МП містять велику кількість обробних пристроїв. Так, МП Alpha 21264 містить два блоки операцій з плаваючою точкою і чотири цілочислових обчислювальних пристрої. МП AMD K6 містить два цілочислових АЛП, блок обчислень операцій з плаваючою точкою і блок мультимедійних команд. Ефективне

завантаження обробних пристроїв, що паралельно функціонують, забезпечується або апаратурою процесора (МП Intel Pentium і МП Alpha), або компілятором, на вхід якого надходять програми будь-якою мовою програмування (Itanium 2), або спільно апаратурою та компілятором.

5.6. Літографія

Процесор є складним цифровим пристроєм, який складається з логічних елементів. Логічні елементи складаються з транзисторів, виконаних за напівпровідниковою технологією. Кожен логічний елемент реалізований у вигляді декількох транзисторів. Отже, процесор складається з мільярдів транзисторів.

Сировиною для напівпровідникової технології є кремній, який у процесі очищення доводять до стану 99.9999999% Si. Процес виготовлення процесора відбувається таким чином: з розплавленого кремнію вирощують монокристал циліндричної форми, потім його ріжуть на пластини, поверхню яких ретельно шліфують і полірують. Після цього на поверхні кремнієвих пластин методами фотолітографії і травлення створюють транзистори, які об'єднують в логічні схеми, що становлять процесор.

Фотолітографія – процес виборчого травлення поверхневого шару з використанням фотошаблону. Технологія побудована за принципом "світло – шаблон – фоторезист" і має такі етапи:

– нанесення на кремнієву підкладку шару матеріалу, з якого потрібно сформувати рисунок;

– нанесення на нього фоторезисту – шару світлочутливого матеріалу, який змінює свої фізико-хімічні властивості при опроміненні світлом;

– виконання експонування, тобто освітлення фотошару протягом точно встановленого проміжку часу через фотошаблон;

– видалення відпрацьованого фоторезиста.

Так формуються комірки, що являють собою транзистори. Для збільшення кількості транзисторів у процесорі потрібно зменшувати їх фізичний розмір, що призводить до збільшення їх щільності. Так, перехід з 32 нм на 22 нм технологію виробництва дав змогу розмістити на кристалі не 4, вже 8 ядер процесора.

До інших ефектів, які впливають на зменшення розміру транзистора, можна віднести:

– зменшення затримок при поширенні сигналів в електронних схемах на високих частотах (до 3 GHz), на яких працюють сучасні процесори;

– зменшення енергоспоживання і виділення тепла;

– зменшення паразитної місткості, тому що кожен транзистор являє собою маленький конденсатор.

Надалі замість терміна "літографія" будемо застосовувати термін "розмір транзистора", припускаючи, що вони є синонімами.

5.7. Закон Мура

Підвищення продуктивності МП насамперед відбувається завдяки збільшенню кількості обробних пристроїв і підвищенню тактової частоти процесора. Збільшення кількості обробних пристроїв потребує збільшення кількості вентилів (транзисторів), розміщених на кристалі МП. Тенденції в цій області були помічені одним із засновників компанії Intel Гордоном Муром у 1965 році. У 1975 році він вніс корективи у свій закон, і в 1985 році закон Мура набув точного формулювання, яке підтверджуються розвитком напівпровідникової техніки протягом 50 років, починаючи з 1965 року .

Закон Мура оснований на емпіричному спостереженні, зробленому Гордоном Муром, який виявив закономірність, що нові моделі мікросхем розробляються через приблизно однакові періоди часу (18 – 24 місяці) після розроблення їх попередників. При цьому кількість транзисторів, що використовуються в них, зростала приблизно в два рази (рис. 5.19).

Суть *закону Мура* в сучасному трактуванні полягає в такому: кількість транзисторів на одиниці поверхні подвоюється кожні 18 місяців.

Закон Мура дає змогу пояснити багато тенденцій в еволюції сучасних процесорів. Так, перший мікропроцесор Intel 4004 (1971 рік) мав лише 2300 транзисторів. У 2015 році кількість транзисторів у процесорах Intel становила вже 1,3 мільярда. Таким чином, кількість транзисторів за більш ніж 40 років збільшилася в 595 000 разів. Це один з найбільш швидко зростаючих показників у комп'ютерних технологіях, який зберігається вже понад 50 років [8].

