

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

АРХІТЕКТУРА КОМП'ЮТЕРІВ
Лабораторний практикум, частина 1

Вінниця ВНТУ 2014

Міністерство освіти і науки України
Вінницький національний технічний університет
Кафедра обчислювальної техніки

АРХІТЕКТУРА КОМП'ЮТЕРІВ
Лабораторний практикум, частина 1

Затверджено Методичною радою Вінницького національного технічного університету як методичні вказівки для студентів напряму підготовки 6.050102 Комп'ютерна інженерія

Протокол № від „ ” 2014 р.

Вінниця ВНТУ 2014

Зміст

Вступ	4
Лабораторна робота 1. Вивчення архітектури персонального комп'ютера за допомогою програми AIDA64 (Everest)	5
Лабораторна робота № 2 Класифікація за комп'ютерів за конструкцією і призначенням	15
Лабораторна робота 3. Системи числення	22
Лабораторна робота 4. Основи машинної арифметики з двійковими числами	29
Лабораторна робота 5. Форми представлення чисел в ЕОМ	35
Лабораторна робота 6. Архітектура ЕОМ фон Неймана. Учбові ЕОМ	44
Лабораторна робота 7. Структура процесора 80x86	57
Лабораторна робота 8. Функціональна схема та сигнали мікропроцесора 8086	64
Лабораторна робота 9 Архітектура сучасних арифметико-логічних пристроїв: елементи RISC - архітектури та конвеєрної обробки даних	72
Лабораторна робота 10. Регістри мікропроцесорів 8086 - 80186	79
Лабораторна робота 11. Асемблювання програм	91
Лабораторна робота 12. Сегментація пам'яті	96
Перелік рекомендованої літератури	104

Вступ

Лабораторний практикум передбачає: ознайомлення з апаратною архітектурою сучасного персонального комп'ютера, системами числення, основами машинної арифметики, формами представлення даних, базовою структурою універсального процесора, організацією пам'яті ЕОМ.

Дисципліна "Архітектура комп'ютерів" є однією з базових дисциплін напряму 6.050102 - "Комп'ютерна інженерія". Метою викладання дисципліни є формування знань, умінь та навичок студентів щодо застосування теоретичних основ та методів розв'язання прикладних задач аналізу та розробки архітектури обчислювальних машин, їх складових елементів, а також обробки даних в сучасних комп'ютерах і системах.

Такі знання є необхідними фахівцям з комп'ютерної інженерії при взаємодії з комп'ютерними системами, а також мережами – локальними та глобальною.

Основними завданнями лабораторного курсу з першої частини предмету «Архітектура комп'ютерів» є:

- ознайомлення з основними архітектурами обчислювальних машин та їх складових елементів;
- вивчення способів організації пам'яті, введення-виведення, обробки даних;
- опанування теорії та практики проектування ЕОМ та мікропроцесорних систем.

При вивченні матеріалів з курсу студенти повинні:

знати:

- матеріал програми курсу "Архітектура комп'ютерів", особливості побудови архітектури обчислювальних машин, їх складових елементів, а також особливості обробки даних в сучасних комп'ютерах і системах;

вміти:

- використовувати принцип програмного управління для організації обчислювальних процесів в комп'ютері;
- оцінювати характеристики комп'ютера на архітектурному та структурному рівнях;
- користуватися мовами опису апаратних і програмних засобів комп'ютерів;
- проводити розрахунки для порівняння ефективності варіантів побудови пристроїв комп'ютера;
- застосовувати сучасні засоби підвищення продуктивності, надійності та функціональних можливостей обчислювальних засобів;
- оцінювати ефективність роботи комп'ютера;
- аналізувати архітектуру мікропроцесорів на базі арифметико - логічних пристроїв з розподіленою та зосередженою логікою і пристроїв управління з жорсткою та гнучкою логікою;
- аналізувати системи команд, формати і структуру даних, способи адресації команд та операндів, мікроалгоритми і мікропрограми реалізації різних операцій;
- розподіляти адресний простір комп'ютера, розробляти архітектуру пам'яті.

Лабораторна робота №1

ВИВЧЕННЯ АРХІТЕКТУРИ ПЕРСОНАЛЬНОГО КОМП'ЮТЕРА ЗА ДОПОМОГОЮ ПРОГРАМИ AIDA64 (EVEREST)

1 Мета роботи: За допомогою програмної утиліти AIDA64 (Everest) вивчити архітектуру персонального комп'ютера (ПК), ознайомитися з основними пристроями ПК, ознайомитися з основними характеристиками пристроїв ПК.

2 Короткі теоретичні відомості:

AIDA64 (вона ж Everest) – програма для перегляду інформації про апаратну і програмну конфігурації комп'ютера. Програма аналізує конфігурацію комп'ютера і видає детальну інформацію про встановлені в системі пристрої – процесора, системні плати, відеокарти, аудіокарти, модулі пам'яті і так далі, а також інформацію про їхні характеристики, набори команд та їх режими роботи, їх виробників, встановлене програмне забезпечення, конфігурації операційної системи і встановлених драйверів.

Для виконання контрольної роботи досить демонстраційної версії програми AIDA64.

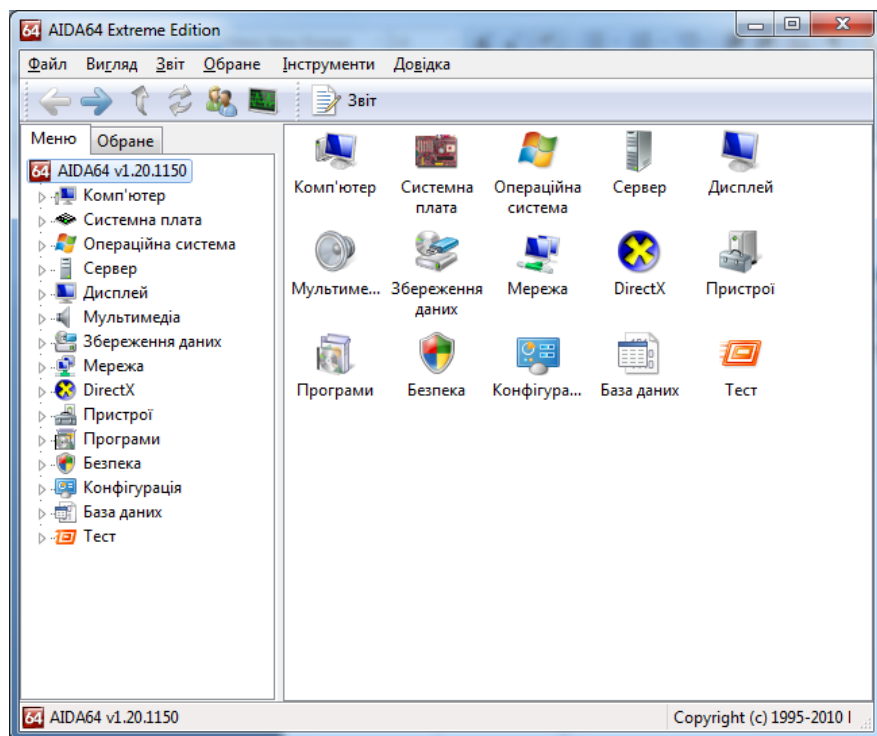


Рисунок 1 – Графічний інтерфейс утиліти AIDA64

У програмі є досить широкий набір тестів:

- читання з пам'яті;
- запис в пам'ять;
- копіювання в пам'яті;
- затримка пам'яті;
- CPU Queen – тестує продуктивність процесора в цілочисельних операціях при вирішенні класичної «задачі з ферзями»;
- CPU PhotoWorxx – тестує продуктивність блоків цілочисельних арифметичних операцій, множення, а також підсистеми пам'яті при виконанні ряду стандартних операцій з RGB-зображеннями;
- CPU ZLib – тестує продуктивність процесора і підсистеми пам'яті при створенні архівів формату ZIP за допомогою популярної відкритої бібліотеки ZLib. Використовує цілочисельні операції;
- CPU AES – тестує швидкість процесора при виконанні шифрування за криптоалгоритмом AES. Здатний використовувати низькорівневі команди шифрування процесорів VIA C3 і C7, що дозволяє останньому бути одним з лідерів тесту, перевершуючи по продуктивності ряд багатоядерних процесорів Intel і AMD;
- FPU Julia – тестує продуктивність блоків процесора, що виконують операції з плаваючою комою, в обчисленнях з 32-розрядною точністю. Моделює кілька фрагментів фрактала Жюліа. При можливості використовує інструкції MMX, SSE і 3DNow!;
- FPU Mandel – тестує продуктивність блоків процесора, що виконують операції з плаваючою комою, в обчисленнях з 64-розрядною точністю шляхом моделювання декількох фрагментів фрактала Мандельброта. Здатний використовувати інструкції SSE2.
- FPU SinJulia – ускладнений варіант тесту FPU Julia. Тестує продуктивність блоків процесора, що виконують операції з плаваючою комою, в обчисленнях з 80-розрядною точністю. Використовує інструкції x87, призначені для обчислення тригонометричних і показових функцій.

3 Завдання до роботи

Для виконання роботи на досліджуваному комп'ютері повинна бути встановлена програма AIDA64 або Everest актуальної версії. Зрозуміти значення багатьох термінів і скорочень допоможе Глосарій:

ACPI (Advanced Configuration and Power Interface) – вдосконалений інтерфейс конфігурації та керування живленням) – відкритий промисловий стандарт, вперше випущений в грудні 1996 року і розроблений спільно компаніями HP, Intel, Microsoft, Phoenix і Toshiba, який визначає загальний інтерфейс для виявлення апаратного забезпечення, керування живленням і конфігурації материнської плати і пристроїв. Завдання ACPI – забезпечити взаємодію між операційною системою, апаратним забезпеченням і BIOS материнської плати.

DMI (Direct Media Interface) – послідовна шина розроблена Intel для під'єднання південного мосту материнської плати (ICH) до північного мосту. Магістраль DMI у деяких сучасних процесорах використовується для під'єднання чіпсета до процесора (для Core i3, Core i5 і деяких серій Core i7).

COM-port – асинхронні послідовні порти (що позначаються COM1 - COM3). Через них раніше приєднувалися миша, модем тощо.

DMI (Desktop Management Interface) – інтерфейс програмування додатків (Application Programming Interface – API), що дозволяє програмному забезпеченню збирати дані про характеристики комп'ютера. Специфікація DMI розроблена консорціумом Distributed Management Task Force (DMTF), очолюваному фірмою Intel. Даний інтерфейс дозволяє користувачеві отримати інформацію про апаратне забезпечення ПК.

FSB (Front Side Bus) – системна шина (магістраль) в архітектурі корпорації Intel, що зв'язує процесор з чіпсетом. Часто використовується як загальна назва для магістралі, що сполучає процесор і чіпсет.

Game-port – порт для ігрових пристроїв (для підключення джойстика).

ICH – контролер–концентратор введення–виведення – південний міст (southbridge) – забезпечує взаємодію між ЦП і жорстким диском, картами PCI, інтерфейсами IDE, SATA, USB і пр. Також іноді до чіпсетів відносять мікросхему Super I/O, яка підключається до південного мосту і відповідає за низькошвидкісні порти RS232, LPT, PS/2.

IEEE – 1394 (FireWire) – інтерфейс для передачі великих обсягів відео інформації в реальному часі (для підключення цифрових відеокамер, зовнішніх жорстких дисків, сканерів та іншого високошвидкісного обладнання). Інтерфейсом FireWire оснащені всі відеокамери, що працюють в цифровому форматі. Може використовуватися і для створення локальних мереж.

LPT – паралельні порти, до них звичайно підключаються принтери.

MCH (Memory Controller Hub) – контролер –концентратор пам'яті – північний міст (northbridge) – забезпечує взаємодію центрального процесора (ЦП) з пам'яттю і відеоадаптером (PCI Express). У нових чіпсетах часто є інтегрована відеопідсистема. Контролер пам'яті може бути інтегрований в процесор (Athlon64, Athlon II, Phenom II, Core i7, Core i5, Core i3).

PCI (Peripheral Component Interconnect) – паралельна шина для підключення різних периферійних пристроїв. В даний час інтенсивно витісняється з комп'ютерної техніки шиною PCI –Express.

PCI –Express – це послідовний інтерфейс, розроблений організацією PCI– SIG на чолі Intel і призначений для використання як локальної шини замість PCI.

PS/2 – асинхронні послідовні порти для підключення клавіатури і маніпулятора миша.

QPB (Quad – Pumped Bus) – 64–бітна процесорна шина забезпечує зв'язок процесорів Intel з північним мостом чіпсета. Характерною її особливістю є передача чотирьох блоків даних (і двох адрес) за такт. Таким чином, для частоти FSB, рівної 200 МГц, ефективна частота передачі даних буде еквівалентна 800 МГц (4 x 200 МГц).

QPI (QuickPath Interconnect) – послідовна шина типу точка–точка для з'єднання процесорів між собою і з чіпсетом, розроблена фірмою Intel. QPI створювався у відповідь на розроблену раніше фірмою AMD шину HyperTransport. Використовується в топових процесорах Intel Core i7 і деяких Core i5.

SPD (Serial Presence Detect) – специфікація, що описує технологію запису, зберігання й зчитування інформації про характеристики 168–контактних модулів DIMM.

USB (Universal Serial Bus) – універсальний інтерфейс для підключення 127 пристроїв (цей інтерфейс може розташовуватися на передній або бічній стінці корпусу).

VGA (Video Graphics Array) – вихід контролера графічного адаптера (відеокарти) для підключення монітора.

iRDA – інфрачервоні порти призначені для бездротового підключення кишенькових або блокнотних ПК або стільникового телефону до настільного комп'ютера. Зв'язок забезпечується за умови прямої видимості, дальність передачі даних не більше 1 м. Якщо в ПК немає вбудованого iRDA адаптера, то він може бути виконаний у вигляді додаткового зовнішнього пристрою (USB iRDA адаптера), що підключається через USB –порт.

HT (HyperTransport) – послідовна двунаправленна магістраль служить для зв'язку процесорів AMD сімейства починаючи з K8 один з одним, а також з чіпсетом. У багатьох чіпсетах використовують HT для зв'язку між мостами.

Таймінги оперативної пам'яті. Схема таймінгів включає в себе затримки **CL–RCD–RP–RAS** відповідно. Для роботи з пам'яттю необхідно для початку вибрати чіп, з яким ми будемо працювати. Робиться це командою CS (Chip Select). Потім вибирається банк і рядок. Перед початком роботи з будь-яким рядком необхідно її активувати. Робиться це командою вибору рядка RAS (Row Address Strobe), при виборі рядка вона активується. Потім потрібно вибрати стовпець командою CAS (Column Address Strobe) – ця ж команда ініціює читання.

- **CL** (Cas Latency) – мінімальний час між подачею команди на читання (CAS) і початком передачі даних (затримка читання).
- **RCD** (RAS to CAS delay) – час, необхідний для активізації рядка банку, або мінімальний час між подачею сигналу на вибір рядка (RAS) і сигналу на вибір стовпця (CAS).
- **RP** (Row Precharge) – час, необхідний для попереднього заряду банку (precharge). Іншими словами, мінімальний час закриття рядка, після чого можна активувати новий рядок банку.
- **RAS** (Active to Precharge) – мінімальний час активності рядка, тобто мінімальний час між активацією рядка (її відкриттям) і подачею команди на предзаряд (початок закриття рядка). Рядок не може бути закритий раніше цього часу.
- **CR** (Command Rate) – час, необхідний для декодування контролером команд і адрес. Інакше кажучи, мінімальний час між подачею двох команд. При значенні 1T команда розпізнається 1 такт, при 2T – 2 такти, 3T – 3 такти.

Це всі основні таймінги. Решта таймінгів мають менший вплив на продуктивність.

Чіпсет (chip set) – набір мікросхем, спроектованих для спільної роботи з метою виконання набору певних функцій. Так, в комп'ютерах чіпсет виконує роль сполучного компонента, що забезпечує спільне функціонування підсистем пам'яті, ЦП, введення–виведення та інших. Чіпсети зустрічаються і в інших пристроях, наприклад, в радіоблоках стільникових телефонів. Чіпсет складається з двох основних мікросхем (іноді вони об'єднуються в один чіп) «північний міст» і «південний міст».

3.1 Ознайомитися з сумарною інформацією про комп'ютер

Для цього в лівому меню у списку «Комп'ютер» слід вибрати пункт «Сумарна інформація», після чого в правому вікні з'явиться список основних параметрів досліджуваного комп'ютера. Виписати:

- тип комп'ютера;
- тип операційної системи;
- ім'я комп'ютера;
- ім'я користувача;
- тип центрального процесора (ЦП);
- тип системної плати;
- тип чіпсета системної плати;
- кількість і тип оперативної (системної) пам'яті;
- тип відеоадаптера;
- тип монітора;
- тип і обсяг дискового накопичувача (жорсткого диска – ЖД);
- перерахувати інші пристрої введення–виведення.

Зробити висновки.

3.2 Ознайомитися з центральним процесором комп'ютера

Для цього в лівому меню у списку «Системна плата» вибрати пункт «ЦП», після чого в правому вікні з'явиться список основних параметрів ЦП досліджуваного комп'ютера. Виписати основні властивості ЦП:

- тип ЦП;

- назва процесора (псевдонім) ЦП;
- кількість ядер;
- степпінг ЦП;
- набори інструкцій;
- вихідна частота;
- розмір і характеристики кеш–пам’яті ЦП;
- фізичні параметри ЦП.

Отримати відомості про реальну частоту процесора, для цього в списку «Комп’ютер» вибрати пункт «Розгін». У даному пункті в реальному масштабі часу відображається поточна частота процесора. Виписати поточну частоту процесора. Порівняти вихідну частоту процесора з поточною частотою. Зробити висновки.

3.3 Ознайомитися з материнською платою

Для цього в лівому меню у списку «Системна плата» вибрати пункт «Системна плата», після чого в правому вікні з’явиться список основних параметрів материнської плати досліджуваного комп’ютера. Виписати:

- назву материнської плати і фірму виробника;
- властивості системної шини (FSB, HT, QPB);
- властивості шини пам’яті;
- назва чіпсета;
- фізичну інформацію про системну плату.

3.4 Ознайомитися з властивостями модулів ОЗП

Для цього в лівому меню у списку «Системна плата» вибрати пункт «SPD». Виписати властивості модулів ОЗП і основні таймінги пам’яті, для різних частот. Якщо встановлені різні модулі пам’яті, виписати параметри для кожного з них. Зробити висновки.

3.5 Ознайомитися з чіпсетом материнської плати

Для цього в лівому меню у списку «Системна плата» вибрати пункт «чіпсет».

3.5.1 Ознайомитися з властивостями «північного моста» чіпсета.

Для цього у верхньому вікні пункт «Північний міст». Перерахувати контролери, вбудовані в «північний міст». вписати:

- назва «північного моста»;
- підтримувані швидкості системної шини (FSB, HT, QPB);
- підтримувані типи оперативної пам'яті;
- тип контролера пам'яті;
- максимальний обсяг оперативної пам'яті;
- основні таймінги пам'яті (CR, tRAS, tRP, tRCD, CL, tREF).

Порівняти характеристики ОЗП з отриманими в попередньому пункті. Зробити висновки.

3.5.2 Ознайомитися з властивостями «південного моста» чіпсета.

Для цього у верхньому вікні пункт «Південний міст». Перерахувати пристрої, що містяться в «південному мосту». Зробити висновки.

3.6 Ознайомитися з системою зберігання даних – постійними запам'ятовуючими пристроями (ПЗП)

Для цього в лівому меню у списку «Збереження даних» вибрати пункт «Збереження даних Windows», після чого в правому верхньому вікні з'явиться список всіх можливих ПЗП досліджуваного комп'ютера. У роботі слід розглянути параметри жорсткого диска і оптичного DVD накопичувача. Вписати їх основні характеристики, такі як:

- назва ЖД;
- виробник;
- ємність;
- швидкодію;
- інтерфейс підключення;
- фізичні параметри:
 - ✓ форм-фактор (розмір в дюймах);

- ✓ кількість пластин (дисків);
- ✓ вага;
- ✓ швидкість обертання.

Зробити висновки.

3.7 Ознайомитися з наявними на платі портами введення–виведення

Для цього в розділі «Комп'ютер» вибрати пункт «DMI». У даному пункті з розділу «Системні роз'єми» виписати наявні на материнській платі роз'єми. З розділу «Роз'єми портів» виписати роз'єми для підключення зовнішніх пристроїв введення–виведення, для кожного вказати тип порту.

3.8 Провести тестування швидкодії ОЗП

Для цього перейти в розділ «Тест» і вибрати відповідні пункти. Для початку тестування слід натиснути кнопку «Оновити», або клавішу «F5» на клавіатурі. Провести наступні тести ОЗП:

- читання з пам'яті – тестує швидкість пересилання даних з ОЗП до процесора;
- запис в пам'ять;
- копіювання в пам'яті – тестує швидкість пересилання даних з одних комірок пам'яті в інші через кеш процесора;
- затримка пам'яті – тестує середній час зчитування процесором даних з ОЗП.

Записати результати тестування. Порівняти продуктивність досліджуваної системи з продуктивністю еталонних систем. Виписати найбільш близькі по продуктивності системи. Зробити висновки.

3.9 За результатами попередніх пунктів побудувати структурну схему ПК

У схемі повинні бути відображені всі пристрої, що входять в ПК, з їх назвами і основними параметрами.

4 Зміст звіту:

- 1 Мета роботи.
- 2 Опис пристроїв ПК та їх основних параметрів (згідно з пунктами роботи).
- 3 Результати тестів і порівняння з іншими конфігураціями.
- 4 Структурна схема ПК.
- 5 Висновки по кожному пункту роботи.

5 Контрольні питання:

- 1 Основні принципи побудови ЕОМ, структура Дж. фон Неймана.
- 2 Намалювати структурну схему ПК, пояснити призначення всіх компонентів.
- 3 Центральний процесор, основні характеристики.
- 4 Система пам'яті. Склад, призначення.
- 5 Системна магістраль. Визначення, призначення, параметри.
- 6 Основні внутрішні шини ПК.
- 7 «Північний міст». Склад, призначення.
- 8 «Південний міст». Склад, призначення.
- 9 Пристрої введення–виведення ПК.
- 10 Чинники, що впливають на продуктивність ПК.
- 11 Які пристрої до яких портів можуть підключатися ?

Лабораторна робота № 2

Класифікація за комп'ютерів за конструкцією і призначенням

Мета роботи: Ознайомитись з призначенням, особливістю застосування супер комп'ютерів, Майн Фреймів – Main Frame, міні комп'ютерів, мікро комп'ютерів, в тому числі персональні комп'ютери.

Короткі теоретичні відомості

Супер комп'ютери – представляють собою багатопроцесорні та багатомашинні комплекси, що базуються на спільній пам'яті та спільних зовнішніх пристроях. Архітектура супер комп'ютерів заснована на засадах паралелізації та конвеєризації обчислень.

Супер комп'ютери мають величезну обчислювальну потужність. Їх використовують для роботи з додатками, що вимагають найбільш інтенсивних обчислень (наприклад, прогнозування погодно-кліматичних умов, моделювання ядерних випробувань тощо). Іноді супер комп'ютери працюють з одним завданням, що використовує всю пам'ять та всі процесори системи; в інших випадках вони забезпечують виконання великого числа різноманітних застосувань.

Топ50 самих потужних комп'ютерів СНД (<http://supercomputers.ru/?page=rating>)

Майн Фрейми (Main Frame) – призначені для вирішення широкого кола науковотехнічних завдань. Вони є дорогими за вартістю та обслуговуванням.

Для Майн фреймів характерними є багатопроцесорна архітектура, розгалужена периферія, багатокористувацький режим роботи. Домінуюче положення у випуску комп'ютерів такого класу займає фірма ІВМ (США).

Майн фрейми застосовують у великих обчислювальних центрах, де підтримується цілодобовий режим роботи, а штат налічує 200-300 працівників. Вартість порядку 100 000 дол.

Міні комп'ютери -це потужні комп'ютери, подібні до Майн фреймів, і розраховані на десятки робочих місць. Представлені як кілька обчислювальних комплексів, що конструктивно розміщені в одному корпусі.

Використовують у великих підприємствах, наукових закладах і установах. Часто використовують для керування виробничими процесами. Вартість порядку 10 000 дол.

Мікрокомп'ютери мають кілька процесорів, надвеликі об'єми оперативної пам'яті і є доступними для багатьох установ. Для обслуговування достатньо обчислювальної лабораторії у складі кількох чоловік.

Персональні комп'ютери – це мікрокомп'ютери універсального призначення, що розраховані на одне робоче місце і не потребують обслуговуючого персоналу.

Широкого поширення персональні комп'ютери набули в останні 20 років. 3

появою Інтернету популярність зростає значно вище, оскільки за допомогою персонального комп'ютера можна користуватись науковою, довідковою, учбовою та розважальною інформацією, отримати дешеві засоби комунікації (е-мейл, IP-телефонія).

Класифікація персональних комп'ютерів

Персональні комп'ютери існують двох типів:

Стаціонарний настільний комп'ютер.

Портативні (мобільні) комп'ютери.