Зростання кількості транзисторів на кристалі процесора забезпечується двома факторами: збільшенням розміру кремнієвої пластини, на яку наносять транзистори, і зменшенням розміру транзистора. Однією з головних тенденцій розвитку процесорів є зменшення розміру транзисторів при виробництві процесорів. Згідно з графіком відбувається зменшення проектних норм мікропроцесорів фірми Intel в середньому кожні 1,4 року в 2 рази (рис. 5.20). Так, у 1969 році розмір транзистора на кристалі становив 3 мкм, у 1983 р. – 1,5 мкм, у 1988 р. – 1 мкм, у 2001 р. – 130 нм, у 2005 р. – 65 нм.

На рис. 5.21 збільшення кількості транзисторів за роками показано за допомогою логарифмічної шкали. На рис. 5.22 показано динаміку зменшення розміру транзистора, якщо існуючі тенденції зберігатимуться в майбутньому.

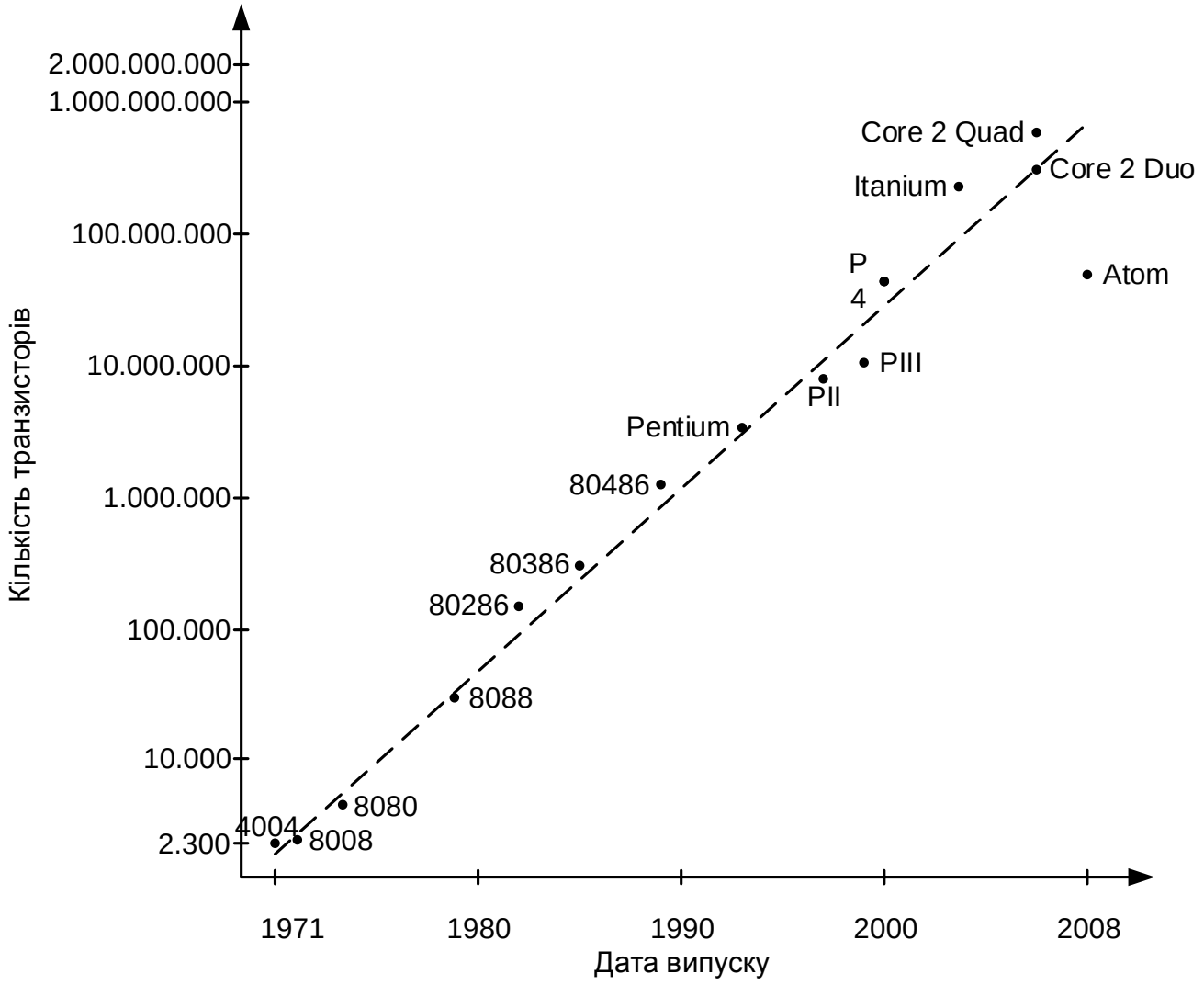


Рис. 5.19. Збільшення кількості транзисторів на кристалі процесора

Експерти вважають, що тенденція зменшення розміру комірки при виготовленні процесора, може істотно загальмуватися, оскільки подолання рубіжу в 7 нанометрів і менше пов'язано з багатьма проблемами. Квантові процеси і значне ускладнення технологій призводять до необхідності витрачати більше ресурсів і коштів на дослідження і доведення технології до досконалості.

Зі збільшенням кількості транзисторів спостерігається також тенденція зменшення вартості транзистора на кристалі процесора. Вартість транзистора зменшується в середньому в 2 рази кожні 1,3 року. Для споживача це означає можливість з часом купувати більш потужний процесор за ті ж самі гроші.

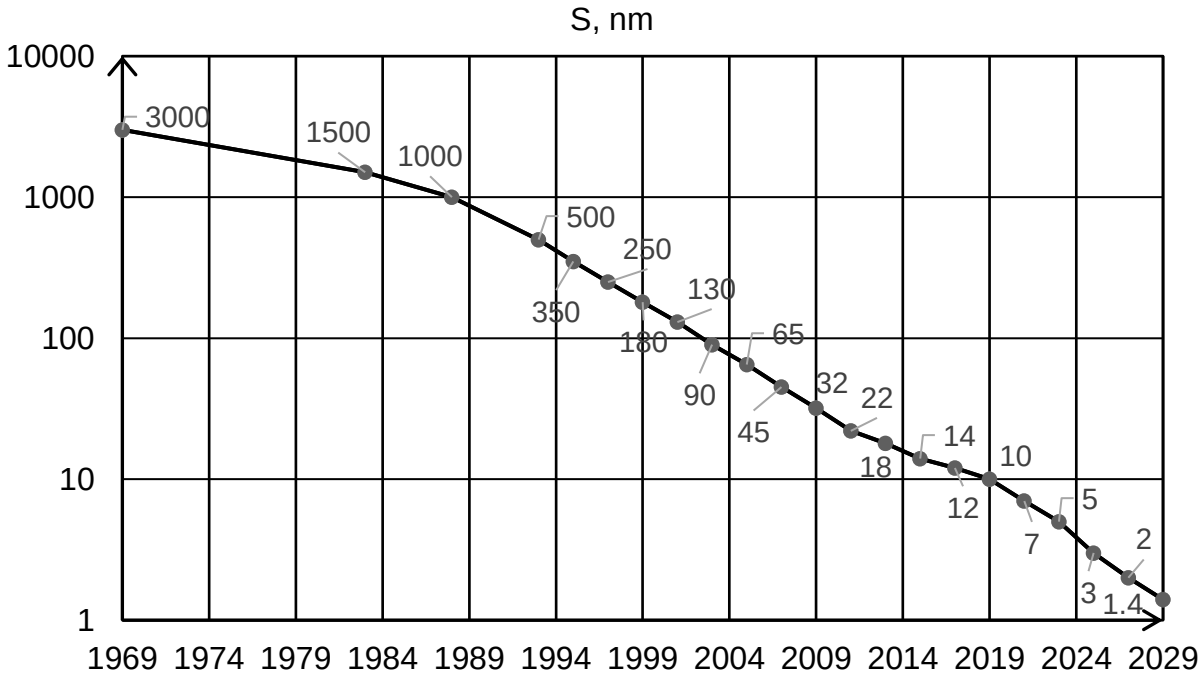


Рис. 5.20. Динаміка зміни розміру транзистора в технологіях,
що використовуються при виготовленні процесора