Настільні персональні комп'ютери привабливі тим, що є подібними до конструктора. Всі пристрої є окремими модулями, які легко збираються і замінюються. Але такі комп'ютери мають стаціонарно стояти в визначеному місці.



В портативних комп'ютерах всі основні пристрої містяться в одному корпусі, зазвичай, пристрої мають невелику вагу і є досить зручними для сучасних умов. Класифікація портативних комп'ютерів : ноутбуки, нетбуки, субноутбуки, планшетні персональні комп'ютери, інтернет планшети iPad, кишенькові персональні комп'ютери, смартфони, мультимедійні смартфони iPhone, пристрої для читання електронних книг e-Book.

Ноутбук (NoteBook) Ноутбук -це портативний персональний



комп'ютер, в корпусі якого містяться базові компоненти комп'ютера, дисплей, клавіатура, сенсорна панель – тачпад (TouchPad), а також акумуляторні батареї. Ноутбук може живитися як від власних акумуляторів так і від адаптера мережі.

Ноутбуки відрізняються невеликими розмірами і вагою, час автономної роботи ноутбуків коливається в межах від 1 до 6-8 годин.

Він виконує всі функції звичайного

стаціонарного комп'ютера, але має важливу перевагу: ноутбук -це переносний комп'ютер, який можна завжди носити з собою і використовувати в будь-якому місці.

Нетбук (NetBook)

Нетбук — це невеликий ноутбук, що призначений для виходу в Інтернет і роботи з офісними програмами. Відрізняється компактними розмірами, невеликою вагою, низьким енергоспоживанням і порівняно невисокою вартістю.

За допомогою нетбука можна переглядати Інтернет сторінки та електронну пошту, вести блоги, читати електронні книги.



Нетбук – не є потужним комп'ютером. На ньому неможливо працювати зі складними програмами, обробляти фотографії, а тим більше переглядати відеофільми. Об'єму оперативної пам'яті і потужності процесора для цих завдань не вистачить. Для щоденної і постійної роботи нетбук є заслабким. Він буде у нагоді як додатковий комп'ютер, який можна брати з собою в дорогу.

Субноутбук (Subnotebook)



Субноутбук – це ультра портативний комп'ютер, гібрид ноутбука і нетбука, що має маленький розмір, вагу і більшість характерних рис звичайного ноутбука.

Він достатньо продуктивний і могутній, хоча діагональ субноутбуків рідко перевищує 13,3 дюйми. Вага таких апаратів коливається в межах від 1,5 до 2 кг, дизайн переважно стильний, несхожий на звичайні моделі ноутбуків і нетбуків. Не дивлячись на «кишенькову потужність», на такі апарати встановлюється інтегрована відеокарта. Пограти в сучасні 3D – ігри навряд чи можна, але можна працювати з офісними програмами або подивитися відеофільм в HD – форматі.



Планшетний персональний комп'ютер (планшетник, Tablet PC)

Планшетники -це клас ноутбуків, обладнаних екраном, що об'єднаний з планшетним пристроєм рукописного введення. Екран дозволяє працювати за допомогою стилуса або пальців, без використання клавіатури і миші, має тонкий корпус і привабливий вигляд.

Користувач може вводити текст, використовуючи екранну (віртуальну) клавіатуру, звичайну клавіатуру (якщо вона є у складі пристрою) і за допомоги вбудованих програми

розпізнавання рукописного тексту та мови.

Типи планшетних персональних комп'ютерів:

Планшетники — пристрої без повноцінної клавіатури.

Планшетні ноутбуки часто називають «конвертованими» або трансформерами, завдяки можливості до трансформації: пристрій може виглядати як ноутбук, екран можна розвернути навколо осі на 180° і покласти на клавіатуру — ноутбук виглядатиме як планшет. Планшетні нетбуки -це нетбуки з поворотним екраном.



UMPC — компактний варіант планшетного комп'ютера, призначений замінити КПК. Має певні конструктивні відмінності, а також деякі відмінності в інтерфейсі, оскільки пристрій призначено спеціально для управління пальцями. Може мати вбудовану клавіатуру, як правило, нестандартну.

Інтернет планшет

Інтернет планшет (iPad) — тип



комп'ютерів, що відносяться до планшетних комп'ютерів. Він суміщає в собі найкращі якості ноутбука і смартфона.

Такі портативні комп'ютери зазвичай використовуються для читання електронних книг, перегляду фотоальбомів та відеофільмів, прослуховування музики і, звичайно ж,

для роботи в Інтернеті.

Особливості Інтернет планшета:

Низька вартість пристрою (в межах 400 — 900\$). Сенсорний екран призначений для роботи за допомогою пальців. Легкий і зручний користувацький інтерфейс (більше схожий на інтерфейс смартфона, ніж на інтерфейс комп'ютера).

Розвинені засоби безпроводного Інтернет з'єднання для швидкого перегляду web-сторінок. Тривалий час автономної роботи (яким раніше могли похвалитися лише мобільні телефони).

Кишеньковий персональний комп'ютер (КПК, наладонник, PalmTop)

КПК — збірна назва класу портативних електронних обчислювальних пристроїв, які спочатку декларувалися як електронні органайзери. Для позначення всього класу пристроїв в англійській мові використовується словосполучення PDA (Personal Digital Assistant), що перекладається як «особистий цифровий секретар».

КПК обладнано процесором, платами розширення, звуковою системою і flash-пам'яттю, яка, на відміну від вінчестера, займає менше фізичного місця.

Дисплей КПК реагує на дотик спеціальної палички стилуса.

Основні функції КПК:

Офісні програми. Для вводу тексту доступна екраннаклавіатура, рукописний ввід і повноцінна висувна клавіатура.

Вихід в Інтернет. Під'єднатися можна через мобільний телефон (Bluetooth / IRDA) або безпроводну мережу Wi-Fi.

Організація розкладу поточних справ та щоденник. Комп'ютер може автоматично нагадувати про пункти розкладів.

Звуковий програвач, диктофон, фотоапарат, відеокамера.

Перегляд зображень, відеороликів, фільмів, наявність графічного редактора.

Дистанційне керування. Вся побутова техніка, що має інфрачервоний порт, піддається управлінню за допомогою спеціалізованих програм.

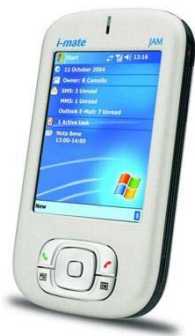
Читання карти місцевості. Особливо ефективними КПК будуть за наявності модуля GPS (глобальна система позиціонування) і спеціальних програм для планування маршрутів.



Смартфон (Smartphone)

Смартфон (розумний телефон) — це мобільний телефон з розширеною функціональністю, в деяких моделях функціональність є наближеною до КПК. У зв'язку з тим, що деякі смартфони дуже вдало суміщають в собі функціональність мобільного телефону і КПК, для позначення подібних пристроїв часто використовується

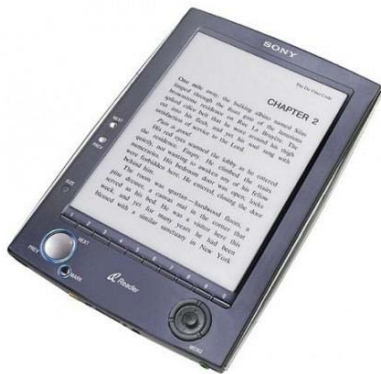
термін «комунікатор».



Мультимедійний смартфон (iPhone)



iPhone — мультимедійні смартфони, що розроблені корпорацією Apple. Смартфони суміщають в собі функціональність плеєра iPod, комунікатора та Інтернет планшету.



Пристрій для читання електронних книг (Ebook reader)

E-Book device — загальна назва для цілої групи вузькоспеціалізованих компактних пристроїв, що призначені для відображення текстової і графічної інформації (у форматах html, txt, pdf тощо).

Основною відмінністю E-book від КПК,

планшетників, ноутбуків або нетбуків є обмежена функціональність, що дозвол яє істотно збільшити робочий час використання.

Контрольні запитання

- 1 Класифікація комп'ютерів за призначенням?
- 2 З якою метою створюють супер комп'ютери?
- 3 До якого типу відносяться персональні комп'ютери учбової лабораторії ?
- 4 Переваги та недоліки стаціонарних комп'ютерів?
- 5 Основні функції пристрою E-book?
- 6 Які комп'ютери називають «комунікаторами»?
- 7 Типи планшетних комп'ютерів?
- 8 Основне призначення нетбуків?

Лабораторна робота № 3

Системи числення

Мета: Отримання практичних навичок переводу чисел із одної системи числення в іншу.

Короткі теоретичні відомості

Під системою числення розуміється спосіб представлення будь-якого числа за допомогою деякого алфавіту символів, тобто цифр. Всі системи числення діляться на позиційні і непозиційні.

Непозиційними системами є такі системи числення, в яких кожен символ зберігає своє значення незалежно від місця його положення в числі.

Прикладом непозиційної системи числення є римська система. До недоліків таких систем належать наявність великої кількості знаків і складність виконання арифметичних операцій.

Система числення називається позиційною, якщо одна і та ж цифра має різне значення, що визначається позицією цифри в послідовності цифр та зображує число. Це значення змінюється в однозначній залежності від позиції, займаної цифрою, по деякому закону.

Прикладом позиційної системи числення є десяткова система, що використовується у повсякденному житті.

Кількість p різних цифр, уживаних у позиційній системі визначає назву системи числення і називається основою системи числення - " p ".

У десятковій системі використовуються десять цифр : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9; ця система має основу число десять.

Будь-яке число N в позиційній системі числення з основою p може бути представлено у вигляді полінома від основи p :

$$N = a_n p^n + a_{n-1} p^{n-1} + \dots + a_1 p + a_0 + a_{-1} p^{-1} + a_{-2} p^{-2} + \dots$$

тут N - число , a_j - коефіцієнти (цифри числа) , p - основа системи числення ($p > 1$).

Прийнято представляти числа у вигляді послідовності чисел:

$$N = a_n a_{n-1} \dots a_1 a_0 . a_{-1} a_{-2} \dots$$

У цій послідовності точка відділяє цілу частину числа від дробової (коефіцієнти при позитивних ступенях, включаючи нуль , від коефіцієнтів при негативних ступенях). Точка опускається, якщо немає негативних ступенів (число ціле).

У ЕОМ застосовують позиційні системи числення з недесятковою основою: двійкову , вісімкову , шістнадцяткову .

В апаратній основі ЕОМ лежать двохпозиційні елементи , які можуть перебувати лише у двох станах; одне з них позначається 0, а інше - 1. Тому основною системою числення застосовується в ЕОМ є двійкова система .

Двійкова система числення . Використовується дві цифри: 0 і 1. У двійковій системі будь-яке число може бути представлено у вигляді:

$$N = b_n b_{n-1} \dots b_1 b_0 . b_{-1} b_{-2} \dots \text{де } b_j \text{ або } 0, \text{ або } 1 .$$

Вісімкова система числення . Використовується вісім цифр : 0, 1, 2, 3, 4, 5, 6, 7. Вживається в ЕОМ як допоміжна для запису інформації в скороченому вигляді. Для представлення однієї цифри вісімковій системі використовується три двійкових розряди (тріада) (Таблиця 1).

Шістнадцяткова система числення . Для зображення чисел вживаються 16 цифр. Перші десять цифр цієї системи позначаються цифрами від 0 до 9 , а старші шість цифр - латинськими буквами: 10- А , 11- В , 12- С , 13- D , 14 -Е , 15 -F.

Шістнадцяткова система використовується для запису інформації в скороченому вигляді. Для представлення однієї цифри шістнадцятковій системі числення використовується чотири двійкових розряди (тетрада) (таблиця 1).

Таблиця 1 - Найбільш важливі системи числення

Двійкова (Основа 2)	Вісімкова (Основа 8)		Десяткова (Основа 10)	Шістнадцяткова (Основа 16)	
		тріади			тетради
0	0	000	0	0	0000
1	1	001	1	1	0001
	2	010	2	2	0010
	3	011	3	3	0011
	4	100	4	4	0100
	5	101	5	5	0101
	6	110	6	6	0110
	7	111	7	7	0111
			8	8	1000
			9	9	1001
				A	1010
				B	1011
				C	1100
				D	1101
				E	1110
				F	1111

Переклад чисел з однієї системи числення в іншу.

Переклад чисел у десяткову систему здійснюється шляхом складання статичного ряду з основою тієї системи, з якої число перекладається. Потім підраховується значення суми. **Пример.**

а) Перевести 10101101.101_2 у 10 "

$$10101101.101_2 = 1_2 \cdot 2^7 + 0_2 \cdot 2^6 + 1_2 \cdot 2^5 + 0_2 \cdot 2^4 + 1_2 \cdot 2^3 + 1_2 \cdot 2^2 + 0_2 \cdot 2^1 + 1_2 \cdot 2^0 + 1_2 \cdot 2^{-1} + 0_2 \cdot 2^{-2} + 1_2 \cdot 2^{-3} = 173.625_{10}$$

б) Перевести 703.04_8 у 10 "

$$703.04_8 = 7_8 \cdot 8^2 + 0_8 \cdot 8^1 + 3_8 \cdot 8^0 + 0_8 \cdot 8^{-1} + 4_8 \cdot 8^{-2} = 451.0625_{10}$$

в) Перевести $B2E.4_{16}$ у 10 "

$$B2E.4_{16} = 11_{16} \cdot 16^2 + 2_{16} \cdot 16^1 + 14_{16} \cdot 16^0 + 4_{16} \cdot 16^{-1} = 2862.25_{10}$$

Переклад цілих десяткових чисел в недесяткову систему числення здійснюється послідовним розподілом десяткового числа на основу тієї системи, в яку воно перекладається, до тих пір, поки не вийде приватне менше цього підстави. Число у новій системі записується у вигляді залишків поділу, починаючи з останнього.

Приклад:

а) перевести 181_{10} у 8 "

$$\begin{array}{r|l} 181 & 8 \\ -176 & 22 \\ \hline 5 & 16 \\ & 2 \end{array}$$

6

Результат: $181_{10} = 265_8$

б) перевести 622_{10} у 16 "

$$\begin{array}{r|l} 622 & 16 \\ -48 & 38 \\ \hline 142 & 32 \\ -128 & 14 \end{array}$$

Результат: $622_{10} = 26E_{16}$

Переклад правильних дробів з десяткової системи числення в недесяткову.

Для перекладу правильної десяткового дробу в іншу систему цю дріб треба послідовно множити на основу тієї системи, в яку вона перекладається. При цьому множаться тільки дробові частини. Дріб у новій системі записується у вигляді цілих частин творів, починаючи з першого.

Приклад:

перевести 0.3125_{10} у 8 " с.с.

$$\begin{array}{r} \downarrow \\ \underline{0.3125 \times 8} \\ 2.5000 \times 8 \\ \downarrow \\ 4.0000 \end{array}$$

Результат: $0.3125_{10} = 0.24_8$

Для перекладу неправильної десяткового дробу в систему числення з недесятковою основою необхідно окремо перевести цілу частину і окремо дробову.

Приклад:

1) Переведемо цілу частину:

2) Переведемо дробову частину:

Перевести $23.125_{10} \rightarrow 2$

$$\begin{array}{r} 23 \mid 2 \\ \underline{22} \mid 11 \mid 2 \\ 1 \mid 10 \mid 5 \mid 2 \\ \quad \underline{1} \mid 4 \mid 2 \mid 2 \\ \quad \quad \underline{1} \mid 2 \mid 1 \\ \quad \quad \quad \underline{0} \end{array}$$

$$\begin{array}{r} \downarrow \\ \underline{0.125 \times 2} \\ 0.25 \times 2 \\ 0.5 \times 2 \\ 1 \mid 0 \end{array}$$

Таким чином: $23_{10} = 10111_2$; $0.125_{10} = 0.001_2$.

Результат: $23.125_{10} = 10111.001_2$.

Необхідно відзначити, що цілі числа залишаються цілими, а правильні дробу - дробом у будь-якій системі числення.

Для перекладу вісімкового або шістнадцятиричного числа в двійкову форму досить замінити кожен цифру цього числа відповідним трохразрядним двійковим числом (тріадою) (таб. 1) або чотирирозрядним двійковим числом (тетрадою) (таб. 1), при цьому відкидають непотрібні нулі в старших і молодших розрядах. Приклад.

а) Перевести $305.4_8 \rightarrow 2$

$$\begin{array}{cccc} 3 & 0 & 5 & . & 4 \\ \hline 011 & 000 & 101 & & 100 \end{array} = 11000101.1_2$$

б) Перевести $7B2.E_{16} \rightarrow 2$

$$\begin{array}{cccc} 7 & B & 2 & . & E \\ \hline 0111 & 1011 & 0010 & & 1110 \end{array} = 11110110010.111_2$$

Для переходу від двійкової до вісімкової (шістнадцяткової) системи діють таким чином: рухаючись від точки вліво і вправо, розбивають двійкове число на групи по три (чотири) розряду, доповнюючи при необхідності нулями крайні ліву і праву групи. Потім тріаду (тетраду) замінюють відповідною вісімковою (шістнадцятковою) цифрою.

Приклад:

а) Перевести $1101111001.1101_2 \rightarrow 8$ с.с.

$$\begin{array}{cccccc} 001 & 101 & 111 & 001 & . & 110 & 100 \\ \hline 1 & 5 & 7 & 1 & . & 6 & 4 \end{array} = 1571.64_8$$

б) Перевести 1111111011.100111_2 в 16 с.с.

$$\underbrace{0111111011}_{7}.\underbrace{10011100}_{9} = 7FB.9C_{16}$$

Переклад з вісімкової в шістнадцяткову систему і назад здійснюється через двійкову систему за допомогою тріад і тетрад.

Приклад: Перевести 175.24_8 в 16 с.с.

$$\underbrace{1}_{001}\underbrace{7}_{111}\underbrace{5}_{101}.\underbrace{2}_{010}\underbrace{4}_{100}_8 = 111101.0101_2 = \underbrace{0111}_{7}\underbrace{1101}_{D}.\underbrace{0101}_{5}_2 = 7D.5_{16}$$

Результат: $175.24_8 = 7D.5_{16}$.

Правила виконання арифметичних дій над двійковими числами задаються таблицями двійкових додавання, віднімання та множення.

Таблиця двійкового додавання	Таблиця двійкового віднімання	Таблиця двійкового множення
$0+0=0$	$0-0=0$	$0 \times 0=0$
$0+1=1$	$1-0=1$	$0 \times 1=0$
$1+0=1$	$1-1=0$	$1 \times 0=0$
$1+1=10$	$10-1=1$	$1 \times 1=1$

При додаванні двійкових чисел в кожному розряді виробляється додавання цифр доданків і перенесення з сусіднього молодшого розряду, якщо він є. При цьому необхідно враховувати, що $1+1$ дають нуль в даному розряді і одиницю переносу в наступний.

Приклад. Виконати додавання двійкових чисел: а) $X=1101$, $Y=101$;

$$\begin{array}{r} \\ 1 \leftarrow \text{одиниця переносу} \\ X= 1101 \\ Y= 101 \\ \hline X+Y= 10010 \end{array}$$

Результат $1101+101=10010$.

б) $X=1101$, $Y=101$, $Z=111$;

$$\begin{array}{r} \\ 1 \leftarrow \text{одиниця переносу} \\ X= 1101 \\ Y= 101 \\ Z= 111 \\ \hline X+Y+Z= 11001 \end{array}$$

Результат $1101+101+111=11001$.

При відніманні двійкових чисел у цьому розряді при необхідності займається 1 з старшого розряду. Ця займана 1 дорівнює двом 1 даного розряду.

Приклад. Задані двійкові числа $X = 10010$ і $Y = 101$. Обчислити $X-Y$.

$$\begin{array}{r} 10010 \\ -101 \\ \hline 01101 \end{array}$$

Результат $10010 - 101 = 1101$.

Множення двійкових чисел проводиться за тими ж правилами, що і для десяткових за допомогою таблиць двійкового множення і додавання.

Приклад. $1001 * 101 = ?$

$$\begin{array}{r} 1001 \\ \times 101 \\ \hline 1001 \\ 1001 \\ \hline 101101 \end{array}$$

Результат $1001 \square 101 = 101101$.

Ділення двійкових чисел проводиться за тими ж правилами, що і для десяткових.

При цьому використовуються таблиці двійкового множення і віднімання. Приклад. $1100.011 : 10.01 = ?$

$$\begin{array}{r} 110001.1 \quad | \quad 1001 \\ \underline{1001} \quad | \quad 101.1 \\ 1101 \\ \underline{1001} \\ 1001 \\ \underline{1001} \\ 0 \end{array}$$

Результат $1100.011 : 10.01 = 101.1$.

Практичні завдання

варіант	Перевести з 10 в 2 сч.	Перевести з 10 в 2 сч і представити в доповняльному коді	Перевести з 2 в 10 сч	Перевести з 2 в 8 і 16 сч.	Задані числа перевести в 2 систему числення додати та отриманий результат перевести в десяткову систему числення
1	4567	-12	1010101001	010101001	A=56, B=35.
2	3456	-14	1010001111	010001111	A=45, B=20.
3	1345	-34	110100101	110100101	A=72, B=10.
4	9064	-32	1010100100	010100100	A=12, B=35.
5	2445	-56	111101001	111101001	A=56, B=38.
6	2356	-67	101000000	101000000	A=78, B=46.
7	3321	-31	110000110	110000110	A=82, B=14.
8	5678	-51	110111100	110111100	A=30, B=90.

9	6785	-69	111110010	111110010	A=13, B=57.
10	2341	-24	111110000	111110000	A=66, B=30.
11	9987	-43	100101111	100101111	A=57, B=11.
12	7809	-78	100101111	100101111	A=46, B=78.
13	8644	-54	101001001	101001001	A=24, B=24.
14	1145	-21	101100001	101100001	A=16, B=61.
15	3000	-46	101111101	101111101	A=72, B=72.
16	9016	-45	1101100101	1101100101	A=32, B=35.
17	5387	-86	100101100	100101100	A=60, B=44.
18	4243	-28	101010101	101010101	A=24, B=89.
19	3456	-44	100110011	100110011	A=14, B=87.
20	8741	-99	100100100	100100100	A=34, B=35.
21	1231	-23	1101101101	1101101101	A=16, B=35.
22	6573	-56	1000010111	1000010111	A=56, B=34.

До здавання лабораторної роботи надаються: Виконанні завдання та вміння переводу чисел з однієї системи числення в іншу.

Перелік посилань

- Пономарев В.С., Красников В.В. Методические указания по курсу "Организация и функционирование ЭВМ и систем". Ч. 1. Арифметические основы ЭВМ. ДГТУ, 1996.
- Каган Б.М. Электронные вычислительные машины и системы. М.: Энергоатомиздат, 1991.
- Савельев А.Я. Прикладная теория цифровых автоматов. М.: Высшая школа, 1983.

Лабораторна робота № 4

Основи машинної арифметики з двійковими числами.

Мета роботи : отримання практичних навичок з основ машинної арифметики з двійковими числами (коди чисел - прямий , зворотний , додатковий , модифіковані коди)

Короткі теоретичні відомості

Будь-яка інформація (числа , команди , записи тощо) представляється в ЕОМ у вигляді двійкових кодів з фіксованою або змінною довжини. Окремі елементи двійкового коду , що мають значення 0 або 1 , називають розрядами або бітами. Двійковий код, що складається з 8 розрядів носить назву байта. Для запису чисел також використовують 32- розрядний формат (машинне слово) , 16- розрядний формат (півслово) і 64 - розрядний формат (подвійне слово) .

2.1 Коди чисел:

У ЕОМ з метою спрощення виконання арифметичних операцій застосовують спеціальні коди для представлення чисел. Використання кодів дозволяє звести операцію віднімання чисел до арифметичного складання кодів цих чисел. Застосовуються прямий , зворотний і доповняльний коди чисел. Прямий код використовується для представлення від'ємних чисел в пристроях ЕОМ , а також при множенні і діленні . Зворотний і доповняльний коди використовуються для заміни операції віднімання операцією додавання, що спрощує пристрій арифметичного блоку ЕОМ. До кодів висуваються такі вимоги:

- 1) Розряди числа в коді жорстко пов'язані з певною розрядної сіткою.
- 2) Для запису коду знака в розрядної сітці відводиться фіксований , суворо певний розряд.

Наприклад , якщо за основу подання коду взятий один байт , то для представлення числа буде відведено 7 розрядів , а для запису коду знака один розряд .

Прямий код

Прямий код двійкового числа збігається по зображенню із записом самого числа. Значення знакового розряду для додатніх чисел дорівнює 0 , а для від'ємних чисел 1

Знаковим розрядом зазвичай є крайній розряд у розрядної сітці . Надалі при запису коду знаковий розряд домовимося відокремлювати комою. Якщо кількість розрядів коду не вказано будемо припускати , що під запис коду виділений один байт.

Приклад : У разі , коли для запису коду виділений один байт , для числа +1101 прямий код 0,0001101 , для числа -1101 прямий код 1,0001101 .

Обернений код

Обернений код для додатнього числа збігається з прямим кодом. Для від'ємного числа всі цифри числа замінюються на протилежні (1 на 0 , 0 на 1) , а в знаковий розряд заноситься одиниця .

Приклад :

Для числа +1101 прямиий код 0,0001101 ; обернений код 0,0001101 .

Для числа -1101 прямиий код 1,0001101 ; обернений код 1,1110010

Доповняльний код

Доповняльний код додатнього числа збігається з прямиим кодом. Для від'ємного числа доповняльний код утворюється шляхом отримання оберненого коду і додаванням до молодшого розряду одиниці.

Приклад :

Для числа +1101 :

Прямиий код	Обернений код	Доповняльний код
0,0001101	0,0001101	0,0001101

Для числа -1101 :

Прямиий код	Обернений код	Доповняльний код
1,0001101	1,1110010	1,1110011

2.2 Особливості складання чисел в оберненому і доповняльному кодах:

При додаванні чисел в доповняльному кодi виникає одиниця переносу у знаковому розрядi відкидається.

При додаванні чисел у оберненому кодi виникає одиниця переносу у знаковому розрядi, вона додається до молодшого розряду суми кодiв .

Якщо результат арифметичних дій є кодом від'ємного числа , необхідно перетворити його в прямиий код . При цьому обернений код перетворюється в прямиий заміною цифр у всіх розрядах крім знакового на протилежні. Доповняльний код перетворюється в прямиий так , як і зворотний , з подальшим збільшенням одиниці до молодшого розряду .

Приклад:

Скласти двійкові числа X і Y в оберненому і доповняльному кодах:

а) X = 111 , Y = -11 ;

1) Складемо числа , користуючись правилами двійковій арифметики :

$$\begin{array}{r} X = \quad _ 111 \\ Y = \quad _ 11 \\ \hline X+Y = \quad 100 \end{array}$$

2) Складемо числа , використовуючи коди :

Прямиий код Додавання у зворотному кодi Додавання в додатковому кодi

$$\begin{aligned} X_{\text{пр}} &= 0,0000111 \\ Y_{\text{пр}} &= 1,0000011 \end{aligned}$$

$$\begin{array}{r} X_{\text{обр}} = \\ Y_{\text{обр}} = \end{array} + \begin{array}{r} 0,0000111 \\ \underline{1,1111100} \\ 1\ 0,0000011 \end{array}$$

└───────────┘ +1

$$(X+Y)_{\text{обр}} = 0,0000100$$

$$\begin{array}{r} X_{\text{доп}} = \\ Y_{\text{доп}} = \end{array} + \begin{array}{r} 0,0000111 \\ \underline{1,1111101} \\ 1)0,0000100 \end{array}$$

← отбрасывается

$$(X+Y)_{\text{доп}} = 0,0000100$$

Так як результат складання є кодом додатнього числа (знак 0), то $(X + Y)_{\text{обр}} = (X + Y)_{\text{доп}} = (X + Y)_{\text{пр}}$.

б) $X = -101$, $Y = -11$;

1) Складемо числа , користуючись правилами двійковій арифметики :

$$\begin{aligned} X &= -101 \\ Y &= -110 \\ X+Y &= -1011 \end{aligned}$$

2) Складемо числа , використовуючи коди :

Прямий код Додавання у зворотному коді Додавання в додатковому коді

$$\begin{aligned} X_{\text{пр}} &= 1,0000101 \\ Y_{\text{пр}} &= 1,0000110 \end{aligned}$$

$$\begin{array}{r} X_{\text{обр}} = \\ Y_{\text{обр}} = \end{array} + \begin{array}{r} 1,1111010 \\ \underline{1,1111001} \\ 1\ 1,1110011 \end{array}$$

└───────────┘ +1

$$(X+Y)_{\text{обр}} = 1,1110100$$

$$\begin{array}{r} X_{\text{доп}} = \\ Y_{\text{доп}} = \end{array} + \begin{array}{r} 1,1111011 \\ \underline{1,1111010} \\ 1)1,1110101 \end{array}$$

← отбрасывается

$$(X+Y)_{\text{доп}} = 1,1110101$$

Оскільки сума є кодом негативного числа (знак 1), то необхідно перевести результати в прямий код :

- З оберненого коду:

$$(X + Y)_{\text{обр}} = 1,1110100 \text{ ----- } (X + Y)_{\text{пр}} = 1,0001011 ;$$

- З доповняльному коду:

$$(X + Y)_{\text{доп}} = 1,1110101 \text{ ----- } (X + Y)_{\text{пр}} = 1,0001010 + 0,0000001 = 1,0001011 .$$

Таким чином , $X + Y = -1011$ і отриманий результат збігається із звичайною записом

2.3 Модифіковані обернений і доповняльний коди .

При переповненні розрядної сітки , відбувається перенесення одиниці в знаковий розряд. Це призводить до неправильного результату , причому позитивне число , що вийшло в результаті арифметичної операції може сприйматися як негативне (так як у знаковому розряді " 1") і навпаки.

Наприклад:

$$\begin{aligned} X &= 0,1010110 \\ Y &= 0,1101000 \\ X+Y &= 1,0111110 \end{aligned}$$

Тут X і Y - коди додатніх чисел , але ЕОМ сприймає результат їх складання як код від'ємного числа ("1" у знаковому розряді) . Для виявлення переповнення розрядної сітки вводяться модифіковані коди .

У модифікованому оберненому і модифікованому доповняльному кодах під знак числа відводиться не один , а два розряди : "00" відповідає знаку " + " , " 11 " - знаку " - ". Будь-яка інша комбінація ("01 " або " 10") , що вийшла в знакових розрядах служить ознакою переповнення розрядної сітки. Додавання чисел в модифікованих кодах нічим не відрізняється від складання у звичайних оберненому і доповняльному кодах.

Розглянемо попередній приклад , виконавши додавання в модифікованому зворотному коді :

$$\begin{array}{r} X = 00,101011 \\ Y = 00,110100 \\ \hline X + Y = 01,011111 \end{array}$$

Комбінація "01" в знакових розрядах означає , що сталося переповнення і вийшов результат - невірний .

Розглянемо ще один приклад .

Приклад . Дано два числа: X = 101001 і Y = -11010 . Скласти їх в модифікованому додатковому коді .

1) Переведемо X і Y в модифікований додатковий код:

$$X = +101001$$

$$Y = -011010$$

Звичайна запис

Модифікований
обернений код

Модифікований
доповняльний код

$$X = +101001$$

$$X_{об\ p}^m = 00,101001$$

$$X_{доп}^m = 00,101001$$

$$Y = -011010$$

$$Y_{об\ p}^m = 11,100101$$

$$Y_{доп}^m = 11,100110$$

2) Виконаємо додавання :

$$\begin{array}{r} X_{доп}^m = 00,101001 \\ Y_{доп}^m = 11,100110 \\ \hline 1) 00,001111 \\ \leftarrow \text{отбрасывается} \\ (X+Y)_{доп}^m = 00,001111 \end{array}$$

Переповнення немає (в знакових розрядах "00") , тому отриманий результат - вірний (X + Y = 1111)

Практичні завдання

1) Записати число в прямому , оберненому і доповняльних кодах :

- V1 –11010 ;
- V2 – -11101 ,
- V3 – 101001 ,
- V4 – -1001110 ,
- V5 – 1010101,
- V6 – -1010100,
- V7 – -111101001,
- V8 – -0011110101,
- V9 –1010111011,
- V10 –1010101,
- V11 – -1110,
- V12 –0001,
- V13 – -0101,
- V14 – -110001,
- V15 –010001.

2) Перевести X і Y в прямій , зворотний і додатковий коди . Скласти їх у зворотному і додатковому кодах. Результат перевести в прямий код . Перевірити отриманий результат , користуючись правилами двійковій арифметики.

- V1 – X = -11010 ; Y = 1001111 ;
- V2 –X = -11101 ;Y = -100110 ;
- V3 – X = 1110100 ;Y = -101101 ;
- V4 – X = -10110 ;Y = -111011 ;
- V5 – X = 1111011 ;Y = -1001010 ;
- V6 – X = -11011 ;Y = -10101 .
- V7 – X = 1111011 ;Y = -1001010 ;
- V8 – X = 1001011 ;Y = -1010010 ;
- V9 – X = 1000011 ;Y = -1111010 ;
- V10 – X = 110011 ;Y = -101010 ;
- V11 – X = -1001011 ;Y = 1101010 ;
- V12 – X = 1100011 ;Y = -1111010 ;
- V13 – X = -1110011 ;Y = -11010 ;
- V14 – X = -11011 ;Y = 1001010 ;
- V15 – X = 1100011 ;Y = 1011010 .

3) Скласти X і Y в модифікованому зворотному і модифікованому додатковому восьмирозрядних кодах. У разі появи ознаки переповнення збільшити число розрядів в кодах і повторити підсумовування . Результат перевести в прямий код і перевірити , користуючись правилами двійковій арифметики.

B1X = 10110 ; Y = 1111101 ;
B2X = 11110 ; Y = -1001 ;
B3X = -11110 ; Y = -10111 ;
B4X = -11101 ; Y = -101011 ;
B5X = -1001 ; Y = 10010 ;
B6X = -1001 ; Y = -10011 .
B7X = -101 ; Y = -100011 .
B8X = -1001 ; Y = -111011 .
B9X = -1001 ; Y = -0011 .
B10X = -10001 ; Y = -1011 .

m - мантиса числа A ($|m| < 1$).

Так як в ЕОМ застосовується двійкова система числення , то

$A = m * 2^p$, причому порядок і мантиса представлені в двійковій формі.

Двійкове число називається нормалізованим , якщо його мантиса задовольняє нерівності

$$1/2 < |m| < 1.$$

Нерівність показує , що двійкове число є нормалізованим , якщо в старшому розряді мантиси варто одиниця . Наприклад , число $0,110100 * 10100$ - нормалізоване , а $0,001101 * 10110$ - ненормалізоване .

Ситуація , коли в процесі обчислень отримано число з $|m| > 1$ називається переповненням розрядної сітки.

Нормалізоване подання чисел дозволяє зберегти в розрядній сітці більша кількість значущих цифр і, отже , підвищує точність обчислень . Однак сучасні ЕОМ дозволяють , при необхідності , виконувати операції також і над ненормалізованих числами.

Діапазон представлення нормалізованих двійкових чисел , узятих за абсолютним значенням , задовольняє нерівності :

$$2^{-1} * 2^{-k} < A < (1 - 2^{-1}) * 2^{2k-1} ,$$

де l - число розрядів мантиси ;

k - число розрядів порядку ;

2^{-1} - найменше значення нормалізованої мантиси ;

$1 - 2^{-1}$ - найбільше значення нормалізованої мантиси .

Широкий діапазон представлення чисел з плаваючою комою зручний для наукових і інженерних розрахунків . Для підвищення точності обчислень в багатьох ЕОМ передбачена можливість використання формату подвійної довжини , однак при цьому відбувається збільшення витрат пам'яті на зберігання даних і сповільнюються обчислення.

Область значень визначається за кількістю розрядів у експоненті , а точність визначається за кількістю розрядів у мантиси . Існує кілька способів подання того чи іншого числа , тому одна форма вибирається в якості стандартної . Щоб вивчити властивості такого способу подання , роздивимось уявлення R з трьох-розрядного мантисою зі знаком в діапазоні $0 < |f| < 1$ і дво-розрядне експонентою зі знаком . Ці числа знаходяться в діапазоні від $+0,100 * 10^{-99}$ до $+0,999 * 10^{+99}$ тобто

простягаються майже на 199 значущих розрядів , хоча для запису числа потрібно всього 5 розрядів і 2 знаки .

1. Від'ємні числа менші $-0,999 \times 10^{99}$.
2. Від'ємні числа від $-0,999 \times 10^{99}$ до $-0,100 \times 10^{-99}$.
3. Від'ємні числа від $-0,100 \times 10^{-99}$ до нуля.
4. Нуль.
5. Додатні числа від 0 до $0,100 \times 10^{99}$.
6. Додатні числа від $0,100 \times 10^{99}$ до $0,999 \times 10^{99}$.
7. Додатні числа від 0 до $0,999 \times 10^{99}$.

Числа з плаваючою точкою можна використовувати для моделювання системи дійсних чисел в математиці, хоча тут є кілька суттєвих відмінностей. На рис. 3.1 представлена вісь дійсних чисел. Вона розбита на 7 областей:

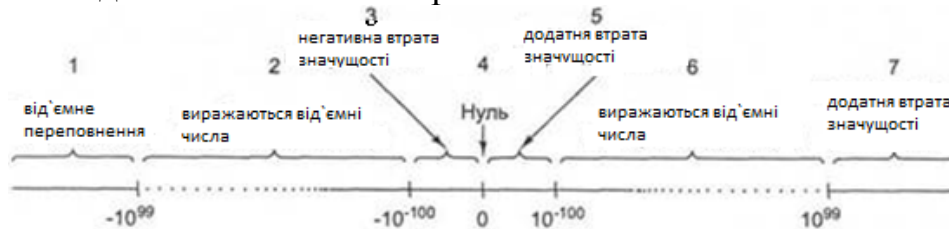


Рис. 3.1. Вісь дійсних чисел розбита на 7 областей

Перша відмінність дійсних чисел від чисел з плаваючою точкою , які записуються трьома розрядами у мантиси і двома розрядами в експоненті , полягає в тому , що останні не можна використовувати для запису чисел з областей 1,3 , 5 і 7. Якщо в результаті арифметичної операції вийде число з області 1 або 7 (наприклад , $10^{60} \times 10^{60} = 10^{120}$), то відбудеться помилка переповнення і результат буде невірним. Причина - обмеження області значень чисел у цьому поданні. Точно так само не можна виразити результат з області 3 або 5 . Така ситуація називається помилкою через втрати значущості. Ця помилка менш серйозна, ніж помилка переповнення , оскільки часто нуль є цілком задовільним наближенням для чисел з областей 3 або 5 . Залишок рахунку в банку на 10^{-102} не сильно відрізняється від залишку рахунку 0 .

Друга важлива відмінність чисел з плаваючою комою від дійсних чисел - це їх щільність . Між будь-якими двома дійсними числами x і y існує інше дійсне число незалежно від того , наскільки близько до y розміщений x . Це властивість впливає з того , що між будь-якими різними дійсними числами x і y існує дійсне число $z = (x + y) / 2$. Дійсні числа формують континуум .

Числа з плаваючою точкою континууму не формують . У двухзначовій п'ятирозрядній системі можна висловити рівно 179100 позитивних чисел , 179100 негативних чисел і 0 (який можна виразити різними способами) , тобто всього 358 201 чисел. З нескінченного числа дійсних чисел в діапазоні від -10^{-100} до $+0,999 \times 10^{99}$ в цій системі можна виразити тільки 358201 число. На рис. 3.1 ці числа показані точками . Результат обчислень може бути й іншим числом , навіть якщо він знаходиться в області 2 або 6 . Наприклад , результат ділення числа $+0,100 \times 10^3$ на 3 не можна виразити точно в нашій системі подання . Якщо отримане число не можна

виразити в використовуваної системі подання , потрібно брати найближчим число , яке представимо у цій системі . Такий процес називається округленням .

Проміжки між суміжними числами , які можна виразити в уявленні з плаваючою комою , в другій і шостій областях не постійні. Проміжок між числами $+0,998 \times 10^{99}$ і $+0,999 \times 10^{99}$ набагато більше проміжку між числами $+0,998 \times 10^0$ і $+0,999 \times 10^0$. Однак якщо проміжки між числом і його сусідом виразити як процентне відношення від цього числа , великої різниці в проміжку не буде. Іншими словами , відносна похибка , отримана при округленні , приблизно дорівнює і для малих , і для великих чисел .

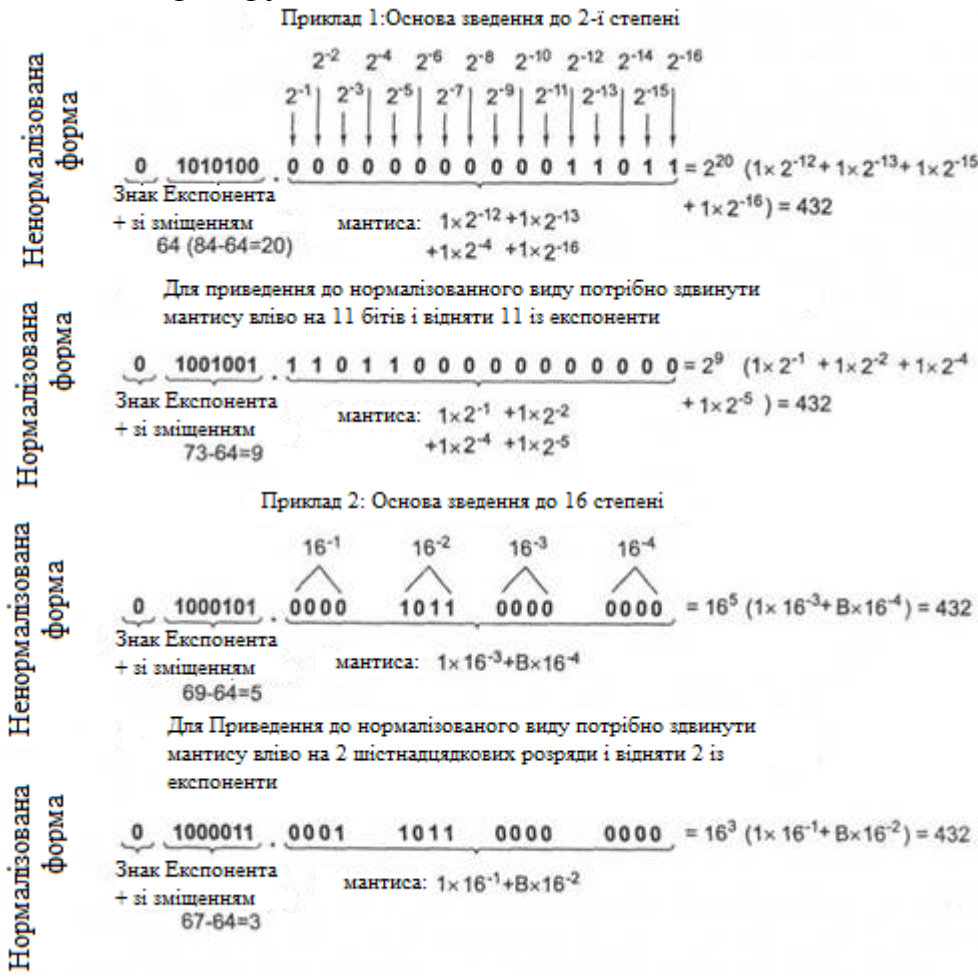
Висновки, зроблені для системи подання з трьохразрядною мантиєю і двох розрядною експонентою , справедливі і для інших систем представлення чисел. При зміні числа розрядів у мантиї або експоненті просто зсовуються кордону другої та шостої областей і міняється число експонованих одиниць в цих областях. Зі збільшенням числа розрядів у мантиї збільшується щільність елементів і , отже , точність наближень . Із збільшенням кількості розрядів у експоненті розмір областей 2 і 6 збільшується за рахунок зменшення областей 1 , 3 , 5 і 7. У табл. 3.1 показані приблизні межі області 6 для десяткових чисел з плаваючою точкою з різною кількістю розрядів у мантиї і експоненті.

Кількість розрядів мантиї	Кількість розрядів в експоненті	Нижня границя	Верхня границя
3	1	10^{-12}	10^9
3	2	10^{-102}	10^{99}
3	3	10^{-1002}	10^{999}
3	4	10^{-10002}	10^{9999}
4	1	10^{-13}	10^9
4	2	10^{-103}	10^{99}
4	3	10^{-1003}	10^{999}
4	4	10^{-10003}	10^{9999}
5	1	10^{-14}	10^9
5	2	10^{-104}	10^{99}
5	3	10^{-1004}	10^{999}
5	4	10^{-10004}	10^{9999}
10	3	10^{-1009}	10^{999}
20	3	10^{-1019}	10^{999}

Варіант такого подання застосовується в комп'ютерах. Основа зведення в ступінь - 2, 4, 8 або 16, але не 10. У цьому випадку мантия складається з ланцюжка двійкових, четвіркових, вісімкових і шістнадцяткових розрядів. Якщо крайній лівий розряд дорівнює 0, всі розряди можна змістити на один вліво, а експоненту зменшити на 1, не змінюючи при цьому значення числа (виняток становить ситуація

втрата значимості). Мантиса з ненульовим крайнім лівим розрядом називається нормалізованою.

Нормалізовані числа зазвичай приймаються ненормалізовані, оскільки існує тільки одна нормалізована форма, а ненормалізованих форм може бути багато. Приклади нормалізованих чисел з плаваючою точкою дано на рис. 3.2. для двох основ зведення в ступінь. У цих прикладах показана 16-бітна мантиса (включаючи знаковий біт) і 7-бітна експонента. Кома знаходиться зліва від крайнього лівого біта мантиси і праворуч від експоненти.



Стандарт IEEE 754

До 80 - х років кожен виробник мав свій власний формат чисел з плаваючою крапкою. Всі вони відрізнялися один від одного. Більше того , в деяких з них арифметичні дії виконувалися неправильно , оскільки арифметика з плаваючою точкою має деякі тонкощі , які не очевидні для звичайного розробника апаратного забезпечення.

Щоб змінити цю ситуацію , в кінці 70 - х років IEEE заснував комісію для стандартизації арифметики з плаваючою крапкою. Метою було не тільки дати можливість переносити дані з одного комп'ютера на інший , а й забезпечити розробника апаратного забезпечення завідомо правильною моделлю . У результаті вийшов стандарт IEEE 754 (IEEE , 1985). В даний час більшість процесорів (у тому

числі Intel , SPARC і JVM) містять команди з плаваючою точкою , які відповідають цьому стандарту . На відміну від багатьох стандартів , які представляли собою невдалі компроміси і мало кого влаштовували , цей стандарт непоганий, більшою мірою завдяки тому , що його спочатку розробляв одна людина , професор математики університету Берклі Вільям Каган (William Kahan) . Цей стандарт буде описаний нижче.

Стандарт визначає три формату : з одинарної точністю (32 біта) , з подвоєною точністю (64 біта) і з підвищеною точністю (80 бітів) . Формат з підвищеною точністю призначений для скорочення помилок округлення. Він застосовується головним чином в арифметичних пристроях з плаваючою точкою , тому ми не будемо про нього говорити . У форматах з одинарної і подвоєною точністю застосовується підстава зведення в ступінь 2 для мантис і зміщена експонента . Формати представлені на рис. 3.3.

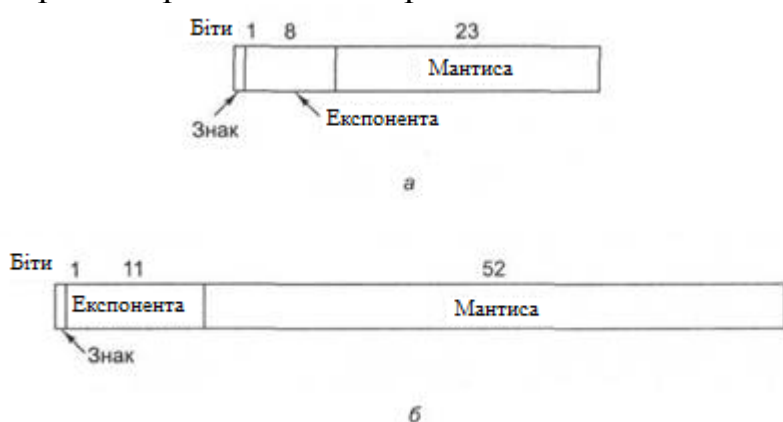


Рис. 3. 3. Формати для стандарту IEEE з плаваючою крапкою:

одинарна точність (а);

подвоєна точність (б).

Обидва формату починаються зі знакового біта для всього числа:

- знак - це один біт визначає знак числа (0 вказує на позитивне число, а 1 - на негативне);
- показник ступеня (експонента) - це 8-й бітове доповнення до 127 показника ступеня, якщо реальний показник ступеня дорівнює 0 то в це поле буде записано число 127 (Для формату одинарної точності зсув (excess) 127, а для формату подвоєною точності зсув 1023);
- мантиса - це мантиса числа (права частина від десяткового дробу про 23 і 52 біти відповідно), значуща частина числа з лівого боку від десяткової точки завжди дорівнює 1 за винятком випадку, коли саме число дорівнює нулю. Індикатором рівності усього числа нулю є рівність нулю показника ступеня. Нормалізована мантиса починається з двійкової комою, за якою слідує 1 біт, а потім залишок мантиси.

Знаючи чому дорівнюють окремі поля, можна легко розрахувати чому одно число, скориставшись наступною формулою:

$$\text{Число} = (-1)^{\text{Знак}} \cdot 2^{(\text{показник степеня} - 127)} \cdot 1.\text{мантиса}$$

Слідуючи практиці, розпочатої з комп'ютера PDP -11, користувачі усвідомили, що 1 біт перед мантисою зберігати не потрібно, оскільки можна просто припускати, що він є. Отже, стандарт визначає мантису наступним чином. Вона складається з неявного біта, який завжди дорівнює 1, неявної двійкової комою, за якими йдуть 23 або 52 довільних біта. Якщо всі 23 або 52 біта мантиси рівні 0, то мантиса має значення 1,0. Якщо всі біти мантиси рівні 1, то числове значення мантиси трохи менше, ніж 2,0. Щоб уникнути плутанини в англійській мові для позначення комбінації з неявного біта, неявної двійкової комою і 23 або 52 явних бітів замість терміна «мантиса» (mantissa) використовується термін significand. Всі нормалізовані числа мають significand s в діапазоні $1 \leq s < 2$.