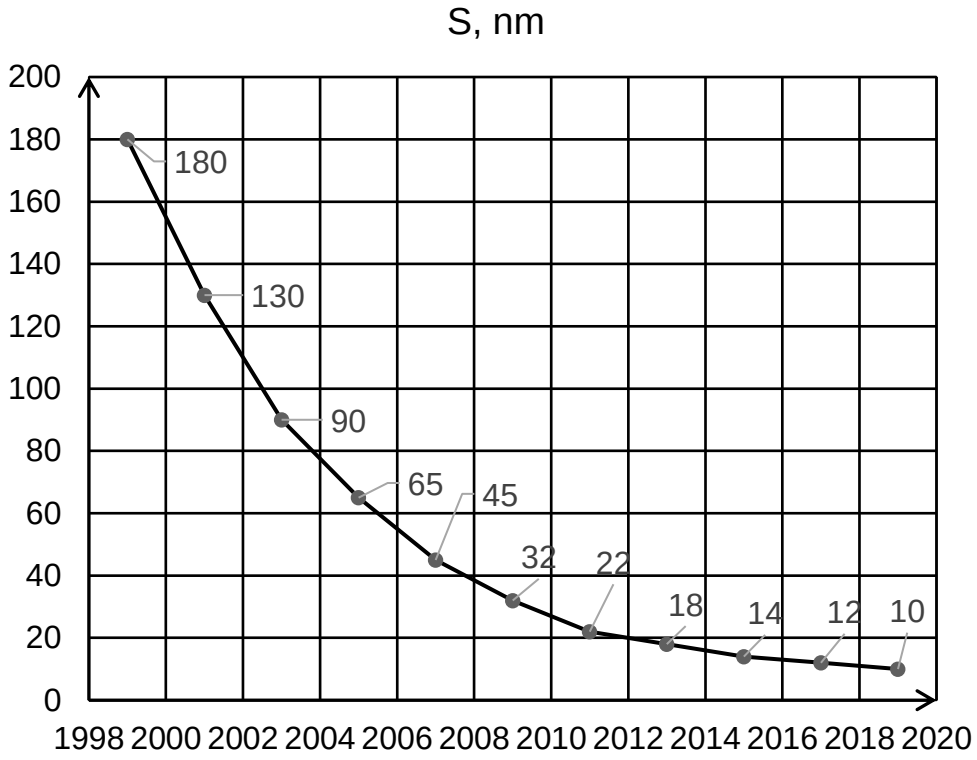


Рис. 5.21. Експоненціальне зменшення розміру транзистора в технологіях,
що використовуються при виготовленні процесора з 1998 року

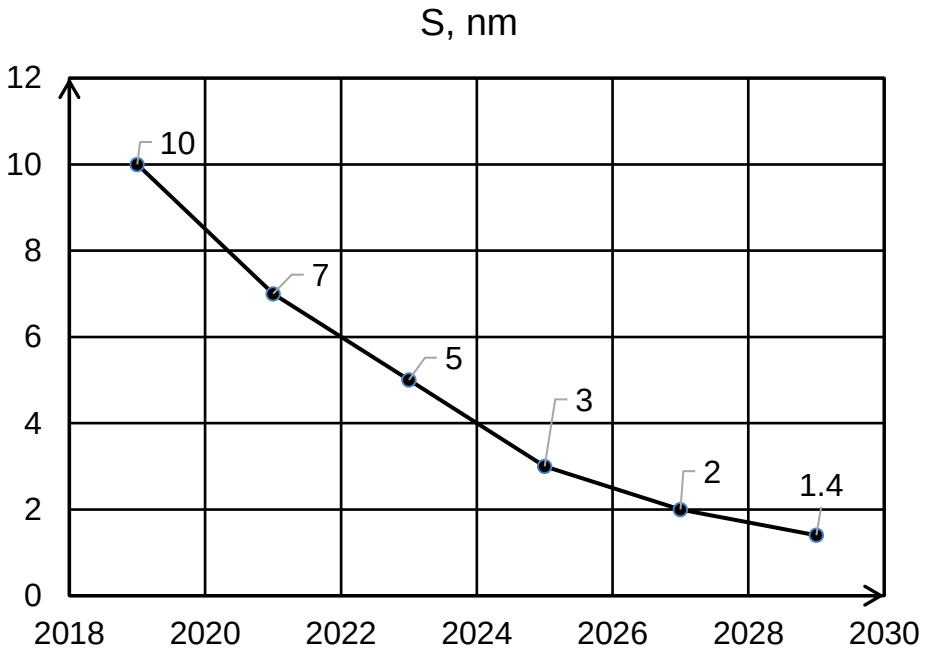


Рис. 5.22. Зменшення розміру транзистора в майбутньому, якщо збережеться нинішня динаміка розвитку технологій