Числові характеристики стандарту IEEE для чисел з плаваючою точкою дані в табл. 3.2.

Таблиця 3.2. Характеристики чисел з плаваючою точкою стандарту IEEE

Параметр	Одинична точність	Подвоєнна точність
Кількість бітів в знаку	1	1
Кількість бітів в експоненті	8	11
Кількість бітів в мантисі	23	52
Загальна кількість бітів	32	64
Зміщення експоненти	Зміщення(excess)127	Зміщення(excess)1023
Область значення експоненти	Від-126 до +127	Від -1022 до+1023
Саме менше нормалізоване число	2^{-126}	2^{-1022}
Саме більше нормалізоване число	-2^{128}	-2^{-1024}
Діапазон десяткових дробів	-10^{-38} до 10^{38}	-10^{-308} до 10^{308}
Саме менше не нормалізоване число	-10^{-324}	-10^{-324}

Традиційні проблеми, пов'язані з числами з плаваючою точкою, - що робити з переповненням, втратою значимості і неініціалізованих числами. Підхід, який використовується в стандарті IEEE, почасти запозичений від машини CDC 6600. Крім нормалізованих чисел в стандарті передбачено ще 4 типи чисел (рис. 3.4).

Проблема виникає в тому випадку, якщо абсолютне значення (модуль) результату менше найменшого нормалізованого числа з плаваючою точкою, яке можна представити у цій системі. Раніше апаратне забезпечення діяло одним із двох

способів: або встановлювало результат на 0, або викликало помилку через втрату значущості. Жоден з цих двох способів не є потрібним, тому в стандарт IEEE введені ненормалізовані числа. Ці числа мають експоненту 0 і мантису, представлену наступними 23 або 52 бітами. Неявний біт 1 ліворуч від двійкової комою перетворюється в 0. Ненормалізовані числа можна легко відрізнити від нормалізованих, оскільки в останніх не може бути експоненти 0.

нормалізоване число	±	$0 < \text{Exp} < \text{Max}$	Будь-який набір бітів
ненормалізоване число	±	0	Будь-який нульовий набір бітів
нуль	±	0	0
безкінченність	±	111...1	0
не число	±	111...1	Будь-який нульовий набір бітів

↙
Знаковий біт

Рис. 3 .4. Числові типи стандарту IEEE

Саме маленьке нормалізоване число з одинарною точністю містить 1 в експоненті і 0 у мантисі і являє $1,0 \times 2^{-126}$. Найбільше ненормалізоване число містить 0 в експоненті і всі одиниці в мантисі і являє приблизно $0,9999999 \times 2^{-127}$, тобто майже те ж саме число. Слід зазначити, що це число містить тільки 23 біта значущості, а всі нормалізовані числа - 24 біта.

У міру зменшення результату при подальших обчисленнях експонента раніше залишається рівною 0, а перші кілька бітів мантиси перетворюються на нулі, що скорочує і значення, і число значущих бітів мантиси. Саме маленьке ненульове ненормалізоване містить 1 в крайньому правому біте, а всі інші біти рівні 0. Експонента представляє 2^{-127} , а мантиса - 2^{-23} , тому значення дорівнює 2^{-150} . Така схема передбачає поступове зникнення значущих розрядів, а не перескакує на 0, коли результат не можна виразити у вигляді нормалізованого числа.

У цій схемі присутні 2 нуля, позитивний і негативний, що визначаються за знаковою біту. Обидва мають експоненту 0 і мантису 0. Тут теж біт зліва двійкової комою за умовчанням 0, а не 1.

З переповненням не можна впоратися поступово. Замість цього існує спеціальне подання нескінченності: з експонентою, яка містить всі одиниці, і мантисою, рівною 0. Це число можна використовувати як операнд. Воно підпорядковується звичайним математичним правилам для нескінченності. Наприклад, нескінченність і будь-яке число в сумі дають нескінченність. Кінцеве число розділити на нескінченність дорівнює 0. Будь-яке кінцеве число, розділене на 0, прагне до нескінченності.

А що вийде, якщо нескінченність розділити на нескінченність? Результат не межа. Для такого випадку існує інший спеціальний формат, NaN (Not a Number - не число). Його теж можна використовувати як операнд.

Практичні завдання

1. Перетворіть наступні числа в формат стандарту IEEE з одинарною точністю. Результати подайте у восьми шістнадцяткових розрядах.

- а. 9
- б. $5/32$
- в. $-5/32$
- г. 6.125

2. Перетворіть наступні числа з плаваючою точкою одинарної точності з шістнадцяткової в десяткову систему числення:

- а. 42E28000 H
- б. 3F880000 H
- в. 00800000 H
- г. C7F00000 H

3. Число з плаваючою точкою у форматі одинарної точності в IBM / 370 зі стоїть з 7-бітною зміщеною експоненти (зміщення 64), 24-бітною мантиєю і знакового біта. Двійкова кома знаходиться зліва від мантиї. Підстава зведення в ступінь - 16. Порядок полів - знаковий біт, експонента, мантия. Виразіть число $7/64$ у вигляді нормалізованого шістнадцятирічного числа в цій системі.

4. Наступні двійкові числа з плаваючою точкою складаються з знакового біта, зміщеною експоненти (зміщення 64) з основою 2 і 16-бітною мантиєю. Нормалізуйте їх.

- а. 0 1000000 0001010100000001
- б. 0 0111111 0000001111111111
- в. 0 1000011 1000000000000000
- г. 0 1110011 1111110000000000
- д. 1 0001100 0001110001110000

5. Щоб скласти два числа з плаваючою точкою, потрібно вирівняти експоненти (зсунувши мантию). Потім можна скласти мантиї і нормалізувати результат, якщо в цьому є необхідність. Складіть числа одинарної точності 3EEO0000H і 3 D 800000 H і висловіть нормалізований результат у шістнадцятковій системі числення.

До складання лабораторної надаються: Зроблені вправи.

ЛАБОРАТОРНА РОБОТА № 6

АРХІТЕКТУРА ЕОМ Фон Неймана. НАВЧАЛЬНІ МАШИНИ

Мета роботи . Ознайомлення з принципами виконання програм комп'ютером на прикладі навчальних машин.

Короткі теоретичні відомості

§1. Принципи Фон-Неймана.

1. Властивості ЕОМ.

Метою створення перших обчислювальних машин було полегшити, спростити громіздкі арифметичні обчислення, які доводилося виконувати при вирішенні фізичних та інженерних завдань. Для того щоб виробництво обчислювальної машини економічно виправдало себе, потрібно щоб:

1) машина була універсальною - придатною для вирішення не однієї конкретної задачі, а цілого класу задач;

2) машина повинна володіти достатнім швидкодією (швидкістю обчислень).

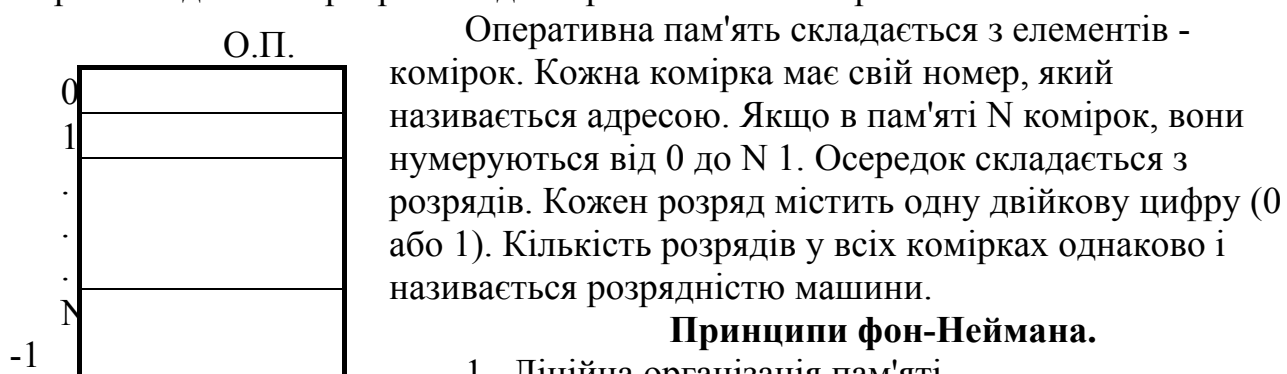
Чим вище швидкодія, тим більше завдань вирішує машина за фіксований відрізок часу. Тим ефективніше робота машини:

3) вартість виробництва машини не повинна бути дуже великою.

2. Принципи Фон-Неймана

У 1943 р. американський математик Джон фон Нейман описав як, на його думку, повинна бути влаштована машина для обчислень. Сформульовані ним принципи отримали назву "принципів фон Неймана", а машини, побудовані відповідно з ними, стали називати фон-Нейманівськими. Більшість сучасних ЕОМ є фон-Нейманівськими.

Основними частинами ЕОМ є процесор і пам'ять. Процесор керує роботою комп'ютера; забезпечує виконання програм. Пам'ять (оперативна пам'ять) служить для зберігання даних і програми під час роботи комп'ютера.



Принципи фон-Неймана.

1 . Лінійна організація пам'яті.

Комірки пам'яті розташовуються послідовно по зростанню номерів.

2 . Прямий доступ до елементів пам'яті.

Доступ до комірки здійснюється за її адресою , в кожен момент роботи комп'ютера можна звернутися до будь-якій комірці пам'яті.

Цей принцип забезпечує полегшення програмування , зручність і надійність використання ЕОМ. (Згадайте машину Тьюринга . Для доступу до комірки , наприклад , яка знаходиться, на три комірки правіше даної , було потрібно вводити три додаткових стани .)

3 . Використання двійкової системи для зберігання та обробки інформації.

Цей принцип слід перш за все з практичних міркувань : досить легко за допомогою електронних пристроїв реалізувати два можливих стани - 0 і 1 .

4 . Принцип збереженої програми .

Програма , що управляє процесом обчислень , зберігається в пам'яті машини .

Цей принцип забезпечує універсальність ЕОМ. (Порівняймо з машиною Тьюринга : кожна машина Тьюринга мала одну програму і могла вирішувати тільки одну задачу !)

5 . Машинні операції .

Існує набір дій з обробки даних , виконуваних апаратно (реалізованих у вигляді електронних схем) . Ці дії називаються машинними операціями .

Чим більше машинних операцій , тим легше програмувати для ЕОМ і тим вище її швидкодію. (Згадайте , щоб додати 1 до числа за допомогою МТ , було потрібно написати досить об'ємну програму.)

Кожній машинній операції відповідає машинна команда - після - довність нулів і одиниць , яку може зрозуміти і виконати процесор.

Таким чином, містяться в комірці пам'яті нулі й одиниці можуть зображати дане, а можуть бути командою. Що ж саме записано в комірці - дане чи команда - визначається під час роботи ЕОМ . Надалі ми обговоримо детальніше це питання.Итак, команда - это приказ процессору выполнить машинную операцию. Последовательность команд называется программой.

6. Послідовне виконання команд.

Команди, записані в пам'яті комп'ютера, виконуються послідовно, один за одним.

§2. Загальна структура ЕОМ.

Обчислювальна в цілому машина складається з наступних компонентів



Призначення компонентів.

- | | |
|--------------------|---|
| Процесор | – керує роботою ЕОМ, забезпечує виконання програм. |
| Оперативна пам'ять | – використовується для зберігання даних и програм в роботі ЕОМ. |

Зовнішні пристрої – слугують для зв'язку ЕОМ із зовнішнім світом.
Розглянемо структуру і роботу кожного з компонентів.

П.1. Процесор. Такт роботи процесора.

Процесор включає в собі наступні пристрої.



АЛП (арифметичний пристрій) – виконує арифметичні і логічні операції (наприклад, складання, віднімання, множення)

ПУ (пристрій управління) – керує роботою процесора.

Регістри - спеціальні комірки, які знаходяться в ЦП.

РК (регістр команди) містить машинну команду, яку виконує в даний момент процесор.

ЛА (лічильник адреси) містить адресу наступної команди.

СС (слово-стан) містить інформацію про результат виконання команди.

Виконання процесором однієї машинної команди назвемо тактом роботи процесора.

Розглянемо, як процесор виконує машинну команду на прикладі команди

$$\underbrace{01}_{\text{КОП}} \underbrace{1011}_{\text{A1}} \underbrace{1100}_{\text{A2}} \underbrace{1011}_{\text{A3}}$$

Нехай 01 покаже код операції (КОП) " складання "; A1, A2, A3 - адреса першого операнда, другого операнда і результату відповідно.

1. В РК рахується із ОП команда, адрес якої записаний в ЛА.

2. Склад ЛА збільшується на 1, так що тепер в ЛА отримали адрес наступної команди програми.

3.ПУ аналізує склад РК и організує виконання команди.

Виділяється КОП. Визначається, що потрібно виконати операцію "складання". Визначаються A1, A2, A3. Вміст комірок ОП з адресами A1, A2 пересилаються в АЛУ. Далі АЛУ виконує дію складання. Результат складання з АЛУ пересилається в осередок пам'яті з адресою A3. У реєстр СС записується інформація про вдале(чи невдалому) закінчення виконання складання.

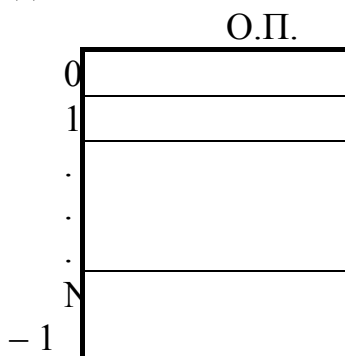
Далі робота повторюється з першого кроку.

Раніше ми відмічали, що осередок ОП може містити дані або команду. Тепер зрозуміло, як процесор відрізняє дані від команди: якщо адреса комірки зустрілася в команді як адресу операнда, процесор обробляє вміст комірки як дане; якщо адреса комірки вийшла в лічильнику адреси, процесор обробляє вміст комірки як команду. Вміст одного і того ж комірки водночас роботи процесора може трактуючи як дане, а в іншій - як команда.

Перед початком роботи процесора в реєстр СА записується апаратно завжди одна і та ж адреса, і перша команда програми повинна розташовуватися в ОП в комірки саме з цією адресою.

П.2. Оперативна пам'ять.

Ми торкалися облаштування оперативної пам'яті в (1. Нагадаємо основні відомості.



Оперативна пам'ять(ОП) складається з комірок. Кожен осередок має свою адресу - число від 0 до N - 1. Кількість комірок(N) називається об'ємом ОП. Помітимо, що об'єм ОП і розмір реєстра СА взаємозв'язані: кількість розрядів в СА має бути достатня для зберігання будь-якої можливої адреси. (Найбільша можлива адреса в нашому припущенні N - 1.)

Комірки складаються з розрядів. Кількість розрядів в усіх комірках однакова і називається розрядністю машини.

Кожен розряд містить одну двійкову цифру. Іноді розряди називаються бітами. У програмуванні слово "біт" використовують в двох сенсах:

Біт - один двійковий розряд комірки.

- вміст одного двійкового розряду.

Вміст комірки називають словом або машинним словом.

Вміст ОП. Комірки можуть зберігати дані і команди. Команди, що становлять програму, зазвичай розташовуються в ОП послідовно, один за одним. Що саме містить осередок - дане або команду - визначається у момент використання вмісту комірки.

Робота ОП. Помітимо, що осередок ОП завжди має деякий вміст. Насправді, електронний пристрій, що є розрядом в комірки, обов'язково знаходиться в якому-небудь стані; цей стан зображує "0" або "1". Таким чином, осередок заповнений нулями і одиницями. Проте цей вміст не має ніякого сенсу. Для того, щоб можна було обробляти дане, таке, що міститься в комірки, потрібно спочатку це дане записати в осередок.

Итак, є дві основні операції роботи з ОП:

1. Запис даного в осередок.

ЦП повідомляє ОП, чт(саме потрібно записати і за якою адресою. При записі в осередок її старий вміст втрачається, стає недоступним.

2. Читання вмісту комірки.

ЦП передає ОП потрібна адреса. Вміст комірки з цією адресою прочитується і пересилається в ЦП. При читанні вміст комірки не змінюється.

Читання і запис в ОП робиться спеціальними електронними схемами. При виключенні обчислювальної машини вміст ОП втрачається.

П.3. Зовнішні пристрої.

Зовнішні пристрої служать для зв'язку ЕОМ з навколишнім світом. Зазвичай до ЕОМ підключаються клавіатура, монітор(екран, дисплей), зовнішні запам'ятовуючі пристрої(жорсткі диски, дисководи для гнучких дисків) і миша. Залежно від того, для яких цілей використовують ЕОМ, набір підключених до ЕОМ пристроїв може сильно змінюватися. До зовнішніх пристроїв відносяться:

- принтер(друкувальний пристрій)
 - пристрій для роботи з магнітними стрічками
 - сканер(пристрій для введення графічних зображень)
 - модем(пристрій для зв'язування комп'ютерів через телефонну мережу)
 - облаштування читання з лазерних дисків
- і інші спеціальні прилади.

Для підключення зовнішніх пристроїв в ЕОМ є канали.



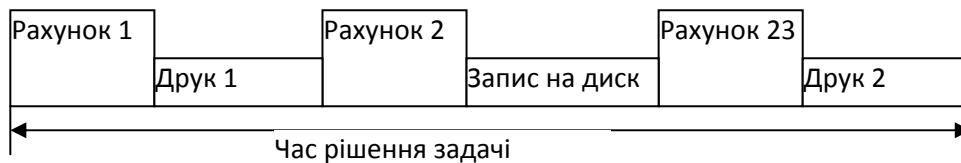
Канал - апаратура і програмне забезпечення(програми), що займаються передачею сигналів між ЕОМ і зовнішнім пристроєм.

У зовнішньому пристрої є контролер.

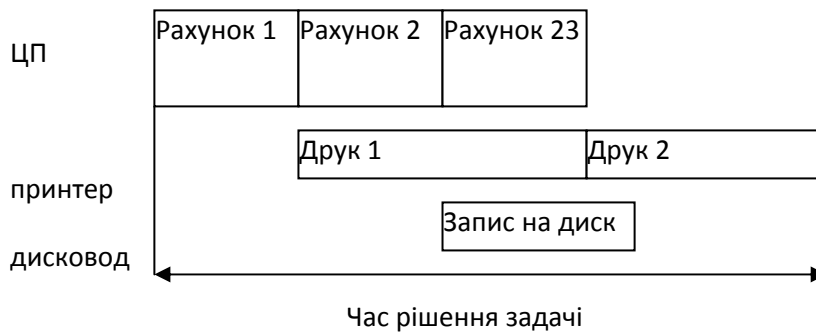
Контролер - апаратура і програмне забезпечення, яке обробляє сигнали ЕОМ і готує дані для пристрою. Контролер враховує особливості роботи свого зовнішнього пристрою.

Зовнішні пристрої працюють набагато повільніше за процесор. Для того, щоб процесор не простоював під час роботи зовнішнього пристрою, організовується паралельна робота процесора і зовнішнього пристрою.

Нехай процес рішення деякої задачі складається з трьох відрізків рахунку(роботи ЦП), двох друку і одного запису даних на диск:



Після закінчення етапу Рахунок 1, коли отримані дані для першого друку, принтер може зайнятися друком, а ЦП може продовжити рішення задачі. Після отримання даних для диска, дисковод починає записувати, а ЦП може продовжити роботу. У результаті, загальний час рішення задачі скоротиться.



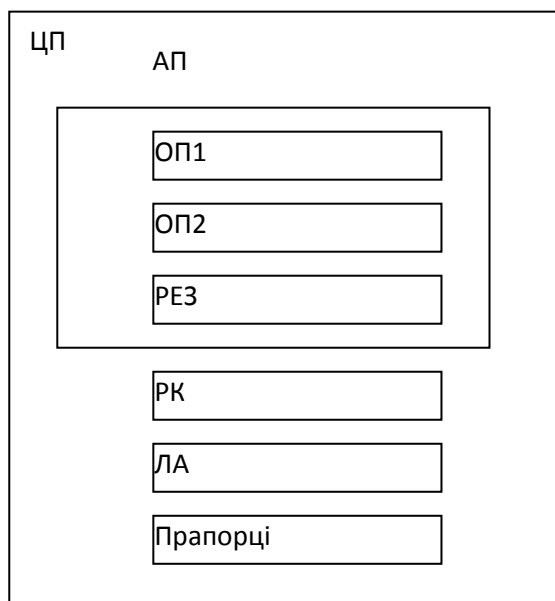
Паралельна організація роботи пристроїв дозволяє підвищити ефективність використання ЕОМ, збільшити швидкість, забезпечуючи виконання другої властивості обчислювальних машин.

§3. Учебні машини.

Раніше ми розглянули загальні принципи побудови ЕОМ. У цьому параграфі ми розберемо 2 учбових обчислювальних машини. Ці машини називаються учбовими, тому що вони не існують насправді. Проте вони допоможуть нам представити роботу реальних ЕОМ, розібрати деякі проблеми, що виникають при побудові ЕОМ, і способи рішення цих проблем.

Для запису вмісту комірок користуватимемося шістнадцятковими цифрами.

П.1. Учебна трьохадресна машина УМ- 3.



1. Структура процесора.

Окрім тих регістрів, які ми обговорювали в (2, в арифметичному облаштуванні УМ- 3 є регістри першого операнда ОП1, другого операнда ОП2 і результату РЕЗ.

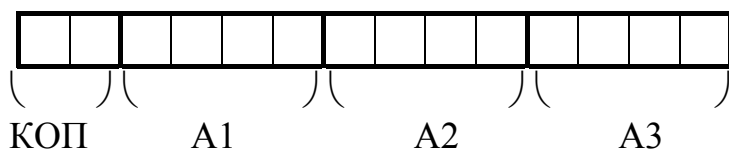
2. Оперативна пам'ять.

Осередок - 14 шістнадцяткових розрядів.
Об'єм ОП - 164 комірок з адресами 0000-FFFF.

Представлення чисел : число міститься в одному комірки; негативні числа записуються в додатковому коді.

Представлення команд : команда займає один осередок. Розряди комірки мають

наступний сенс(формат команд) :



Тут КОП - код операції(вказує, яку машинну операцію потрібно виконати); A1, A2, A3 - адреси першої, другої операндів і результату. Розрядність адреси співпадає з розрядністю регістра СА і визначається об'ємом ОП(а саме, кількістю можливих адрес). Кількість розрядів в регістрах ОП1, ОП2, РЕЗ і РК повинно співпадати з кількістю розрядів комірки ОП, оскільки в ОП1, ОП2 і РЕЗ повинні поміститися числа, а в РК - команда, які займають по одній комірці ОП.

3. Система команд УМ- 3.

Система команд ЕОМ - це набір усіх машинних команд цієї ЕОМ.

Назва	КОП	Операція	Примітка
зупинка	99	стоп	A1, A2, A3 - будь-хто
пересилка	00	[A1] → [A3]	A2 - будь-хто
арифметичні			
складання	01	[A1]+[A2] → [A3]	Встановлюються арифметичні прапори
віднімання	02	[A1]-[A2] → [A3]	
множення зі знаком	03	[A1]*[A2] → [A3]	
без знаку	13		
ділення зі знаком	04	[A1] div [A2] → [A3], [A1] mod [A2] → [A3+1]	
без знаку	14		
переходи			
безумовний	80	перейти до A3	A1, A2 - будь-хто
умовні			Після виконання команди переходу робота триває з команди, записаної в комірці з адресою A3
по =	81	при [A1]=[A2] перейти до A3	
по ≠	82	при [A1] ≠ [A2] перейти до A3	
по <	з/зн - 83 би/зн - 93		
по ≥	з/зн - 84 би/зн - 94		
по >	з/зн - 85 би/зн - 95		
по ≤	з/зн - 86 би/зн - 96		

У таблиці використано позначення [A] - вміст комірки з адресою A.

Арифметичні команди виконуються таким чином: вміст комірок з адресами A1 і A2 пересилається в регістри ОП1 і ОП2, виконується арифметична операція(результат - в регістрі РЕЗ), вміст регістра РЕЗ пересилається в осередок з

адресою АЗ. Слід звернути увагу на те, що є дві команди множення і дві команди ділення. Програміст використовує відповідну команду залежно від того, з якими даними(числами зі знаком або без знаку) повинна працювати програма. Команда ділення дає два результати - частку і залишок. Вони записуються в сусідні комірки: частка - за адресою АЗ, залишок - в осередок з адресою АЗ+1.

Дія команд переходу полягає в зміні утримуваного регістра СА. При цьому на наступному такті роботи ЦП виконуватиме команду, записану в комірки з адресою АЗ. (Згадайте (2).)

У командах умовного переходу робиться перевірка відповідної умови. Якщо умова виконується, тоді в СА записується адреса АЗ. Порівняння значень [А1] і [А2] виконується так: в ОП1 записується [А1], у ОП2 - [А2]; виконується віднімання ОП1-ОП2, при цьому набувають значення усі чотири арифметичні прапорці. По значеннях прапорців і визначається істинність умови. Деяким умовам(наприклад<) відповідають різні набори значень прапорців для різних даних - чисел зі знаком і чисел без знаку. Тому команди переходів для чисел зі знаком і чисел без знаку різні.

Розглянемо на прикладах відповідність між умовами і значеннями прапорців.

1) Перехід по рівності. В результаті віднімання [А1] - [А2] вийде ZF=1 тоді і тільки тоді, коли [А1] = [А2].

2) Перехід по "менше", числа без знаку.

[А1] < [А2] ([А1] - [А2] < 0 (CF=1.

3) Перехід по "менше", числа зі знаком. Якщо віднімаються числа зі знаком, то при [А1] < [А2], [А1] - [А2] < 0, можливі ситуації

а) результат вичислили правильно(OF=0) і він виявився негативним, значення SF=1

б) результат вичислили неправильно(OF=1), при цьому знак вичисленого значення суперечить знаку числа [А1] - [А2], вичислений результат виявився позитивним, тобто SF=0.

Таким чином, для чисел зі знаком

[А1] < [А2] ([А1] - [А2] < 0 ((OF=1) and (SF=0) or (OF=0) and (SF=1), або коротше OF (SF. При виконанні команди переходу по "менше" для чисел зі знаком процесор перевіряє здійсненість умови OF (SF.

4. Розберемо тепер декілька прикладів програм для УМ- 3. Умовимося, що перед початком роботи

1) у СА записується 0100, тобто виконання розпочинається з команди за адресою 0100,

2) у комірки ОП вже записані усі дані, необхідні для роботи програм(введення і виведення не розглядаємо).

Приклад 1. Вичислити значення $x = (a \bmod 50 - b) \cdot 2$ по заданим а і b.

Оскільки у вираженні є операція віднімання, краще працювати з числами зі знаком. Нехай дані розташовуються в наступних комірках

Адреса	Вміст
0000	<i>A</i>
0001	32_{16} ($=50_{10}$)
0002	<i>B</i>
0003	<i>X</i>
0004	робоча(знадобиться для ділення)

Програма.

Адреса	Вміст комірки	Коментар
0100	04 0000 0001 0003	$a \bmod 50 \rightarrow [0004]$
0101	02 0004 0002 0003	$[0004] - b \rightarrow x$
0102	03 0003 0003 0003	$x x \rightarrow x$
0103	99 0000 0000 0000	стоп

Нагадаємо, що ділення дає два результати, тому перша команда програми записує в $[0003]$ $x \text{ div } 50$ (це значення надалі не використовується) і в $[0004]$ $a \bmod 50$.

Приклад 2. Вичислити

$$S1 = \max(a, b) * 20$$

$$S2 = \min(a, b) \text{ div } 3.$$

Обчислення організуємо так:

begin if a < b then begin S1 := b; S2 := a end

else begin S1 := a; S2 := b end;

*S1 := S1 * 20;*

S2 := S2 div 3

end.

Дані:

0000 *a*

0001 *b*

0002 1416 (=2010)

0003 3

0004 *S1*

0005 *S2*

0006 робоча.

Вважатимемо, що працюємо з числами без знаку.

Програма:

Адр еса	Вміст комірки	Коментар	
0100	94 0000 0001 0104	<i>if a ≥ b, go to 0104</i>	
0101	00 0001 0000 0004	<i>S1 := b</i>	частина <i>then</i>
0102	00 0000 0000 0005	<i>S2 := a</i>	
0103	80 0000 0000 0106	<i>go to 0106</i>	
0104	00 0000 0000 0004	<i>S1 := a</i>	частина <i>else</i>
0105	00 0001 0000 0005	<i>S2 := b</i>	
0106	13 0004 0002 0004	<i>S1 := S1 * 20</i>	
0107	14 0005 0003 0005	<i>S2 := S2 div 3</i>	
0108	99 0000 0000 0000	СТОП	

Приклад 3. Вичислити $p = n!$.

Алгоритм:

```

begin p:=1; k:=2;
  while k ( n do
    begin p := p * k;
      k := k + 1
    end
  end.

```

У цьому завданні можуть виникнути досить великі позитивні числа, тому працюватимемо з числами без знаку.

Дані:

```

0000 n
0001 1
0002 2
0003 p
0004 k

```

Програма:

0100	00 0001 0000 0003	<i>p := 1</i>
0101	00 0002 0000 0004	<i>k := 2</i>
0102	95 0004 0000 0106	<i>if k > n, go to 0106</i>
0103	13 0003 0004 0003	<i>p := p * k</i>
0104	01 0004 0001 0004	<i>k := k + 1</i>
0105	80 0000 0000 0102	<i>go to 0102</i>
0106	99 0000 0000 0000	СТОП

Помітимо, що в машинній програмі початок циклу *while k (n do* змінився на перехід на кінець програми по запереченню умови. Тіло циклу закінчується безумовним переходом на початок циклу.

П.2. Учебна двоадресна машина УМ- 2.

Аналіз великого числа програм показує, що тільки 25% команд містять три різні адреси. У інших командах або два, або усі три адреси співпадають. Цей факт нашоєхує на ідею відмовитися від одного з адрес — зробити команди двоадресними. Тоді можна використати розряди, що звільнилися, для завдання довших адрес — з'являється можливість адресувати б(льший об'єм пам'яті. Або можна скоротити розмір комірки, зробивши дешевшою оперативну пам'ять.

Як позбавитися від третьої адреси? У УМ- 3 істотно трьохадресними були арифметичні команди і команди умовного переходу.

а) Арифметичні команди. Домовимося, що результат записується на місце першого операнда(за адресою А1).

б) Команди умовного переходу. Можна розбити команду дві, виділивши в окрему команду дію порівняння.

1. Опис УМ- 2.

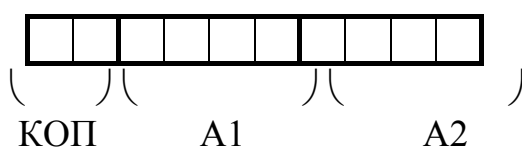
Структура процесора та ж, що і в УМ- 3; розрядність регістрів відповідає розрядності комірки і об'єму ОП.

Осередок ОП — 10 шістнадцяткових розрядів.

Об'єм ОП — 164 комірок.

Представлення чисел. Число займає один осередок. Негативні числа записуються в додатковому коді.

Формат команд. Команда займає один осередок:



Система команд. Коды операцій ті ж, що в УМ- 3. Відмінності полягають в наступному.

Команди	Дія
Пересилка 00 А1 А2	$[A1] := [A2]$
Арифметичні команди	$[A1] := [A1] \otimes [A2]$, где $\otimes \in \{+, -, *\}$
Ділення	обчислює частку і залишок від ділення $[A1]$ на $[A2]$; частка записується в $[A1]$, залишок в $[A1+1]$
Порівняння 05 А1 А2	обчислюється $[A1] - [A2]$, встановлюються арифметичні прапори; результат віднімання не зберігається
Умовні переходи	перехід по А2, якщо умова виконується; А1 не

Інші команди ті ж, що в УМ- 3.

2. Розглянемо два приклади програм для УМ- 2.

Приклад 1. Вичислити значення $x = (a \bmod 50 - b)^2$ по заданим a і b .

Розподіл пам'яті :

```
0000  a
0001  x
0002  3216 (=5010)
0003  b
```

Програма:

0100	04 0000 0002	$x := a \bmod 50$
0101	02 0001 0003	$x := x - b$
0102	03 0001 0001	$x := x * x$
0103	99 0000 0000	стоп

Приклад 2. Вичислити $p = n!$.

Алгоритм:

```
begin p:=1; k:=2;
  while k ( n do
    begin p := p * k;
      k := k + 1
    end
  end.
```

Розподіл пам'яті :

```
0000  1
0001  2
0002  n
0003  p
0004  k.
```

Програма:

0100	00 0003 0000	$p := 1$
0101	00 0004 0001	$k := 2$
0102	05 0004 0002	$k (n ?$
0103	95 0000 0107	$k > n, go to 0107$
0104	13 0003 0004	$p := p * k$
0105	01 0004 0000	$k := k + 1$
0106	80 0000 0102	$go to 0102$
0107	99 0000 0000	стоп

Завдання.

Парні варіанти виконують завдання на основі команд УМ-3, а непарні на основі команд УМ-2.

- 1) $y = (a + b) * 5$
- 2) $y = (x - z) / 2$
- 3) $y = (a + b) \bmod 4$
- 4) $y = \frac{z}{a} + 5$
- 5) $y = (c - d) * 10$
- 6) $y = (x + b) * 3$
- 7) $y = (z - x) / 8$
- 8) $y = \frac{a}{b} - 9$
- 9) $y = (b - x) \bmod 4$
- 10) $y = (a + b) \bmod 2$
- 11) $y = (x * z) \bmod 5$
- 12) $y = \frac{x}{z} + 12$
- 13) $y = a * b - x$
- 14) $y = (c * d) \bmod a$
- 15) $y = (c * x - a) / 8$
- 16) $y = (x + c) * d$
- 17) $y = (x - d) / 3$
- 18) $y = \left(a + \frac{b}{x} - c\right) \bmod 2$
- 19) $y = \left(\frac{d}{z} - x + b\right) * 4$
- 20) $y = \left(\frac{x}{z} + f\right) * 10$

Контрольні запитання.

1. Назвіть основні властивості ЕОМ.
2. Назвіть принципи Фон-Неймана.
3. Намалюйте структуру ЕОМ.
4. Для чого призначений процесор?
5. Для чого призначена оперативна пам'ять?
6. Для чого призначені зовнішні пристрої?
7. Які пристрої включає в себе процесор?
8. Які машини називаються учбовими?
9. Назвіть види учбових машин.
10. Структура процесора учбової трьох адресної машини.
11. Структура процесора учбової двоадресної машини.

Лабораторна робота 7. Структура процесора 80x86

Мета роботи . Ознайомитись зі структурною організацією одного з перших універсальних процесорів Intel 80x86 – базовою моделлю процесорів для персональних комп'ютерів IBM PC.

Короткі теоретичні відомості.

Intel 8086 (також відомий як iAPX86) — перший 16-бітний мікропроцесор компанії Intel, що розроблявся з весни 1976 року і випущений 8 червня 1978. Процесор мав набір команд, який застосовується і в сучасних процесорах, саме від нього бере свій початок відома на сьогодні архітектура x86.

Основними конкурентами мікропроцесора i8086 були Motorola 68000, Zilog Z80, чипсети F-11 і J-11 сімейства PDP-11, MOS Technology 65C816. Деякою мірою, в області військових розробок, конкурентами були процесори-реалізації MIL-STD-1750A.

Аналогом мікропроцесора i8086 був процесор NEC V30 (на 5% продуктивніший за i8086 і при цьому повністю з ним сумісний). Радянським клоном був мікропроцесор K1810BM86, що входив в серію мікросхем K1810.

У 1972 році Інтел випустила 8008, перший 8-бітний мікропроцесор. Він використовував набір інструкцій, розроблений корпорацією Datapoint для програмованих комп'ютерних терміналів, придатний і для універсальних процесорів. Цей процесор вимагав декількох додаткових мікросхем для використання в повноцінному комп'ютері, тому що використовував маленький 18-піновий корпус від мікросхем DRAM, вироблених Інтел, і відповідно не міг мати окрему шину адрес.

Двома роками пізніше, в 1974, був запущений 8080, у новому, 40-піновому DIP-корпусі, спочатку розробленому для мікросхем калькуляторів. Він мав окрему шину адрес і розширений набір інструкцій, кодово- (не бінарно-) сумісний з 8008, доповнений для зручності програмування декількома 16-бітними інструкціями. Процесор i8080 часто називають першим по-справжньому зручним і корисним мікропроцесором. У 1977 році він був замінений на i8085, з однією напругою живлення (+5 В) замість трьох різних на попереднику і декількома іншими удосконаленнями. Найвідомішими суперниками були 8-бітні Motorola 6800 (1974), Microchip PIC16X (1975), MOS Technology 6502 (1975), Zilog Z80 (1976), і Motorola 6809 (1978).

Процесор i8086 являє собою модернізований процесор i8080 і, хоча, розробники не ставили перед собою мету досягти повної сумісності на програмному рівні, більшість програм написаних для i8080 здатні виконуватися і на i8086 після перекомпіляції. Новий процесор несе у собі безліч змін, які дозволили значно (в 10 разів) збільшити продуктивність у порівнянні з попереднім поколінням процесорів компанії.

Всього в процесорі i8086 було 14 16-розрядних регістрів: 4 регістри загального призначення (AX, BX, CX, DX), 2 індексні регістри (SI, DI), 2 вказівні (BP, SP), 4 сегментні регістри (CS, SS, DS, ES), програмний лічильник або показник команди

(IP) і реєстр прапорців (FLAGS, включає 9 прапорів). При цьому реєстри даних (AX, BX, CX, DX) допускали адресацію не лише цілих реєстрів, але і їх молодшої половини (реєстри AL, BL, CL, DL) і старшої половини (реєстри AH, BH, CH, DH), що дозволяло використовувати не лише нове 16-розрядне ПЗ, але зберігало сумісність і зі старими програмами (правда, їх необхідно було, принаймні, перекомпілювати).

Розмір шини адреси був збільшений з 16 біт до 20 біт, що дозволило адресувати 1 Мбайт (220 байт) пам'яті. Шина даних була 16-розрядною. Проте в мікропроцесорі шина даних і шина адреси використовували одні й ті ж контакти на корпусі. Це призвело до того, що не можна одночасно подавати на системну шину адреси і дані. Мультиплексування адрес і даних в часі скорочує число контактів корпусу до 20, але й уповільнює швидкість передачі даних.

Для того щоб адресувати більший, ніж 18080, обсяг пам'яті, треба було змінити спосіб адресації пам'яті. Адже якщо використовувати старі методи, коли адресу до комірки пам'яті містився у вказівних реєстрах, то довелося б збільшувати розмір тих самих реєстрів, щоб мати можливість звертатися до більшого обсягу пам'яті. Тому для адресації 1 Мбайт пам'яті застосовували наступну схему. На шину адреси подавалася фізична адреса розміром 20 біт, яка формувалася шляхом складання вмісту одного із сегментних реєстрів (16 біт), помноженого на 24, з вмістом вказівного реєстра: таким чином, адресація комірки пам'яті вироблялася за номером сегмента і ефективною адресою комірки в сегменті (яка також називається зсувом). Якщо результат додавання виявлявся більше ніж 220 -1, то 21-ий біт відкидався; така процедура називається «загортанням» адреси (англ. address wraparound). Цей метод згодом (після появи захищеного режиму) назвали реальним режимом адресації процесора, такий режим дозволяє адресувати до 1 Мбайт пам'яті.

Для того, щоб адресувати 1 мегабайт пам'яті (20 біт адреси) з використанням 16-бітових реєстрів використовується сегментування. Старші 4 біт адреси виводяться на окремі контакти корпусу, а молодші 16 виводяться на поєднану шину адреси-даних. Але межа сегменту не жорстка, а плаваюча. Для того, щоб адресувати потрібний сегмент використовуються 16-бітові реєстри сегменту, значення яких зсувається на 4 біта вгору і складається з вказівним 16-бітовим реєстром. Отримане значення — 20 бітова адреса пам'яті або пристрою виводиться на контакти. Якщо результат складання виявляється більше ніж 1 мегабайт, виводяться тільки молодші 20 біт адреси, 21-біт відкидається.

Таким чином, пам'ять розділяється на сегменти, розміром 64 Кбайт кожен і починаються з адреси, кратної 16 (межа параграфа); пам'ять в 1 Мбайт розділялася, таким чином, на 16 сегментів. Ці 16 сегментів називають сторінками пам'яті. У комп'ютері, подібному IBM PC, останні 6 сторінок (A, B, C, D, E, F) пам'яті (т. зв. верхня пам'ять — англ. upper memory) використовувалися для відеопам'яті і BIOS-а, це обмежувало пам'ять, доступну користувачеві, об'ємом в 640 Кбайт (т. зв. звичайна пам'ять — англ. conventional memory; сторінки 0~9).

На той час такий режим адресації забезпечував безліч переваг: ємність пам'яті могла складати до 1 Мбайт, хоча команди оперували 16-бітовими адресами; спрощувалося використання окремих областей пам'яті для програми, її даних і стоку; спрощувалася розробка пристроїв, сумісних один з одним.

Система команд процесора i8086 складається з 98 команд (і більше 3800 їх варіацій): 19 команд передачі даних, 38 команд їх обробки, 24 команди переходу і 17 команд управління процесором. Можливі 7 режимів адресації. Мікропроцесор не містив команди для роботи з числами з плаваючою комою. Ця можливість реалізовувалася окремою мікросхемою, що називається математичний співпроцесор, який встановлювався на материнській платі. Співпроцесор зовсім не обов'язково мав бути зроблений Intel (модель i8087), наприклад, деякі виробники мікросхем, такі як Weitek, випускали продуктивніші співпроцесори, ніж Intel.

Система команд процесора i8086 включає в себе декілька дуже потужних рядкових інструкцій. Якщо інструкція має префікс REP (повтор), то процесор виконуватиме операції з блоками — переміщення блоку даних, порівняння блоків даних, присвоєння певного значення блоку даних певної величини, і так далі, тобто, одна інструкція 8086 з префіксом REP може виконувати 4-5 інструкцій, що виконуються на деяких інших процесорах. Але слід згадати, що подібні прийоми були реалізовані і в інших процесорах — Zilog Z80 мав інструкції переміщення і пошуку блоків, а Motorola 68000 може виконувати операції з блоками, використовуючи всього дві команди.

У мікропроцесорі i8086 була використана примітивна форма конвеєрної обробки. Блок інтерфейсу з шиною подавав потік команд до виконавчого пристрою через 6-байтову чергу команд. Таким чином, вибірка та виконання нових команд могли відбуватися одночасно. Це значно збільшувало пропускну спроможність процесора і позбавляло необхідності чекати зчитування команди з пам'яті при зайнятості іншими операціями інтерфейсі мікросхеми (у той час швидкість пам'яті значно випереджала швидкість цього процесора).

У персональних комп'ютерах процесор i8086 практично не використовувався через дорожнечу спеціалізованих мікросхем, які були потрібні для роботи процесора. Це зрозуміли і в Intel, в 1979 році вона випускає процесор i8088, у якого шина даних була 8-бітовою.

Через брак (ще не були розроблені) допоміжних 16-бітових мікросхем, і можливості використання великого парку 8-бітових, а також для здешевлення і зменшення розмірів плат, було вирішено випустити 8-бітовий варіант процесора (i8088). У 70-і роки мікросхеми динамічної оперативної пам'яті мали 1-бітову організацію і для 8 бітової системи було потрібно 8, а для 16-бітової — 16 мікросхем пам'яті. Тому випуск 8 розрядної версії здешевлював виробництво і зменшував розмір друкованої плати комп'ютера.

Але все ж в деяких мікрокомп'ютерах застосовувався і i8086, одним з таких є Mусron 2000 — перший комерційний мікрокомп'ютер на базі i8086. Машина для обробки текстів IBM Displaywriter, Compaq DeskPro і Wang Professional Computer також використовували i8086.

Технічні характеристики

Тактова частота (МГц): від 4 до 10

5 (модель 8086), при частоті 4,77 продуктивність — 0,33 MIPS

8 (модель 8086-2, 0,66 MIPS)

10 (модель 8086-1, 0,75 MIPS)

Приблизні витрати часу на операції, процесорних циклів (ЕА — час, необхідний для розрахунку ефективного адреси пам'яті, яке варіюється від 5 до 12 циклів):

+ Підсумовування: 3-4 (реєстрові), 9 + ЕА — 25 + ЕА — при операціях з пам'яттю + Множення: 70-118 (реєстрові), 76 + ЕА — 143 + ЕА — при операціях з пам'яттю + Переміщення даних: 2 (між регістрами), 8 + ЕА — 14 + ЕА — при операціях з пам'яттю

Розрядність регістрів: 16 біт

Розрядність шини даних: 16 біт

Розрядність шини адреси: 20 біт

Обсяг пам'яті, що адресується: 1 Мбайт

Адресний простір I / O: 64 Кбайт

Кількість транзисторів: 29 000

Техпроцес (нм): 3000 (3 мкм)

Площа кристала (кв. мм): ~ 30 (за іншими даними, 16 мм²)

Максимальна тепловиділення: 1,75 Вт

Напруга живлення: +5 В

Роз'єм: немає (мікросхема припаюється до плати)

Корпус: 40-контактний керамічний чи пластиковий DIP, пізніше — 56-контактний QFP і 44-контактний PLCC

Підтримувані технології: 98 інструкцій

Обсяг черги команд: 6 байт

Особливості. Формування фізичної адреси. Організація зовнішньої пам'яті.

Особливості МП : - розширена система адресації - 24 способи ; - наявність команд множення , ділення і послідовності байтів і слів; - наявність шести регістрів черзі - прообраз КЕШ -пам'яті. Регістри черги заповнюються наступною командою . Заповнення регістрів черзі відбувається одночасно з виконанням попередньої команди ; - розвинена система переривань , використовується 256 запитів на переривання ; - є апаратні засоби для реалізації багатопроцесорної системи . По організації пам'яті МП є машиною фон - неймановскую типу , так як пам'ять даних і пам'ять команд знаходяться в єдиному адресному просторі . Крім того , МП відноситься до НД з програмним управлінням , при якому необхідний дешифратор команд.

Фізична адреса

МП i8086 має 20- розрядну ША , яка забезпечує адресний простір в 1 Мбайт. Адреса на ША МП називається фізичною адресою . У МП широко використовуються різні способи непрямой адресації , коли джерелом адреси служать індексні і базові 16 - розрядні регістри . У цьому випадку говорять про виконавче (або логічному) адресі - ЕА. Виконавчий адресу можна визначити як адресу в межах сторінки пам'яті , номер якої визначений сегментним регістром . Яким же чином формується 20 - розрядний фізичну адресу з 16 - розрядного виконавчого адреси ?

На рисунку 1 представлена схема формування фізичної адреси, прийнята для МП i8086.

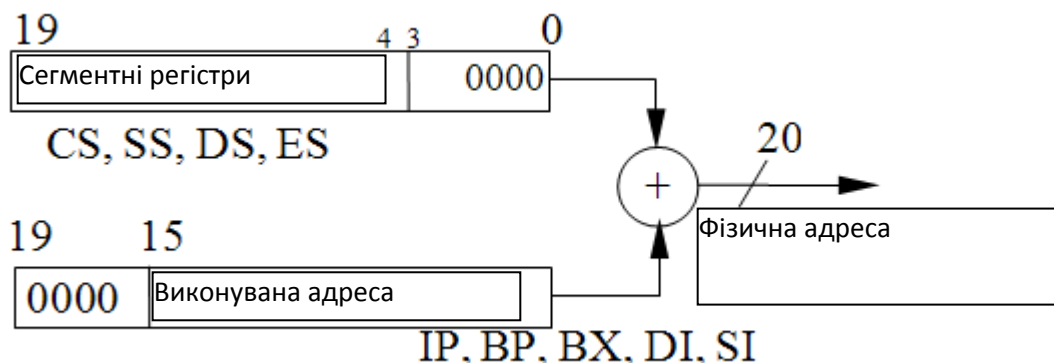


Рисунок 1 – Формування фізичної адреси

Для формування фізичної адреси виконується підсумовування виконавчої адреси з вмістом сегментного реєстра, зрушеного на 4 розряди вліво. Виконавчий адресу формується за допомогою комбінації вмісту декількох (до двох) реєстрів і за допомогою прямого адресного зміщення, зазначеного в команді. Наприклад, виконавчий адресу в команді MOV AX, [BX + SI + 159] формується з двох реєстрів і додаткового прямого зміщення (рисунок .2).

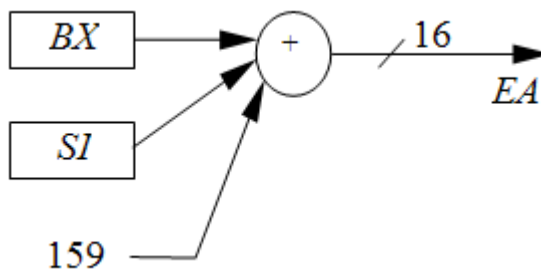


Рисунок 2 - Формування виконуваної адреси

При проектуванні МК необхідно врахувати, що пуск МП i8086 після зняття сигналу скидання SR виконується з фізичної адреси FFFF0h. Ця адреса виходить в результаті підсумовування за схемою, представленої на малюнку 3.4.1, і з урахуванням того, що в сегментний реєстр завантажується код FFFFh, а реєстр адреси в межах поточного сегмента IP формує код 0000h.

Організація зовнішньої пам'яті

Пам'ять в МП, що розробляються фірмою Intel, організована побайтно. Навіть в МП останнього покоління типу Pentium при 64-розрядній ШД пам'ять все одно організована побайтно. Для реалізації можливостей, які відкривають ШД більше одного байта у всіх МП цієї архітектури передбачено читання кодів з пам'яті словами. Нагадаємо, що слово - це основний формат шини даних розглянутої обчислювальної системи. Для МП i8086 слово - це 16 розрядів, тобто два байти.