Наслідок 1 – збільшення тактової частоти процесора. Закономірністю, яка діяла до 2000 року, було збільшення частоти процесорів на порядок кожні 10 років. Однак уже з 2005 року цей показник набув максимуму і не піднімається вище 5 GHz. Типові моделі мають частоту до 5 GHz, наприклад, модель 2017 року Intel Core i7 7700K у режимі TurboBoost має тактову частоту 4,5 GHz. Це пов'язано з тим, що тепловиділення на високих частотах є дуже високим і руйнує чіп.

Наслідок 2 – збільшення продуктивності процесора. Найважливішою споживчою характеристикою процесорів є продуктивність. У багатьох роботах показано, що збільшення продуктивності процесорів на порядок за показником MIPS відбувається кожні 7 років. Збільшення продуктивності внаслідок зростання тактової частоти в майбутньому є неможливим. Такі показники зростання продуктивності зберігаються завдяки створенню багатоядерних процесорів.

Наслідок 3 – збільшення кількості ядер. Однією з діючих закономірностей є збільшення кількості ядер на одному кристалі процесора. Тенденцію збільшення кількості ядер можна простежити на прикладі настільних комп'ютерів систем. Перший двохядерний процесор для настільних комп'ютерів (Pentium D) з'явився в 2005 році. У 2017 році компанія Intel представила 18-ядерні процесори серії Core i9 (i9-7980XE). За 12 років кількість ядер збільшилася в 9 разів.

Наслідок 4 – збільшення місткості кеш-пам'яті. Зменшення розміру транзистора призводить до збільшення місткості кеш-пам'яті другого і третього рівнів (L2 + L3). У 2005 році цей показник становив 2 мегабайти. У 2006 році цей параметр збільшився до 4 мегабайтів. Починаючи з 2017 року можна відзначити тенденцію збільшення кеш-пам'яті (L2 + L3) у 2 рази кожні 2 роки. Це одна з тих закономірностей, які будуть діяти так довго, як і закономірність збільшення кількості ядер.

5.8. Архітектура процесорів фірми Intel

Раніше фірма Intel підтримувала три архітектури своїх процесорів для універсальних комп'ютерів: IA-32 (x86), x64 і IA-64. Це змушувало розробників операційних систем, прикладного ПЗ і користувачів ураховувати особливості архітектури МП Intel.

5.8.1. Архітектура процесорів Intel x86

Архітектура x86 – загальна назва МП архітектури, запропонованої фірмою Intel. Архітектура x86 у цей момент домінує на ринку настільних і мобільних комп'ютерів, а також малих серверів. Назва x86 походить від останніх цифр у назвах перших моделей (8086, 80186, 80286, 80386, 80486 і т.д.) цієї лінійки процесорів [5–7].

МП з архітектурою x86 мають CISC-архітектуру зі змінною довжиною команд. Сумісність нових моделей процесорів з попередніми була ключовою в розвитку архітектури x86. Архітектура двічі розширювалася в напрямку більшої довжини машинного слова. Перші x86 процесори були 16-бітовими. У 1985 році фірма Intel випустила 32-бітовий процесор 80386. Його архітектуру називають *x86*, або *IA-32* (Intel Architecture, 32 bits). Сучасні x86 процесори перетворюють набір команд x86 на RISC-подібні мікрокоманди для більш ефективного їх виконання.

5.8.2. Архітектура процесорів Intel x64

Архітектура Amd64 (або x86-64 / intel64 / em64t / x64) – 64-бітова архітектура МП з відповідним набором інструкцій, розроблена компанією AMD. Це розширення архітектури x86 з повною зворотною сумісністю. Набір інструкцій x86-64 у цей час підтримується процесорами AMD Athlon, Phenom, Opteron і Ryzer. Він з незначними доповненнями був ліцензований основним конкурентом AMD – компанією Intel – під назвою Intel 64, раніше відомий як Em64t і IA-32e. Ця архітектура реалізована в більш пізніх моделях процесорів Pentium 4, а також в Core i3, Core i5, Core i7 і Xeon. Корпорація Microsoft

використовує для позначення цього набору інструкцій термін "x64", проте каталог з файлами для архітектури в дистрибутивах Microsoft називається "amd64", тоді як для архітектури x86 він називається "i386".