Тому, поряд з режимами побайтного звернення до зовнішньої пам'яті в цьому МП передбачений режим читання-запису слова

ВНЕ	ША0	Пояснення
0	0	Чт/Зп 2х байтів
0	1	Чт/Зп ст. байтів
1	0	Чт/Зп мл. байтів
1	1	Немає звернення

Режим читання-запису слова

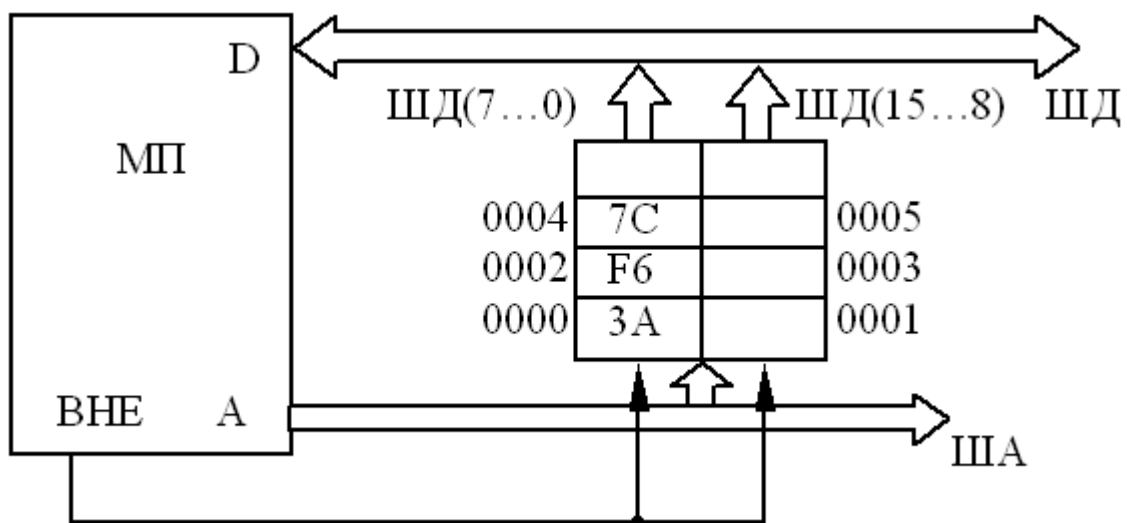


Рисунок 3 – Організація зовнішньої пам'яті

Для можливості звернення за словами і по байтам пам'ять розділена на 2 банку з парними і непарними адресами, причому осередки з парними адресами підключаються до молодшого байту ШД, а з непарними - до старшого. Для вибору відповідного банку пам'яті використовується сигнал ВНЕ спільно з розрядом ША0. Таким чином при проектуванні МК необхідно два ПЗУ і два ОЗУ. Сигнал вибору банку пам'яті повинен враховувати таблицю наведену вище і загальний сигнал дозволу вибору пам'яті - ПЗУ або ОЗУ.

Питання до виконання практичної роботи

1. Місце процесора в комп'ютері та його функції.
2. Що таке командний цикл?
3. Дві основні фази командного циклу.
4. Основні вузли процесора.
5. Одношинна структура процесора комп'ютера із складною системою команд і

його зв'язки з іншими пристроями комп'ютера.

6. Виконання процесором операції "Вибірка слова з пам'яті".

7. Виконання процесором операції "Запам'ятовування слова в пам'яті".

8. Виконання процесором операції обміну між регістрами.

9. Виконання процесором арифметичних і логічних операцій.

10. Порівняння одношинної та багатошинної структур процесора комп'ютера із складною системою команд.

11. Чому в процесорі комп'ютера із складною системою команд команда виконується за багато тактів?

12. Чому в процесорі комп'ютера із складною системою команд потрібна складна система розпізнавання команди?

13. Основні вимоги до процесора комп'ютера з простою системою команд.

14. Сформуйте правила вибору системи команд комп'ютера з простою системою команд.

15. Чому в системі команд комп'ютера з простою системою команд відносно небагато операцій та способів адресації?

16. Чому в комп'ютері з простою системою команд команди обробки даних мають реалізуватися лише у формі "регістр-регістр"?

Лабораторна робота 8. Функціональна схема та сигнали мікропроцесора 8086

Мета роботи . Ознайомитись з апаратною будовою та сигналами процесора Intel 80x86 і принципами його використання при створенні комп'ютерних схем.

Короткі теоретичні відомості.

Мікросхема 8086 являє собою однокристальний високопродуктивний 16 - розрядний мікропроцесор з фіксованою системою команд. Мікропроцесор призначений для використання в якості центрального процесорного пристрою при побудові засобів обчислювальної техніки - від найпростіших одноплатних мікроЕОМ до високопродуктивних мультипроцесорних систем .

Мікропроцесор володіє високою швидкістю , забезпечує можливість прямої адресації пам'яті об'ємом до 1М байта , 65536 пристроїв введення та 65536 пристроїв виводу. Для обчислення адрес операндів , розміщених в пам'яті , використовується 24 режиму адресації . Мікропроцесор має векторну структуру переривань і забезпечує обробку до 256 запитів переривань трьох типів: зовнішніх , внутрішніх і програмних .

Архітектурною особливістю мікропроцесора 8086 є наявність апаратно - програмних засобів, що дозволяють спростити побудову мультипроцесорних систем на його основі. Ці кошти забезпечують синхронізацію роботи кількох незалежних (що виконують власні потоки команд) процесорів , що мають спільні ресурси , а також синхронізацію паралельної роботи мікропроцесора і сопроцесорів (спеціалізованих процесорів , апаратно реалізують команди складних процедур). Мікропроцесор 8086 характеризується двома режимами роботи - мінімальним і максимальним , які відрізняються способом формування сигналів обміну і відповідно можливостями реалізованих систем .

Функціональна схема мікропроцесора наведена на рисунку 1. Структура мікропроцесора 8086 орієнтована на паралельне виконання функцій вибірки і виконання команд і складається з пристрою сполучення каналу (ПСК) , пристрої обробки (ПО) і пристрої керування і синхронізації .

Пристрій сполучення каналу призначено для формуючої фізичної адреси пам'яті , вибірки команд з пам'яті і запису їх в чергу команд , читання операндів команд з пам'яті або регістрів введення / виводу , записи результатів виконання команд в пам'ять або регістри введення / виводу.

У ПСК входять: шість 8 - розрядних регістрів черги команд ; чотири 16 - розрядних сегментних регістра ; 16 - розрядний регістр адреси (покажчика) команди ; 16 - розрядний регістр обміну ; 16 - розрядний суматор адреси.

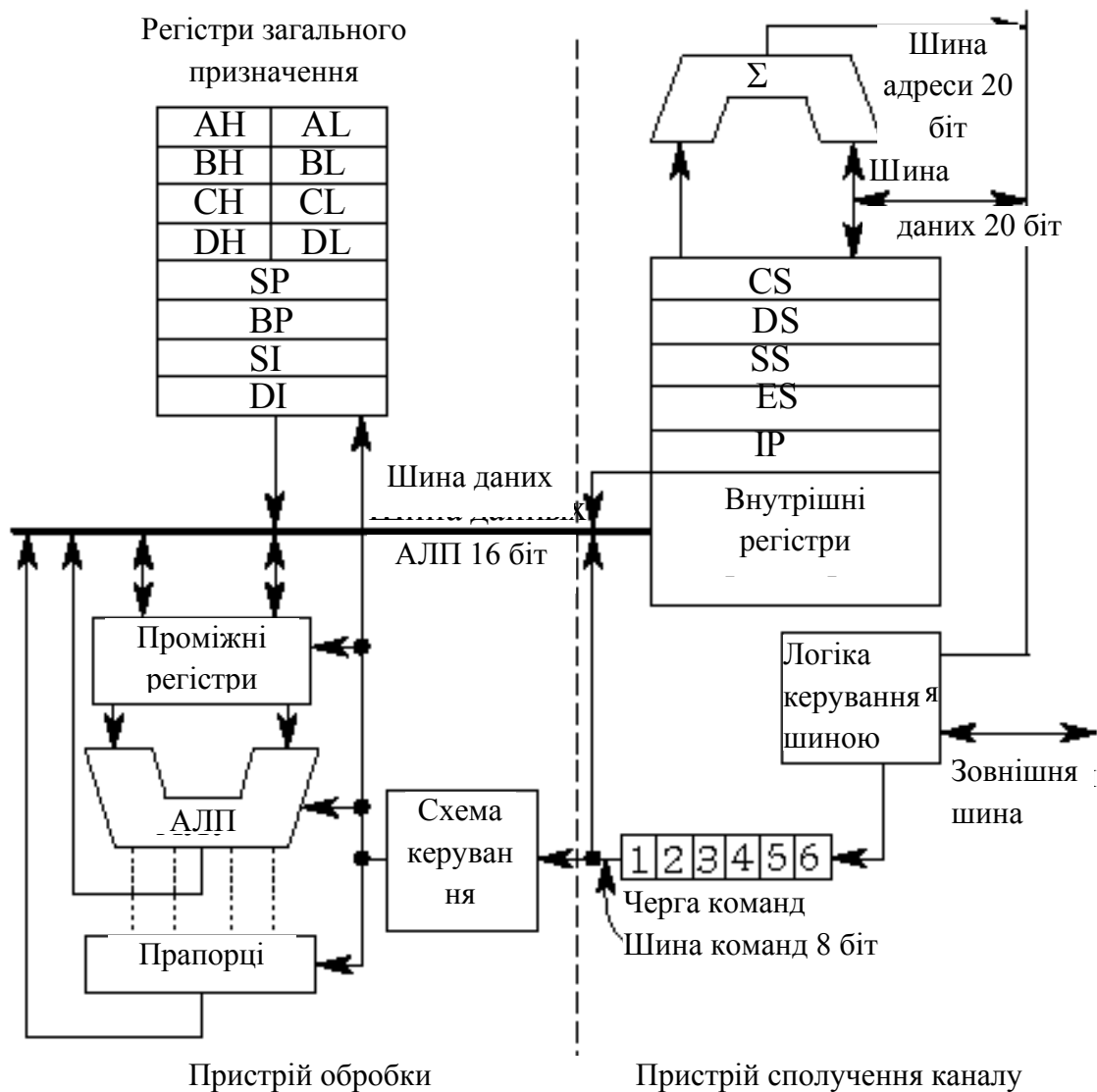


Рисунок 1 . Функціональна схема мікропроцесора

Пристрій обробки призначене для виконання операцій з обробки даних . Команди , вибрані з пам'яті і записані в реєстри черги команд ПСК , за запитами від ПО надходять через 8 - розрядну магістраль команд на мікропрограмне пристрій управління , яке декодує команди і виробляє відповідну послідовність мікрокоманд , керуючу процесом виконання поточної операції . ПО не має непосредственої зв'язки з зовнішнім магістраллю системи і обмінюється даними через реєстр обміну з УСК .

У пристрій обробки входять: 16 - розрядне арифметико - логічний пристрій , вісім 16 - розрядних реєстрів загального призначення , 16 - розрядний реєстр ознак стану Мікропроцесора .

Команди завжди вибираються з пам'яті як слова , незалежно від парності або непарності адреси, за якою проводиться читання команди.

Відмінною особливістю 8086 є можливість апаратної перебудови внутрішньої структури схеми керування і синхронізації. Вибір режиму функціонування цієї

схеми надає розробнику системи можливість вибору підмножини вихідних керуючих сигналів у відповідності зі ступенем складності проектованої мікропроцесорної системи . Системна настройка забезпечується спеціальним висновком вибору режиму MN / MX .

Мікропроцесор дозволяє обробляти 256 типів переривань з номерами від 0 до 255 , які діляться на зовнішні апаратні , внутрішні апаратні і програмні . Запити на зовнішні переривання формуються зовнішніми стосовно мікропроцесору пристроями. Запити на внутрішні переривання формуються при виконанні певних команд або за деякими умовами при виконанні команд. По любому перериванню управління передається програмі (процедурою) обслуговування переривання допомогою вектора переривання , обираного з таблиці векторів переривання , що розташовується в пам'яті.

Запити на зовнішні переривання сприймаються і обробляються після виконання поточної команди . Зовнішні переривання надходять на мікропроцесор за двома зовнішніх висновків (INT і NMI) і діляться на маскуючі і немасковані .

Питання, пов'язані з використанням режиму переривань обговорюються більш докладно далі в шостому розділі , яка присвячена принципам організації обміну даними між мікропроцесором і зовнішніми пристроями.

Інтерфейсні сигнали мікропроцесора , цикли обміну з шиною

Мікропроцесор взаємодіє з системою за допомогою зовнішніх інтерфейсних сигналів. Ці сигнали можуть бути розділені на три групи : адреса / дані , управління та службові . Призначення некого з висновків мікропроцесора залежить від його режиму роботи. Режим роботи визначається спеціальним сигналом MN / MX . Мінімальний режим включається при подачі на вхід MN / MX логічної 1 , максимальний - при подачі на вхід MN / MX логічного 0 . Графічне представлення мікропроцесора показано на рис.2.2 . На цьому малюнку заперечення назви сигналу увазі , що активний рівень сигналу низький . Позначення виводів мікропроцесора в мінімальному режимі , якщо функції висновків у мінімальному режимі відрізняються від їх функцій в максимальному.

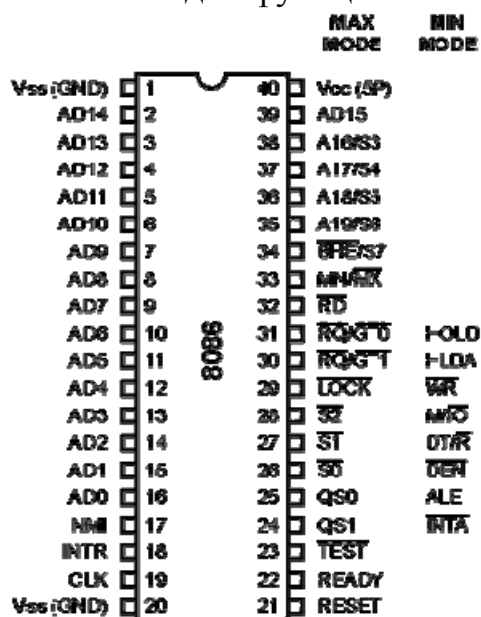


Рис . 2.2 . Виводи мікропроцесора 8086

Опис основних сигналів мікропроцесора.

Спочатку ми розглянемо сигнали адреси / даних. На рис. 2 ці сигнали мають мнемоніку AD або A. Мікропроцесор 8086 має поєднану шину адреси і даних, зазвичай звану мультиплексованою шиною адреси / даних. Тимчасове мультиплексування даних і адрес дозволяє скоротити необхідну кількість висновків корпусу мікропроцесора. Шина може бути розділена за допомогою зовнішніх регістрів - клямок на окремі шини адреси і даних. Висновки шини адреси (A19 ... A16) вказують чотири старших адресних біта. Активний рівень цих сигналів - високий. Сигнали шини адреси / даних (AD0 ... AD15) утворюють мультиплексовану шину адреси даних мікропроцесора.

Сигнал BHE (дозвіл старшого байта шини) використовується, щоб дозволити передачу даних по старшій половині шини даних (D8 ... D15). Сигнал BHE повинен мати низький рівень для дозволу передачі старшого байта даних. Сигнал BHE не потребує проміжного запам'ятовування в регістрі - засувці. Передача слова або байта по шині даних визначається значеннями сигналів A0 і BHE.

Сигнали управління відрізняються в мінімальному і максимальному режимі. У мінімальному режимі мікропроцесор виробляє всі необхідні системні сигнали управління. У максимальному режимі необхідний зовнішній системний контролер. Мікропроцесор виробляє всі необхідні керуючі сигнали, які надходять на цей контролер, а він виробляє всі необхідні керуючі сигнали.

Далі описані сигнали для мікропроцесора в мінімальному режимі.

Сигнал ALE (дозвіл фіксації адреси) забезпечується процесором, щоб зафіксувати адресу на мультиплексованій шині адреси / даних. Активний рівень сигналу ALE - високий, правильне значення адреси гарантується по зрізу даного сигналу.

Строб запису (WR) вказує, що дані із шини даних повинні бути написані в пам'ять або пристрій введення / виводу. Цей сигнал має активний низький рівень.

Строб читання (RD) - сигнал з активним низьким рівнем, який вказує, що процесор виконує цикл читання пам'яті або пристрою введення / виводу.

Передача / прийом даних (DT / R) управляє напрямком потоку даних через зовнішній приймач шини даних. Коли сигнал має низький рівень, дані передаються до процесора. При високому рівні процесор передає дані на шину даних.

Дозвіл даних (DEN) активізує приймачі шини даних. DEN активний завжди, коли відбувається передача даних. Активний рівень - низький. DEN не активний всякий раз, коли DT / R змінює стан.

Пам'ять / ПБВ (M / IO) - визначає, куди віддаються дані. Коли рівень сигналу низький, дані передаються до пристрою введення / виводу. Коли високий - дані передаються в пам'ять.

Запит на переривання (INTR) сигнал вимоги переривання поза шнім пристроєм. Це вхідний сигнал, його активний рівень - високий. У відповідь мікропроцесор забезпечує сигнал підтвердження переривання (INTA) з активним низьким рівнем. Це сигнал маскуемого переривання.

Немасковане переривання (NMI) викликає переривання з вектором 2 . Немасковане переривання не можна заборонити програмно .

HOLD (активний при високому рівні сигналу) вказує , що інший пристрій управління передачею даних по шині вимагає зовнішню шину. Процесор генерує сигнал HLDA у відповідь на запит. Одночасно з формуванням сигналу HLDA , процесор переводить шини в високоімпедансний стан . Після зняття сигналу HOLD процесор переводить сигнал HLDA в неактивний стан . Коли процесор повинен виконати інший цикл шини , він буде знову управляти локальною шиною і шинами управління .

У максимальному режимі сигнал LOCK вказує , що інші пристрої керування передачею даних по шині системи не можуть отримати прав керування . Сигнал LOCK має активний низький рівень. Сигнал LOCK формується за спеціальною командою префікса блокування (LOCK) і активізується на початку першого циклу передачі даних , пов'язаного з командою наступній безпосередньо після префікса блокування і залишається активним до завершення цієї команди. Якщо сигнал LOCK активний, вибірки команд з пам'яті в чергу команд не відбувається.

Висновки QS0 і QS1 несуть інформацію про стан черги команд процесора. Табл.1 показує можливі стани черги команд.

Сигнали стану циклу шини (S0 ... S2) кодують тип машинного циклу , що виконується мікропроцесором , як показано в табл.2 .

У максимальному режимі HOLD - HLDA протокол трансформірується в протокол управління доступом до шини Запит / Дозвіл (RQ / GT) . Це дозволяє іншим сопроцесорам включатися в загальну систему з мікропроцесором 8086.

Таблиця 1 Операції з чергою команд

S1	Q	S0	Q	Операції з чергою команд
	0		0	Немає операцій
	0		1	Перший байт коду операції вибирається з черги
	1		1	Вибірка наступного байта з черги
	1		0	Черга команд порожня

Таблиця 2 Типи машинних циклів

2	S	1	S	0	S	Тип цикла
	0		0		0	Підтвердження переривання
	0		0		1	Читання ПБВ
	0		1		0	Запис ПБВ
	0		1		1	Зупинка
	1		0		0	Вибірка команди
	1		0		1	Читання даних з пам'яті
	1		1		0	Запис даних у пам'ять
	1		1		1	Пасивний (немає операцій)

CLK , RESET , READY - сервісні сигнали. Активний рівень сигналу RESET змушує процесор негайно закінчити поточний дію , очистити внутрішню логіку, і перейти в

неактивний стан . Процесор починає вибирати команди через кілька циклів тактових сигналів після того , як RESET повертається в неактивний стан . Вхідний сигнал CLK повинен бути підключений до генератора синхронізації. Сигнал READY повідомляє процесору , що пам'ять або пристрій вводу / виводу закінчило передачу або прийом даних. З'єднання READY з рівнем логічної 1 буде завжди встановлювати стан готовності для процесора.

Для користувача дії, що виконуються мікропроцесором , представляють собою послідовність циклів каналу з обміну інформацією з пам'яттю або периферійними пристроями. Кожен цикл каналу мікропроцесора складається , як мінімум , з чотирьох машинних тактів T1 - T4. Машинний такт починається по спаду імпульсу синхронізації CLK і триває один період синхронізації.

Будь-який цикл шини можна умовно розділити на дві фази:

фаза передачі адреси / статусу;

фаза передачі даних.

Фаза передачі адреси починається перед початком такту T1 і триває протягом цього такту . Фаза передачі даних починається в такті T2 і закінчується в такті T4. У такті T1 на канал адреси / даних завжди видається адресна інформація . У цьому ж такті виробляється сигнал ALE , який дозволяє ідентифікувати початок циклу каналу і використовується як стробируючий імпульс для занесення адресної інформації в зовнішній регістр адреси.

У такті T2 проводиться перемикання напрямки роботи ка \rightarrow на \leftarrow ла адреси / даних.

Передача даних по каналу відбувається в тактах T3 і T4. Тривалість циклу каналу може бути подовжена використанням керуючого сигналу READY . Цей сигнал дозволяє розробнику синхронізувати швидкість роботи зовнішньої пам'яті зі швидкістю роботи мікропроцесора введенням в цикл каналу між тактами T3 і T4 додаткових тактів очікування. Протягом тактів очікування дані на каналі залишаються незмінними. Між тактом T4 поточного циклу і тактом T1 наступного циклу каналу процесор може вводити додаткові (холості) такти , призначені для виконання внутрішніх дій . Моменти введення цих тактів і їх число залежать від стану черги команд і виконуваної команди в УО .

На рис.3 представлена типова часова діаграма виконання циклів читання і запису.

Цикл читання починається з вироблення сигналу ALE . Цей сигнал використовується для занесення адресної інформації в зовнішній регістр адреси. У такті T2 канал A / D перемикається в високоомне стан , виробляється сигнал RD , який використовується для читання адресуємо \rightarrow мого пристрою. Для управління шинними формувачами , що забезпечують розв'язку каналу адреси / даних мікропроцесора від системного каналу даних , використовуються сигнали DT / R і DEN .

Цикл запису (як і цикл читання) починається з видачі сигналу ALE і адреси на шину адреси / даних. У такті T2 безпосередньо за видачею адреси на шину A / D видаються дані для запису в пристрій, що адресується . Ця інформація залишається істинною на каналі даних до закінчення такту T4. Сигнал WR виробляється на початку такту T2 і залишається в цьому стані до початку такту T4.

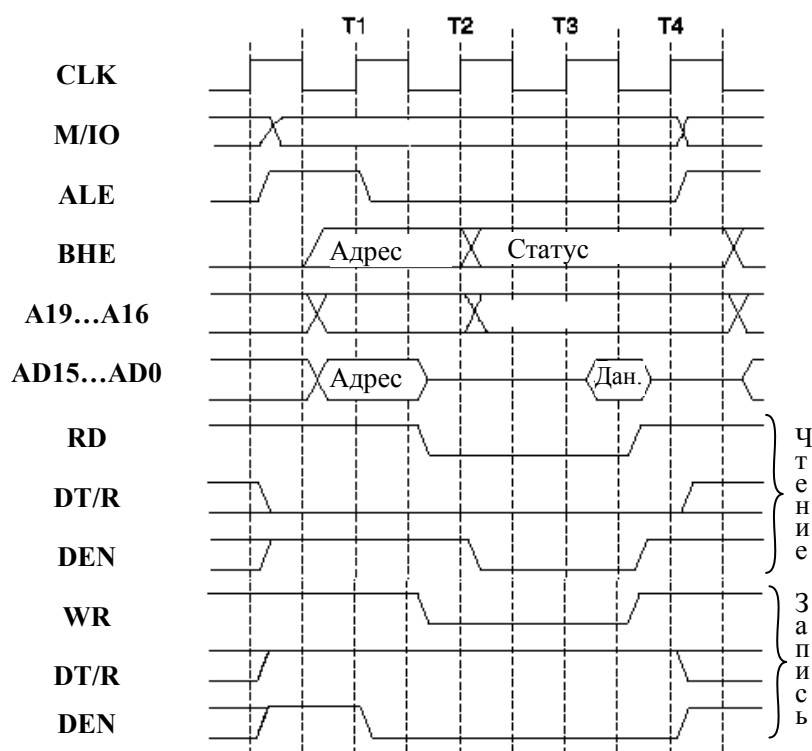


Рисунок 3 - Часова діаграма

Питання до виконання практичної роботи

1. Як організовується зв'язок між процесором і основною пам'яттю?
2. Який порядок виконання команд в комп'ютері?
3. Як кодуються команди в комп'ютері?
4. Що таке асемблерна мова і для чого використовується асемблер?
5. Як класифікуються команди за типами операцій?
6. Назвіть команди обробки даних
7. Назвіть базові операції зсуву
8. Назвіть команди переміщення даних
9. Поясніть принципи організації послідовного виконання команд і розгалуження
10. Назвіть команди передачі керування
11. Назвіть команди переходу
12. Назвіть команди пропуску
13. Назвіть команди звертання до підпрограм
14. Поясніть принципи конвеєрного виконання команд

15. Якою є продуктивність 4-ярусного конвеєра з тактом 20 нс при виконанні 100 команд?
16. Назвіть можливі конфлікти, які можуть сповільнити конвеєр
17. Які використовуються формати команд при роботі з основною пам'яттю?
18. Які формати команд використовуються при роботі з регістрами процесора?
19. Які головні критерії вибору формату команд?
20. Поясніть різницю між акумуляторною архітектурою, стековою архітектурою та архітектурою на основі регістрів загального призначення
21. Поясніть різницю між архітектурами системи команд типу регістр-регістр, регістр-пам'ять і пам'ять-пам'ять
22. Які переваги та недоліки команд з фіксованим та зі змінним форматом? Який формат є вживанішим в сучасних комп'ютерах і чому?
23. Яким чином знаходяться дані в пам'яті коли в команді відсутня адресна частина?
24. Яка програма має більше команд: та, що складається з безадресних команд, одноадресних команд, чи з двоадресних команд? Чому?
25. Що таке спосіб адресації?
26. Які є способи адресації пам'яті? Їх призначення?
27. Як організовується стекова пам'ять?
28. Поясніть порядок організації обчислень при використанні стекової адресації
29. Наведіть приклади використання інфіксної, префіксної та постфіксної форм запису арифметичних виразів
30. Наведіть приклади безпосередньої, прямої, непрямої, відносної та базової адресацій
31. Чим відрізняється індексна адресація від базової?
32. Чому необхідна велика кількість різних способів адресації?