Архітектура Intel 64 має дві нові особливості:

1. Розширені регістри:

– 8 регістрів загального призначення (R8 – R15);

– всі 16 регістрів загального призначення 64-бітові;

– 8 нових 128-бітових регістрів SSE (XMM8 – XMM15);

– новий командний префікс (REX) для доступу до розширених регістрів.

2. Спеціальний режим Long Mode:

– 64-бітові віртуальні адреси;

– 64-бітові покажчики команд (RIP);

– плоский (flat) адресний простір.

Існує кілька варіантів назв цієї архітектури, які іноді призводять до плутанини.

Архітектура 64-бітових процесорів фірми Intel раніше іменувалася *x64*, проте офіційно назва даної архітектури – *Intel 64*. Поступово фірма Intel відмовляється від найменувань IA-32, IA-32E і EM64T на користь цієї назви, яка тепер є єдиною офіційною для цієї архітектури.

5.8.3. Архітектура процесорів Intel IA-64

Itanium – серверний процесор з архітектурою IA-64, розроблений спільно компаніями Intel і Hewlett-Packard. Уперше він був представлений 29 травня 2001 року. Виробництво оригінального процесора Itanium було зупинено в липні 2002 року одночасно з появою процесора Itanium 2.

Фірми HP і Intel в 1989 році почали спільну роботу з розроблення МП. HP був потрібен процесор наступного покоління для заміни успішних серій робочих станцій і серверів, побудованих на базі процесорів з архітектурою PA-RISC. Компанія хотіла скористатися досягненнями і досвідом Intel в розробленні й виробництві МП.

Itanium був спеціально розроблений для отримання дуже високого рівня паралельних обчислень і досягнення високої продуктивності без збільшення частоти. У цей час випуск процесора припинено.

Запитання до розділу 5

1. Що розуміють під архітектурою процесора?

2. Опишіть ключові властивості процесора, який має архітектуру з акумулятором.

3. Опишіть ключові властивості процесора, який має архітектуру з

регістрами загального призначення.

4. Опишіть ключові властивості процесора, який має архітектуру зі стеком.

5. Укажіть основні неархітектурні характеристики процесорів.

6. Що означає характеристика процесора MIPS?

7. В чому полягає закон Мура? Які його перспективи у майбутньому?

8. Які наслідки випливають із закону Мура?

9. Які дві архітектури процесорів підтримує фірма Intel?

Вправи до розділу 5

Завдання 1. Обчислити значення виразу

$$Y = 20 \cdot B - A / B^2.$$

Завдання 2. Обчислити значення виразу

$$Z = (C / A)^2 - 400.$$

Завдання 3. Обчислити значення виразу

$$Z = (B - A)^2 - 50 \cdot A.$$

Варіанти

1. Вирішити завдання 1 для процесора з архітектурою H51.
2. Вирішити завдання 1 для процесора з архітектурою H86.
3. Вирішити завдання 1 для процесора з архітектурою H87.
4. Вирішити завдання 2 для процесора з архітектурою H51.
5. Вирішити завдання 2 для процесора з архітектурою H86.
6. Вирішити завдання 2 для процесора з архітектурою H87.
7. Вирішити завдання 3 для процесора з архітектурою H51.
8. Вирішити завдання 3 для процесора з архітектурою H86.
9. Вирішити завдання 3 для процесора з архітектурою H87.

ВИСНОВКИ

У цьому посібнику дано визначення архітектури й структури комп'ютера, а також розглянуто принципи організації двох базових архітектур комп'ютерів – прінстонської (фон Неймана) і гарвардської. Цьому присвячено розділ 1.

У розділі 2 розглянуто характеристики підсистеми пам'яті, а в розділі 3 – характеристики процесора. Загальний опис архітектури пам'яті й процесора завершується прикладами реалізації у конкретних пристроях. Моделювання дає змогу краще зрозуміти організацію пам'яті й процесора і спонукає без побоювань *виконувати програмне моделювання будь-яких підсистем комп'ютера*.