Лабораторна робота 9 Архітектура сучасних арифметико-логічних пристроїв: елементи RISC - архітектури та конвеєрної обробки даних

Мета роботи : Ознайомитись з архітектурою та правилами побудови сучасних арифметико-логічних пристроїв. Засвоїти поняття RISC - архітектури та конвеєрної обробки даних

Короткі теоретичні відомості

Стекова архітектура

Особливістю обчислювачів , побудованих за стековою архітектурою є те , що вхідні , проміжні та результуючі дані зберігаються в пам'яті даних з послідовним доступом.

Для здійснення операції в стек необхідно записати дані на вершину стека (операція PUSH) . Вершина і наступна за вершиною позиції стека подаються на вхід АЛУ , вихід АЛУ в свою чергу може керувати вершиною стека. OT - визначає виконувану операцію.

PUSH A

PUSH B

ADD

POP C

// Де A , B , C - адреси в основній пам'яті .

Достоїнства :

Простота апаратної реалізації .

Простота запису алгоритмів обчислення.

Простота мнемонічного опису мікрооперацій (з одним або без операндом) .

недоліки:

Стек - запам'ятовуючий пристрій з послідовним доступом володіє повільною швидкістю роботи .

Дана архітектура не дозволяє виробляти розширення або доповнення для збільшення потужності.

Акумуляторна архітектура.

стекова архітектура

Характерною рисою акумуляторної архітектури є використання акумулятора.

Акумулятор - спеціальний регістр , який зберігає операнди або проміжні результати обчислення АЛУ , при цьому сам акумулятор завжди жорстко пов'язаний з одним із входів АЛП.

Для управління таким обчислювачем необхідно завантажувати дані на вільний вхід АЛУ (операція LOAD) . При завантаженні першого операнда він автоматично потрапляє в акумулятор , і копіюється на вхід АЛУ , завантаження другого операнда поміщає його на другий вхід АЛП. АЛУ виконує операцію , а результат виконання поміщає в акумулятор. Для збереження результатів з акумулятора в основну пам'ять використовується операція STORE .

переваги:

Реалізація довільних алгоритмів здійснюється з використанням одне або без операндних інструкцій.

Складність програми еквівалентна складності програми для обчислювачів зі стековою архітектурою.

Відсутня пам'ять з послідовним доступом.

Позитивний ефект від акумулятора , при реалізації ітеративних алгоритмів .

недоліки:

Наявність акумулятора , що вимагає використання операцій послідовної завантаження операндів і збереження результатів .

Регістри загального призначення

Крім основної пам'яті даних використовується швидка пам'ять меншого об'єму з довільним доступом , реалізована у вигляді багатопостового запам'ятовуючого пристрою.

АЛУ може підключати на свої входи довільні регістри і вихід може також підключатися до довільного регістру. У загальному випадку необхідно

завантажувати операнди з основної пам'яті в регістри загального призначення , виконати операцію і зберегти результат в основну пам'ять.

LOAD R0 , A

LOAD R1 , B

ADD R2 , R0 , R1

STORE C , R2

// Де A , B , C - адреси в основній пам'яті , а R0 , R1 , R2 - регістри загального призначення.

За один такт звернення до регістрів загального призначення виконується 3 паралельних дії (2 - е операції читання і 1- а операція запису).

Достоїнста :

Наявність регістрів загального призначення дозволяє довго зберігати як операнди , так і результати обчислень в регістрах загального призначення без завантаження або збереження даних в основній пам'яті (ця особливість лягла в основу КЕШ пам'яті).

недоліки:

Більш складна структура пристрою керування.

Складніша реалізація системного програмного забезпечення через складність команд (до 3 -х операндів) .

RISC архітектура

Традиційна архітектура обчислювальних ядер розвивалася за принципом об'єднання часто використовуваних послідовностей елементарних машинних команд в одну складну мікрооперацію . У результаті сформувався набір команд , що складається як з простих машинних команд , так і з набору більш складних мікрооперацій , що об'єднує в одній машинній команді операції читання / запису даних і арифметичні дії над даними. Через велику кількість способів адресації даних кількість складних мікрооперацій в кілька разів перевищило кількість елементарних машинних команд. Подібна архітектура отримало назву CISC (Complex Instruction Set Computing - обчислювач з повним набором інструкцій).

На думку розробників CISC - архітектури , апаратна підтримка виконання складних машинних команд повинна була збільшити продуктивність програм, що

використовують складні мікрооперації , в порівнянні з програмами , написаними з використанням елементарних машинних команд.

Однак на практиці все було трохи інакше . Проведені в 1970 -х роках співробітниками IBM дослідження показали , що найбільш часто при написанні програм програмісти використовували обмежений набір мікрооперацій , кількість яких становила всього 20 % від повного набору машинних команд CISC - архітектури , інші ж мікрооперації практично ігнорувалися.

Причиною такого дисбалансу з'явилися:

Обмежена підтримка повного набору команд CISC - архітектури існуючими на той момент компіляторами .

Відсутність уніфікованого формату команд , що ускладнювало використання повного набору машинних команд програмістами .

Крім того , складові мікрооперації , покликані збільшити швидкість обчислень , почали програвати у швидкодії послідовностям елементарних машинних команд. Це стало результатом того , що в процесі еволюції обчислювальних ядер основна робота велася над оптимізацією виконання найбільш часто використовуваних елементарних машинних команд. Крім того , за обмеженого набору спеціалізованих реєстрів в CISC - архітектурі , більшість обчислень велося за схемою : читання операндів з оперативної пам'яті в реєстри , виконання арифметичної дії над операндами , запис отриманого результату з реєстра в оперативну пам'ять. Так як швидкість читання даних з оперативної пам'яті в реєстр і запису даних з реєстра в оперативну пам'ять на порядок нижче швидкості пересилання даних між реєстрами , з цієї причини інтенсивна робота з оперативною пам'яттю , властива CISC - архітектурі знижувала продуктивність програм.

Ще одним недоліком CISC - архітектури балу різна довжина машинних команд і різний час їх виконання , це ускладнювало розрахунок часу , необхідного на виконання програми , але крім того не дозволяла реалізувати конвеєрну обробку машинних команд.

Для вирішення проблем , властивих CISC - архітектурі була розроблена нова RISC - архітектура з скороченим набором машинних команд. RISC (Reduced Instruction Set Computer - обчислювач з скороченим набором інструкцій). У набір команд RISC - архітектури увійшли тільки основні елементарні мікрооперації , що дозволило уніфікувати формат команд обчислювального ядра , спростити конструкцію і знизити вартість виготовлення обчислювальних ядер. Розробниками було прийнято рішення зрівняти час виконання всіх машинних команд , що спростило розрахунок

часу виконання програм , а найголовніше дозволило реалізувати конвеєрну обробку інструкцій.

Зменшення набору машинних команд в RISC - архітектурі дозволило розмістити на кристалі обчислювального ядра велика кількість регістрів загального призначення. Збільшення кількості регістрів загального призначення дозволило мінімізувати звернення до повільної оперативної пам'яті , залишивши для роботи з RAM тільки операції читання даних з оперативної пам'яті в регістр і запис даних з регістра в оперативну пам'ять , всі інші машинні команди використовують в якості операндів регістри загального призначення.

Основними перевагами RISC - архітектури є наявність наступних властивостей:

Велике число регістрів загального призначення.

Універсальний формат всіх мікрооперацій .

Рівний час виконання всіх машинних команд.

Практично всі операції пересилання даних здійснюються за маршрутом регістр - регістр .

Рівний час виконання всіх машинних команд дозволяють обробляти потік командних інструкцій з конвеєрним принципом , тобто виконується синхронізація апаратних частин з урахуванням послідовної передачі управління від одного апаратного блоку до іншого.

Апаратні блоки в RISC - архітектурі:

Блок завантаження інструкцій включає в себе наступні складові частини: блок вибірки інструкцій з пам'яті інструкцій , регістр інструкцій , куди поміщається інструкція після її вибірки і блок декодування інструкцій. Цей ступінь називається шаблоном вибірки інструкцій .

Регістри загального призначення спільно з блоками управління регістрами утворюють другу сходинку конвеєра , що відповідає за читання операндів інструкцій. Операнди можуть зберігатися в самій інструкції або в одному з регістрів загального призначення. Цей ступінь називається шаблоном вибірки операндів.

Арифметико- логічний пристрій і , якщо в даній архітектурі реалізований , акумулятор , разом з логікою управління , яка виходячи з вмісту регістра інструкцій визначає тип виконуваної мікрооперації . Джерелом даних крім регістру інструкцій може бути лічильник команд , при виконанні мікрооперацій умовного або безумовного переходу . Дана шабелю називається виконавчою шаблоном конвеєра.

Набір складається з регістрів загального призначення , логіки записи і іноді з RAM утворюють шабелі збереження даних. На цьому ступені результат виконання інструкцій записуються в регістри загального призначення або в основну пам'ять.

Однак до моменту розробки RISC - архітектури , промисловим стандартом мікропроцесорів де- факто стала архітектура Intel x86 , виконана за принципом CISC - архітектури. Наявність великої кількості програм , написаних під архітектуру Intel x86 , унеможливила масовий перехід EOM на RISC - архітектуру. З цієї причини основною сферою використання RISC - архітектури з'явилися мікроконтролери , завдяки тому , що вони не були прив'язані до існуючого програмного забезпечення. Крім того деякі виробники EOM на чолі з IBM так само почали випускати EOM , побудовані за RISC - архітектурі , однак несумісність програмного забезпечення між Intel x86 і RISC - архітектурою в значній мірі обмежувала поширення останніх.

Однак , переваги RISC - архітектури були настільки істотні , що інженери знайшли спосіб перейти на обчислювачі , виконані за RISC - архітектурі , при цьому не відмовляючись від існуючого програмного забезпечення. Ядра більшість сучасних мікропроцесорів , що підтримують архітектуру Intel x86 , виконані за RISC - архітектурі з підтримкою мультіскалярної конвеєрної обробки . Мікропроцесор отримує на вхід інструкцію в форматі Intel x86 , замінюємо її декількома (до 4 -х) RISC - інструкціями .

Таким чином , ядра більшості сучасних мікропроцесорів , починаючи з Intel 486DX , виконані за RISC - архітектурі з підтримкою зовнішнього Intel x86 інтерфейсу . Крім того , переважна більшість мікроконтролерів , а так само деякі мікропроцесори випускаються за RISC - архітектурі.

Конвеєрна обробка

Для реалізації конвеєрного принципу обробки інструкцій необхідно, щоб всі чотири шаблі конвеєра синхронізувалися в часі. Це досягається шляхом установки фіксованого часу , необхідної для роботи кожного ступеня , тобто пристрій управління обчислювальним ядром виділяє рівні проміжки часу протягом яких працюють всі шаблі конвеєра.

При виконанні першої інструкції всі чотири шаблі конвеєра працюють в часі послідовно , при цьому для однієї інструкції одночасно функціонувати не можуть. Після закінчення першої операції блок , відпрацювавши свій такт , три наступних такту простоює.

Конвеєрна організація передбачає одночасну роботу всіх апаратних блоків , але з різними даними. Таким чином для виконання 2 -х інструкцій необхідно 5 конвеєрних тактів , в той час , як в CISC архітектурі для цього треба було б 8 конвеєрних тактів .

Час роботи ступенів конвеєра визначається як максимальний час функціонування самої повільної ступені конвеєра. Для прискорення обчислень використовують суперскалярні обчислення , а для прискорення запису даних в пам'ять додають ще один щабель конвеєра для запису даних в пам'ять.

Безперервне функціонування всього конвеєра забезпечується незалежністю даних і безперервним послідовним використанням апаратних блоків.

Через те , що інструкції часто взаємопов'язані , можуть виникати конвеєрні конфлікти , які полягають у примусовій зупинці однієї або більше шаблів конвеєра.

Посилання (<http://programmators.ru/apparatnoe-ispolnenie/arhitektyra/registry-obschego-naznacheniya.html>)

Питання до виконання практичної роботи

1. Як будується процесор для того, щоб команда виконувалася за один такт?
2. Поясніть принципи роботи RISC процесора комп'ютера .
3. Опишіть фази виконання команди в RISC процесорі комп'ютера DLX.
4. Поясніть роботу конвеєрного RISC процесора комп'ютера DLX.
5. Проаналізуйте та поясніть мікродії, що виконуються RISC-операції при виконанні команд завантаження і збереження (load/store).
6. Проаналізуйте та поясніть мікродії, що виконуються RISC-операції при виконанні команд умовного переходу (branch).
7. Основна ідея суперконвеєрних процесорів.
8. Суперскалярні процесори - структура та принцип роботи.
9. Процесори векторних комп'ютерів - структура та принцип роботи.
10. Наведіть класифікацію архітектури комп'ютера за рівнем суміщення в ньому опрацювання команд та даних.

Лабораторна робота 10. Регістри мікропроцесорів 8086 - 80186

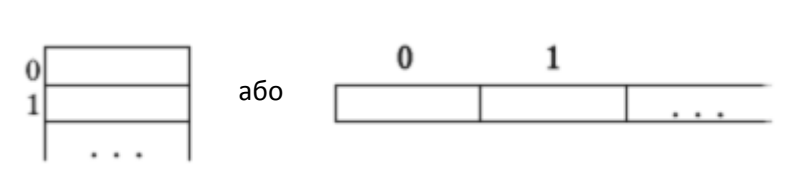
Мета роботи: мати уявлення про регістри мікропроцесорів.

Короткі теоретичні відомості.

Оперативна пам'ять. Регістри

Об'єм оперативної пам'яті ПК - 2^{20} байтів (1 Мб). Байти нумеруються починаючи з 0 , номер байта називається його адресою . Для посилань на байти пам'яті використовуються 20- розрядні адреси : від 00000 до FFFFF (в 16- ковій системі) .

Значимо , що в даному курсі на всіх рисунках, що зображають пам'ять , байти з меншими адресами будуть розташовуватися вгорі (або зліва) , а з великими адресами - внизу (або праворуч) :



Байт містить 8 розрядів (бітів) , кожен з яких може приймати значення 1 або 0. Розряди нумеруються справа наліво від 0 до 7 :

|||||

7 6 5 4 3 2 1 0

час слова і подвійного слова можна обробляти і побайтно .

Зазначимо , що адреса комірки ще неоднозначно визначає комірку , оскільки з цієї адреси може починатися комірка розміром в байт , комірка розміром в слово і комірка розміром в подвійне слово . Тому потрібно ще тим чи іншим способом вказувати розмір комірки . Як це робиться , ми побачимо пізніше.

Байти використовуються для зберігання невеликих цілих чисел і символів , слова - для зберігання цілих чисел і адрес , подвійні слова - для зберігання "довгих" цілих чисел і т.зв. адресних пар (сегмент : зсув) .

Регістри мікропроцесорів 8086 - 80186

Крім комірок оперативної пам'яті для зберігання даних (правда , короткочасного) можна використовувати і регістри - комірки , що входять до складу процесора і доступні з машинної програми . Доступ до регістрів здійснюється значно швидше, ніж до комірок пам'яті , тому використання регістрів помітно зменшує час виконання програм.

Всі регістри мають розмір слова (16 бітів) , за кожним з них закріплено певне ім'я (AX , SP і т.п.). За призначенням і способом використання регістри можна розбити на наступні групи:

- регістри загального призначення (AX , BX , CX , DX , BP , SI , DI , SP) ;
- сегментні регістри (CS , DS , SS , ES) ;
- лічильник команд (IP) ;
- регістр прапорів (Flags) .

Таблиця № 1 . регістри даних

AX		BX		CX		DX	
AH	AL	BH	BL	CH	CL	DH	DL

Таблиця № 2. Регістри - покажчики

CS	DS	ES	SS
----	----	----	----

Таблиця № 3 . сегментні регістри

CS	DS	ES	SS
----	----	----	----

(Розшифровка цих назв : A - accumulator , акумулятор; B - base , база; C - counter , лічильник ; D - data , дані ; BP - base pointer , покажчик бази ; SI - source index , індекс джерела; DI - destination index , індекс приймача ; SP - stack pointer ,

показчик стека ; CS - code segment , сегмент команд ; DS - data segment , сегмент даних ; SS - stack segment , сегмент стека ; ES - extra segment , додатковий сегмент ; IP - instruction pointer , лічильник команд .)

Регістри загального призначення можна використовувати у всіх арифметичних і логічних командах. У той же час кожен з них має певну спеціалізацію (деякі команди "працюють" тільки з певними регістрами) . Наприклад , команди множення і ділення вимагають , щоб один з операндів знаходився в регістрі AX або в регістрах AX і DX (залежно від розміру операнда) , а команди управління циклом використовують регістр CX як лічильника циклу . Регістри BX і BP дуже часто використовуються як базові регістри , а SI і DI - як індексні . Регістр SP зазвичай вказує на вершину стека , апаратно підтримуваного в ПК.

Регістри AX , BX , CX і DX конструктивно влаштовані так , що можливий незалежний доступ до їх старшої та молодшої половин ; можна сказати , що кожен з цих регістрів складається з двох байтових регістрів , що позначаються AH , AL , BH і т.д. (H - high , старший ; L - low , молодший) :

- Таким чином , з кожним із цих регістрів можна працювати як з єдиним цілим , а можна працювати і з його " половинками " . Наприклад , можна записати слово в AX , а потім вважати тільки частина слова з регістра AH або замінити тільки частину в регістрі AL і т.д. Такий пристрій регістрів дозволяє використовувати їх для роботи і з числами , і з символами .

Всі інші регістри не діляться на " половинки " , тому вважати або записати їх вміст (16 бітів) можна тільки цілком.

Сегментні регістри CS , DS , SS і ES не можуть бути операндами ніяких команд , крім команд пересилання і стекові команд. Ці регістри використовуються тільки для сегментування адрес (див. далі).

Лічильник команд IP завжди містить адресу (зсув від початку програми) тієї команди , яка повинна бути виконана наступної (початок програми зберігається в регістрі CS) . Вміст регістра IP можна змінити тільки командами переходу .

Прапорці

I , нарешті , у ПК є особливий регістр прапорців. Прапорець - це біт , що приймає значення 1 (" прапорець встановлений") , якщо виконана деяка умова , і значення 0

(" прапорець скинутий ") в іншому випадку . У ПК використовується 9 прапорців , кожному з них присвоєно певне ім'я (ZF , CF і т.д.). Всі вони зібрані в реєстрі прапорців (кожен прапорець - це один з розрядів реєстра , частина його розрядів не використовується) :

Flags | x | x | x | x | OF | DF | IF | TF | SF | ZF | x | AF | x | PF | x | CF |

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Деякі прапорці прийнято називати прапорцями умов; вони автоматично змінюються при виконанні команд і фіксують ті чи інші властивості їх результату (наприклад , чи дорівнює він нулю). Інші прапорці називаються прапорцями станів ; вони міняються з програми і впливають на подальшу поведінку процесора (наприклад , блокують переривання) .

Прапорці умов :

CF (carry flag) – прапорець переносу. Приймає значення 1 , якщо при додаванні цілих чисел з'явилася одиниця переносу , не « влізла » в розрядну сітку , або якщо при відніманні чисел без знаку перше з них було менше другого . У командах зрушення в CF заноситься біт , що вийшов за розрядну сітку. CF фіксує також особливості команди множення.

OF (overflow flag) – прапорець переповнення . Встановлюється в 1 , якщо при додаванні або вирахуванні цілих чисел зі знаком вийшов результат , по модулю перевершує допустиму величину (сталося переповнення мантиси і вона " залізла " в знаковий розряд).

ZF (zero flag) - прапор нуля . Встановлюється в 1 , якщо результат команди виявився рівним 0 .

SF (sign flag) - прапор знака. Встановлюється в 1, якщо в операції над знаковими числами вийшов негативний результат.

PF (parity flag) - прапор парності . Дорівнює 1 , якщо результат чергової команди

містить парна кількість двійкових одиниць. Враховується звичайно тільки при операціях введення-виведення .

AF (auxiliary carry flag) - прапор додаткового перенесення . Фіксує особливості виконання операцій над двійково- десятковими числами.

Прапорці станів:

DF (direction flag) – прапорець напрямку . Встановлює напрямок перегляду рядків у строкових командах : при DF = 0 рядки проглядаються "вперед" (від початку до кінця) , при DF = 1 - у зворотному напрямку.

IF (interrupt flag) – прапорець переривань. При IF = 0 процесор перестає реагувати на що надходять до нього переривання , при IF = 1 блокування переривань знімається .

TF (trap flag) – прапорець трасування. При TF = 1 після виконання кожної команди процесор робить переривання (з номером 1) , чим можна скористатися при налагодженні програми для її трасування.

Регістри даних (Таблиця № 1)

Можуть використовуватися програмістом за своїм розсудом (за винятком деяких випадків). У них можна зберігати будь-які дані (числа , адреси та ін.)

У верхньому ряду Таблиці (AX (акумулятор) , BX (база) , CX (лічильник) , DX (регістр даних)) знаходяться шістнадцяти-розрядні регістри , які можуть зберігати числа від 0 до 65.535 (від 0h до FFFFh в шістнадцятковій системі (згадуємо останню тему)) . Під ними йде ряд восьми-розрядних регістрів (AH , AL , BH , BL , CH , CL , DH , DL) , які можуть зберігати максимальне число 255 (FFh) . Це половинки (старша або молодша) шістнадцяткових регістрів .

Приклад:

Ми вже знаємо оператор mov , який призначений для завантаження числа в регістр . Щоб привласнити , наприклад , регістру AL число 35h , нам необхідно записати так:

```
mov al , 35h
```

а регістру AX число 346Ah - так :

```
mov ax , 346Ah
```

Якщо ми спробуємо завантажити більше число , ніж може містити регістр , то , при асемблюванні програми відбудеться помилка .

Наприклад , такі записи будуть помилкові:

```
mov ah , 123h а максимум FFh
```

```
mov bx , 12345h а максимум FFFFh
```

```
mov dl , 100h а максимум FFh
```

Тут треба відзначити , що якщо шістнадцяткове число починається з цифри (напр.: 12h) , а з літери (AF) (напр.: C5h) , то перед таким числом ставиться нуль : 0 C5h . Це необхідно для того , щоб програма - асемблер могла відрізнити , де шістнадцяткове число , а де мітка . Нижче ми розглянемо це на прикладі.

Припустимо , ми виконали команду `mov ax , 1234h` . У цьому випадку в регістрі AH перебуватиме число 12h , а в регістрі AL - 34h . Тобто AH , AL , BH , BL , CH , CL , DH і DL - це молодші (Low) або старші (High) половинки шістнадцяткових регістрів.

Таблиця № 4 . Результати виконання різних команд

Команда:	Результат:
<code>mov ax,1234h</code>	AX = 1234h, AH = 12h, AL = 34h
<code>mov bx,5678h</code>	BX = 5678h, BH = 56h, BL = 78h
<code>mov cx,9ABCh</code>	CX = 9ABCh, CH = 9Ah, CL = 0 BCh
<code>mov dx, 0 DEF0h</code>	DX = 0 DEF0h, DH = 0 DEh, DL = 0 F0h

Розглянемо ще два оператори : add і sub .

Оператор add має наступний формат (у наслідку ми завжди будемо оформляти нові команди в такі таблиці):

У стовпці **Команда** буде описуватися нова команда і її застосування . У стовпці **Призначення** - що виконує або для чого служить дана команда , а в стовпці **Процесор** - модель (тип) процесора з якого вона підтримується. **Переклад** - з якого англійського слова утворений оператор і його переклад. У даному прикладі - це 8086 процесор , але працювати команда буде , природно і на наступних, більш сучасних процесорах (80286 , 80386 і т.д.).

Команда add виробляє складання двох чисел.