У розділі 4 описано базові характеристики процесорів. Усього розглянуто шість таких характеристик, кожну з них підкріплено прикладами реальних процесорів. Вивчення цього розділу має спрямувати зусилля студентів при опануванні нового процесора на аналіз саме цих класифікаційних характеристик. Це дає змогу майбутньому фахівцю *чітко уявляти ті можливості, які слід очікувати від процесора певного типу*, а можливо, і запропонувати нову архітектурну характеристику процесора для аналізу.

У розділі 5 з єдиних позицій розглянуто три гіпотетичних процесори з базовими архітектурами. Для кожного з них розглянуто структуру, мову програмування Асемблер, а також вирішено просте завдання. Оцінювання результатів за певними кількісними показниками дає можливість виконати порівняльний аналіз розглянутих архітектур і виявити їх переваги і недоліки. Але головне посилання, яке повинен усвідомити студент при вивченні цього розділі, полягає в такому: *не існує хороших або поганих архітектур*, у кожному процесорі потрібно шукати риси кожної архітектури, розуміти, як певна архітектура вплинула на його властивості, до яких наслідків (переваг або недоліків) це привело.

Кожний розділ завершується переліком контрольних запитань, які дають можливість закріпити отримані знання, а в кінці посібника наведено завдання, які потрібно вирішити на кожному з трьох розглянутих процесорів, котрі мають одну з базових архітектур. Це дасть змогу самостійно оцінити кожну архітектуру з використанням наведеної методики. Для цього необхідно написати програми й оцінити кілька базових показників для порівняльного аналізу і самостійно зробити висновки про кожну архітектуру.

Не можна сказати, що підхід до вивчення архітектури комп'ютерів, використаний у цьому посібнику, є оригінальним. Він базується на матеріалах книги Уокерлі Дж. [4], але його розширено й поглиблено.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. Мельник, А. О. Архітектура комп'ютера: підручник / А. О. Мельник. – Луцьк: Волинська обласна друкарня, 2008. – 470 с.

2. Корнеев, В. В. Современные микропроцессоры / В. В. Корнеев, А. В. Киселев. – М.: Нолидж, 1998. – 240 с.

3. Морс, С. П. Архитектура микропроцессора 80286: пер. с англ. / С. П. Морс, Д. Д. Алберт. – М.: Радио и связь, 1990. – 304 с.

4. Уокерли, Дж. Архитектура и программирование микроЭВМ: пер. с англ. В 2 кн. / Дж. Уокерли. – М.: Мир, 1984. – Кн.1. – 486 с.; кн. 2. – 359 с.

5. IA-32 Intel® Architecture Software Developer's Manual. Volume 1: Basic Architecture, Order Number 245470-012. URL: http://www.intel.com/.

6. IA-32 Intel® Architecture Software Developer's Manual. Volume 2: Instruction Set Reference, Order Number 245471-012. URL: http://www.intel.com/.

7. IA-32 Intel® Architecture Software Developer's Manual. Volume 3: System Programming Guide, Order Number 245472-012. URL: http://www.intel.com/.

8. Парфенов, Д. А. Исследование закономерностей развития процессоров Intel [Електронний ресурс] / Д. А. Парфенов, А. Я. Аноприенко URL: http://masters.donntu.org/2017/fknt/parfenov/library/article1.htm.

Навчальне видання

Дужий Вячеслав Ігорович
Дужа Вікторія Вікторівна

АРХІТЕКТУРА КОМП'ЮТЕРІВ. ВСТУП

Редактор А. М. Ємленінова

Зв. план, 2020
Підписано до видання 27.11.2020
Ум. друк. арк. 5,2. Обл.-вид. арк. 5,81. Електронний ресурс

Видавець і виготовлювач
Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»
61070, Харків-70, вул. Чкалова, 17
http://www.khai.edu
Видавничий центр ’’ХАІ”
61070, Харків-70, вул. Чкалова,17
izdat @ khai. edu

Свідоцтво про внесення суб'єкта видавничої справи
до Державного реєстру видавців, виготовлювачів і розповсюджувачів
видавничої продукції сер. ДК № 391 від 30.03.2001