Команда	Переклад	Назначение	Процесор
ADD приемник, источник	ADDition – сложение	Сложение	8086

Приклади :

mov al , 10 -> завантажуюмо в регістр AL число 10

add al , 15 -> AL = 25; AL - приймач , 15 - джерело

mov ax , 25000 -> завантажуюмо в регістр AX число 25000

add ax , 10000 -> AX = 35000 ; AX - приймач , 10000 - джерело

mov cx , 200 -> завантажуюмо в регістр CX число 200

mov bx , 760 -> а в регістр BX - 760

add cx , bx -> CX = 960 , BX = 760 (BX не змінюється) ; CX - приймач , BX – джерело

Команда	Переклад	Призначення	Процесор
SUB приймач, джерело	SUBtraction- віднімання	Віднімання	8086

Команда sub виробляє віднімання двох чисел.

Приклади :

```
mov al , 10
sub al , 7 -> AL = 3 ; AL - приймач , 7 - джерело
mov ax , 25000
sub ax , 10000 -> AX = 15000 ; AX - приймач , 10000 - джерело
mov cx , 100
mov bx , 15
sub cx , bx -> CX = 85 , BX = 15 ( BX не змінюється ! ) ; CX -
приймач , BX - джерело
```

Регістри - покажчики (Таблиця № 2)

Регістри SI (індекс джерела) і DI (індекс приймача) використовуються в строкових операціях. Регістри BP і SP необхідні при роботі зі стеком . Ми їх будемо детально розглядати в наступних розділах.

Сегментні регістри (Таблиця № 3)

Сегментні регістри необхідні для звернення до того чи іншого сегменту пам'яті (наприклад , відеобуферу) . Сегментація пам'яті досить складна і об'ємна тема , яку також будемо розглядати в наступних розділах.

Команда	Переклад	Призначення	Процесор
INC приймач	Increment - інкремент	Збільшення на одиницю	8086

Команда inc збільшує на одиницю регістр . Вона еквівалентна команді :

```
ADD джерело , 1
```

тільки виконується швидше на старих комп'ютерах (до 80486) і займає менше байт.

Приклади :

```
mov al , 15
inc al -> тепер AL = 16 ( еквівалентна add al , 1)
mov dh , 39h
inc dh -> DH = 3Ah ( еквівалентна add dh , 1)
mov cl , 4Fh
inc cl -> CL = 50h ( еквівалентна add cl , 1)
```

Завдання для практичної роботи

Розглянемо одну невелику програмку , яка виводить на екран повідомлення , і чекає , коли користувач натисне будь-яку клавішу. Після чого повертається в DOS .

Управляти клавіатурою дозволяє переривання 16h . Це переривання BIOS (ПЗУ) , а не MS- DOS (як 21h) . Його можна викликати навіть до завантаження операційної системи , в той час , як переривання 21h доступно тільки після завантаження IO.SYS / MSDOS.SYS (тобто певної частини ОС MS -DOS) .

Щоб зупинити програму до натискання будь-якої клавіші слід викликати функцію 10h переривання 16h . Ось як це виглядає (після стрілки (->) йде коментар) :

```
mov ah , 10h -> у АН завжди вказується номер функції  
int 16h -> викликаємо переривання 16h - сервіс роботи з
```

клавіатурою BIOS (ПЗП)

Після натискання на будь-яку клавішу , комп'ютер продовжить виконувати програму , а регістр АХ буде містити код клавіші , яку натиснув користувач .

Наступна програма виводить на екран повідомлення і чекає натискання будь-якої клавіші (рівнозначна команді " PAUSE " в *. Bat файлах) :

```
(01) CSEG segment  
(02) org 100h  
(03 ) Start :  
(04)  
(05) mov ah , 9  
( 06) mov dx , offset String  
( 07) int 21h  
( 08)  
(09) mov ah , 10h  
( 10 ) int 16h  
( 11 )  
( 12 ) int 20h  
( 13 )  
( 14 ) String db ' Натисніть будь-яку клавішу ... '$  
( 15 ) CSEG ends  
( 16 ) end Start
```

До складання лабораторної надаються: Скомпільована програма з коментарями.

Варіанти завдань до лабораторної роботи

- 1) Написати програму мовою асемблер яка буде виводити довільне повідомлення на екран.
- 2) Написати програму мовою асемблер яка буде зчитувати символ з клавіатури.
- 3) Написати програму мовою асемблер яка буде виконувати арифметичні дії.

I) $2+5-3$

II) $8-5+2$

III) $9-7+5$

Програми

Приклад 1 -----

```
.MODEL small
```

```
.DATA
```

```
    HelloMessage DB 'Hello, world $'
```

```
.CODE
```

```
    mov ax,@data
```

```
    mov ds,ax
```

```
    mov ah,9
```

```
    mov dx,OFFSET HelloMessage
```

```
    int 21h
```

```
    mov ah,4ch
```

```
    int 21h
```

```
END
```

Приклад 2

```
.MODEL small
```

```
.DATA
```

```
    HelloMessage DB '<Write any symbol> $'
```

```
.CODE
```

```
    mov ax,@data
```

```
    mov ds,ax
```

```
mov ah,9
mov dx,OFFSET HelloMessage
int 21h
mov ah,1
int 21h
mov ah,4ch
int 21h
```

END

Приклад 3

```
.MODEL small
```

```
.DATA
```

```
    rez DW dup(0)
```

```
.CODE
```

```
mov ax,@data
```

```
mov ds,ax
```

```
mov ax,2
```

```
mov bx,5
```

```
add ax,bx
```

```
mov bx,3
```

```
sub ax,bx
```

```
mov rez,ax
```

```
mov ah,4ch
```

```
int 21h
```

END

Лабораторна робота 11

Асемблювання програм

Мета роботи: мати уявлення про мову Assembler . Отримати практичні навички роботи на даному мовою.

Короткі теоретичні відомості .

Загальний принцип асемблювання програм.

Крок 1 . асемблювання

Програма - асемблер (MASM , TASM , WASM , NASM і пр.) створює об'єктний файл з розширенням OBJ . Цей файл є перехідним між асемблерним файлом (. ASM) і програмою (. COM / . EXE) .

У разі , якщо асемблерний лістинг занадто великий , то програму розбивають на кілька частин. У більшості випадків обходяться директивою INCLUDE (так чином ми будемо асемблювати нашу оболонку) .

Однак , якщо файли , що приєднуються зазначеної вище директивою великі і в основному незмінні (тобто готові процедури , які не потребують редагування) , то постійне асемблювання цих процедур може зайняти багато часу. У такому випадку , кожна окремо взята частина програми (асемблерний код) асемблює окремо , при цьому створюється один або кілька об'єктних файлів (. OBJ) , які не вимагають поточноасемблювання , тільки компонування (лінковки) (див. Крок 2).

Крок 2 . компонування

Якщо в процесі асемблювання не було виявлено помилок у асемблерного лістингу , то програма - асемблер створює об'єктний файл (. OBJ) .

Потім необхідно скористатися лінковщик (компоновщиком) , які входять у комплект програми - асемблера . Дана процедура виконується набагато швидше асемблювання .

Саме компоувальник створює готовий до запуску файл (програму) з розширенням COM або EXE з об'єктного файлу (. OBJ) . Обидва типи мають відмінності в структурі асемблерної програми . Перший тип (COM) не може перевищувати 64Кб і використовується тільки в MS - DOS , проте він дуже компактний і зручний для написання невеликих програм і резидентів під операційну систему Microsoft DOS . У більшості випадків , якщо програма написана на чистому Асемблері під MS - DOS , немає необхідності створювати EXE -файли. У цій книзі в першій частині розглядаються саме програми типу COM.

Для створення стандартних програм типу EXE під MS - DOS немає необхідності вказувати небудь параметри лінковщик при компонуванні , чого не скажеш про створення програм типу COM. Справа в тому , що компоувальник не може автоматично визначити який тип піддається компонуванні .

Лінковщик також перевіряє , чи немає яких-небудь помилок в об'єктних файлі , але не граматичних , а логічних . Наприклад , відсутність необхідної об'єктної бібліотеки , зазначеної в самому файлі або в командному рядку (програма - асемблер цього не робить).

Якщо помилок не було виявлено , компоувальник створює машинний код (програму типу COM або EXE), яку можна запускати на виконання.

Примітка . Виходячи з усього вищевикладеного , робимо висновок , що для створення машинного коду необхідно скористатися двома програмами : програмою асемблером і компоувщиком .

Однак , для програми - асемблера MASM версій 6.00 - 6.13 достатньо вказати в командному рядку параметр " / AT " у процесі асемблювання . У такому випадку MASM після асемблювання (якщо не було помилок у асемблерного лістингу) автоматично запустить компоувальник (LINK . EXE) , який створить файл типу COM.

Асемблювання і компоування програмними пакетами Microsoft (MASM)

Припустимо , ви створили в текстовому редакторі файл з ім'ям PROG . ASM .

Тоді :

- Якщо Ви використовуєте MacroAssembler версії 6.11 - 6.13 (MASM 6.11 - 6.13) : У командному рядку необхідно вказати наступне:

```
> ML.EXE PROG.ASM / AT
```

У результаті створюється два файли: PROG.OBJ і PROG.COM . Р ROG . OBJ , як правило , нам більше не знадобиться , і його можна видалити , а PROG . COM запускаєте на виконання. Це і буде машинний код асемблерної програми .

Параметр " / AT " вказує програмі - асемблеру (MASM) , що після асемблювання , у разі , якщо помилок не буде виявлено , слід запустити компоувальник (LINK . EXE) і передати йому параметри для створення файлу типу COM.

Зверніть увагу , що параметр " / AT " повинен бути зазначений ЗАГЛАНІМІ символами !

- Якщо Ви використовуєте TurboAssembler (TASM) :

У командному рядку необхідно вказати наступне:

```
> TASM.EXE PROG.ASM
```

Якщо prog.asm не містить помилок , то в результаті створюється файл PROG.OBJ , якою скомпоувати за допомогою компоувальника (лінковщик) TLINK . EXE :

```
> TLINK.EXE PROG.OBJ / t
```

T LINK . EXE створить файл PROG . COM , який і потрібно запустити на виконання. Параметр " / t " вказує TLINK . EXE , що необхідно створити файл типу COM.

Отже, переходимо до нашої першої програмі :

```
(01) CSEG segment
(02) org 100h
(03)
(04) Begin :
(05)
( 06) mov ah , 9
( 07) mov dx , offsetMessage
( 08) int 21h
(09)
( 10 ) int 20h
( 11 )
( 12 ) Message db ' Hello , world ! $'
( 13 ) CSEG ends
( 14 ) endBegin
```

Для того , щоб пояснити всі оператори даного прикладу , нам потрібно кілька глав . Тому опис деяких команд ми просто опустимо на даному етапі. Просто вважайте , що так має бути. У самий найближчий час ми розглянемо ці оператори докладно. Отже , рядки з номерами (01) , (02) і (13) ви просто ігноруєте .

Рядки (03) , (05) , (09) і (11) залишаються порожніми . Це робиться для наочності . Асемблер їх буде просто опускати .

Тепер перейдемо до розгляду інших операторів. З рядка (04) починається код програми . Це мітка , яка вказує Асемблеру на початок коду. У рядку (14) коштують оператори endBegin (Begin - від англ. - Початок; end - кінець). Це кінець програми . Взагалі замість слова Begin можна було б використовувати що-небудь інше . Наприклад , Start : . У такому випадку , нам довелося б і завершувати програму EndStart (14).

Рядки (06) - (08) виводять на екран повідомлення "Hello , world ! " . Тут доведеться коротенько розповісти про реєстрах процесора (більш докладно цю тему ми розглянемо у наступних розділах) .

Реєстр процесора - це спеціально відведена пам'ять для зберігання якого-небудь числа .

Наприклад:

Якщо ми хочемо скласти два числа , то в математиці запишемо так:

$$A = 5$$

$$B = 8$$

$$C = A + B.$$

A , B і C - це свого роду реєстри (якщо говорити про комп'ютер) , в яких можуть зберігатися деякі дані . A = 5 слід читати як: " Привласнюємо A число 5".

Для присвоєння реєстру якогось значення, в Асемблері існує оператор mov (від англ. Move - в даному випадку - завантажити) . Рядок (06) слід читати так : " Завантажуємо в реєстр AH число 9 " (простіше кажучи , присвоюємо AH число 9) . Нижче розглянемо , навіщо це треба.

У рядку (07) завантажуюємо в реєстр DX адреса повідомлення для виводу (в даному прикладі це буде рядок "Hello , world ! \$ ") .

Далі , у рядку (08) , викликаємо переривання MS- DOS , яке і виведе нашу рядок на екран. Переривання будуть детально розглядатися в наступних розділах. Тут я скажу кілька слів.

Переривання MS- DOS - це свого роду підпрограма (частина MS -DOS) , яка знаходиться постійно в пам'яті і може викликатися в будь-який час з будь-якої програми .

Розглянемо вищесказане на прикладі :

Програма (алгоритм) додавання двох чисел

Початок основної програми

A=5 у змінну A заносимо значення 5

B=8 у змінну B заносимо значення 8

Визов підпрограми додавання

Тепер C дорівнює 13

A=10

B=15

Визов підпрограми додавання з новими значення ми A і B

Тепер C дорівнює 13

Кінець основної програми

Підпрограма додавання

$C=A+B$

Повертання з підпрограми в місце її визивання

Кінець підпрограми додавання

У даному прикладі ми двічі викликали підпрограму Додавання , яка склала два числа , передані їй в змінних A і B. Результат поміщається в змінну C. Коли викликається підпрограма , комп'ютер запам'ятовує з якого місця вона була викликана , а потім , коли закінчила роботу підпрограма , комп'ютер повертається в те місце , звідки вона викликала . Т.ч. , можна викликати підпрограми невизначену кількість разів з будь-якого місця .

При виконанні рядка (08) програми на Асемблері ми викликаємо підпрограму (в даному випадку - це називається переривання) , яка виводить на екран рядок. Для цього ми , власне , і поміщаємо необхідні значення в реєстри . Всю роботу (висновок рядка , переміщення курсору) бере на себе підпрограма . Рядок (08) слід

читати так : викликаємо двадцять першому переривання (int - від англ. Interrupt - переривання) . Зверніть увагу , що після числа 21 стоїть буква h . Це , як ми вже знаємо , шістнадцяткове число (33 в десятковій системі). Звичайно , нам нічого не заважає замінити рядок int 21h на int 33. Програма буде працювати коректно . Просто в Асемблері прийнято вказувати номери переривань в шістнадцятковій системі .

У рядку (10) ми , як ви вже здогадалися , викликаємо переривання 20h . Для виклику даного переривання не потрібно вказувати які-небудь значення в регістрах . Воно виконує тільки одну задачу : вихід з програми (вихід в DOS) . В результаті виконання переривання 20h , програма повернеться туди , звідки її запускали (завантажували , викликали) . Наприклад , в NortonCommander або DOS Navigator .

Рядок (12) містить повідомлення для виводу. Перше слово (message - повідомлення) - назва повідомлення. Воно може бути будь-яким (наприклад , mess або string і пр.) . Зверніть увагу на рядок (07) , в якій ми завантажуюємо в регістр DX адресу нашого повідомлення.

Можна створити ще один рядок , яку назвемо Mess2 . Потім , починаючи з рядка (09) вставимо наступні команди:

```
...
(09) movah , 9
( 10 ) mov dx , offset Mess2
( 11 ) int 21h
( 12 ) int 20h
( 13 ) Messagedb ' Hello , world ! $'
( 14 ) Mess2 db ' Це Я! $'
( 15 ) CSEG ends
( 16 ) endBegin
```

Зверніть увагу на останній символ у рядках Message і Mess2 - \$. Він вказує на кінець рядка. Якщо ми його приберемо , то 21h переривання продовжить висновок до тих пір, поки не зустрінеться де-небудь в пам'яті символ \$. На екрані , крім нашої рядки на самому початку , ми побачимо "сміття " (різні символи , яких у рядку зовсім немає) .

Тепер асемблюйте програму.

Для виконання роботи необхідно виконати наступне.

- 1) Встановити MASM32 і Emu8086 які додаються в лабораторній роботі.
- 2) Скомпілювати і виконати програми які надаються в файлі «Lab_4.txt»
- 3) Написати програму мовою асемблер яка буде виконувати обрахунки відповідно варіанту.

I) 2+5-1

II) 8-5+1

III) 9-7+1

IV) 4+2-1

До складання лабораторної надаються: Скомпільована програма.

Лабораторна робота 12 Сегментація пам'яті.

Мета роботи: Ознайомитись з принципами та отримати практичні навички щодо сегментації пам'яті при роботі з процесорами i80x86.

Короткі теоретичні відомості.

Для того, щоб краще зрозуміти сегментацію пам'яті, нам потрібно скористатися відладчиком. Можна працювати з CodeView(CV.EXE) і AFD Pro(AFD.EXE).

Припустимо, ви написали програму на асемблері та назвали її PR.ASM.

Засемблювавши, ви отримали файл PR.COM. Тоді, щоб запустити її під відладчиком CodeView/AFD, необхідно набрати в командному рядку MS-DOS наступне:

CV.EXE PR.com

або:

AFD.EXE pr.com

або TD.EXE pr.com якщо випрацюєте в TASM5.0

Розглянемо, як упам'яті комп'ютера зберігаються дані.

Взагалі, як комп'ютер може зберігати, наприклад, слово "диск"?

Один біт може бути нулем або одиницею. Кожен наступний встановлений біт (починаючи праворуч) збільшує число в два рази: 0001 у нашому прикладі - одиниця ; 0010 - два ; 0100 - чотири; 1000 - вісім і т.д. Це і є т.зв. двійкова форма представлення даних.

Т.ч. , щоб позначити числа від 0 до 9 , нам потрібно чотири біти (хоч вони і не до кінця використані. Можна було б продовжити: десять - 1010 , одинадцять - 1011 ... , п'ятнадцять - 1111) .

Комп'ютер зберігає дані в пам'яті саме так. Для позначення якого-небудь символу (цифри , букви, коми , крапки ...) в комп'ютері використовується певна кількість біт. Комп'ютер " розпізнає " 256 (від 0 до 255) різних символів по їх коду . Цього достатньо , щоб вмістити всі цифри (0 - 9) , літери латинського алфавіту (а - z , А - Z) , українського (а - я , А - Я) та ін Для представлення символу з максимально можливим кодом (255) потрібно 8 біт. Ці 8 біт називаються байтом.

Можна елементарно перевірити. Створіть у текстовому редакторі файл з будь-яким ім'ям і запишіть у ньому один символ , наприклад , "М" (але не натискайте Enter !) .

Якщо ви подивитесь його розмір , то файл буде дорівнює 1 байту . Якщо ваш редактор дозволяє дивитися файли в шістнадцятковому форматі , то ви зможете дізнатися і код збереженого символу. У даному випадку - буква " М" має код 4Dh в шістнадцятковій системі , яку ми вже знаємо або 1001101 у двійковій .

Т.ч. слово " диск" буде займати 4 байти або $4 * 8 = 32$ біта . Як ви вже зрозуміли , комп'ютер зберігає в пам'яті не самі букви (символи) цього слова , а послідовність " одиничок " і " нуликів " .

Всю роботу з виведення самого 0 символу на екран (а не бітів) виконує відеокарта (

відеоадаптер) , яка знаходиться у вашому комп'ютері. І якби її не було , то ми , природно , нічого б не бачили , що у нас відображено на екрані.

У Асемблері після двійкового числа завжди повинна стояти літера " b " . Це потрібно для того , щоб при асемблюванні програми Асемблер зміг відрізнити десяткові , шістнадцяткові та виконавчі числа. Наприклад: 10 - це "десять" , 10h - це " шістнадцять " а 10b - це " два" в десятковій системі .

Т.ч. в регістри можна завантажувати виконавчі , десяткові і шістнадцяткові числа.

наприклад:

```
...  
movax , 20  
movbh , 10100b  
movcl , 14h  
...
```

У результаті в регістрах AX , BH і CL перебуватиме одне і теж число , тільки завантажуюмо ми його , використовуючи різні системи числення . Комп'ютер же буде зберігати його в двійковому форматі (як в регістрі BH) .

У комп'ютері вся інформація зберігається в двійковому форматі (двійковій системі) приблизно в такому вигляді: 10101110 10010010 01111010 11100101 (природно , без пробілів. Для зручності я розділив байти) . **Вісім біт - це один байт.** Один символ займає один байт , тобто вісім біт.

Сегментація пам'яті в реальному режимі .

Візьмемо таке речення: "Вивчаємо сегменти пам'яті". Тепер давайте порахуємо , на якому місці стоїть буква " и " в слові " сегменти " від початку пропозиції , включаючи пробіли ... На шістнадцятому . Підкреслюю , що ми вважали від початку пропозиції .

Тепер трохи ускладнити завдання і розіб'ємо речення наступним чином (символом " _ " позначений пробіл) :

Приклад № 1:

```
0000 : Вивчаємо_  
0010 : сегменти_  
0020 : пам'яті  
0030 :
```

У слові "Вивчаємо " символ " В " стоїть на нульовому місці ; символ "и" на першому

, "в" на другому і т.д. У даному випадку ми вважаємо літери , починаючи з нульової позиції , використовуючи два числа. Назвемо їх сегмент і зсув . Тоді , символ " ч " матиме таку адресу: 0000:0003 , тобто сегмент 0000 , зсув 0003 .

У слові " сегменти " будемо вважати букви , починаючи з десятого сегмента , але з нульового зсуву. Тоді символ " н " матиме таку адресу: 0010:0005 , тобто п'ятий символ , починаючи з десятої позиції : 0010 - сегмент , 0005 - зміщення .

У слові "пам'ять " вважаємо літери , починаючи з 0020 сегмента і також з нульовою позиції. Т.ч. символ "а" буде мати адресу 0020:0001 , тобто сегмент - 0020 , зсув - 0001 . Знову перевіримо ...

Отже , ми з'ясували , що для того , щоб знайти адресу потрібного символу , необхідні два числа: сегмент і зміщення всередині цього сегменту . У Асемблері сегменти зберігаються в сегментних регістрах : CS , DS , ES , SS, а зміщення можуть зберігатися в інших (але не у всіх) .

- **Регістр CS** служить для зберігання сегмента коду програми (CodeSegment - сегмент коду) ;

- **Регістр DS** - для зберігання сегменту даних (DataSegment - сегмент даних);

- **Регістр SS** - для зберігання сегмента стека (StackSegment - сегмент стека);

- **Регістр ES** - додатковий сегментний регістр , який може зберігати будь-який інший сегмент (наприклад , сегмент відеобуфера) .

Приклад № 2:

Давайте спробуємо завантажити в пару регістрів ES : DI сегмент і зсув літери " м " у слові "пам'яті" з Прикладу № 1 (див. вище). Ось як це запишеться на Асемблері :

...

```
( 1 ) movax , 0020
```

```
( 2 ) moves , ax
```

```
( 3 ) movdi , 2
```

...

Тепер в регістрі ES знаходиться сегмент з номером 20 , а в регістрі DI - зміщення до букви (символу) " м " у слові " пам'яті". Перевірте , будь ласка ...

Тут варто відзначити , що завантаження числа (тобто якого-небудь сегмента) безпосередньо в сегментний регістр заборонена . Тому ми в рядку (1) завантажили

сегмент в AX , а в рядку (2) завантажили в регістр ES число 20, яке знаходилося в AX :

```
movds , 15 -> помилка!  
movss , 34h -> помилка!
```

Коли ми завантажуюмо програму в пам'ять , вона автоматично розташовується в першому вільному сегменті. У файлах типу * . Com всі сегментні регістри автоматично ініціалізувалися для цього сегмента (встановлюються значення рівні того сегменту , який завантажено програма). Це можна перевірити за допомогою відладчика . Якщо , наприклад , ми завантажуюмо програму типу * . Com в пам'ять , і комп'ютер знаходить перший вільний сегмент з номером 5674h , то сегментні регістри будуть мати наступні значення :

```
CS = 5674h  
DS = 5674h  
SS = 5674h  
ES = 5674h
```

Інакше кажучи : $CS = DS = SS = ES = 5674h$

Код програми типу * . Com повинен починатися зі зсуву 100h . Для цього ми , власне, і ставили в наших минулих прикладах програм оператор ORG 100h , вказуючи Асемблеру при асемблюванні використовувати зсув 100h від початку сегменту , в який завантажена наша програма (пізніше ми розглянемо, чому так). Сегментні ж регістри , як я вже говорив , автоматично беруть значення того сегменту, в який завантажилася наша програма.

Пара регістрів CS : IP задає поточний адресу коду. Тепер розглянемо , як все це відбувається на конкретному прикладі:

Приклад № 3 .

```
(01) CSEG segment  
(02) org 100h  
(03) _start :  
(04) movah , 9  
(05) movdx , offsetMy_name
```

(06) int 21h
(07) int 20h
(08) My_name db ' Dima \$'
(09) CSEG ends
(10) end_start

Отже , рядки (01) та (09) описують сегмент :

CSEG (даємо ім'я сегменту) **segment** (оператор Асемблера , який вказує , що ім'я CSEG - це назва сегмента) ;

CSEGends (END Segment - кінець сегмента) вказує Асемблеру на кінець сегмента. Рядок (02) повідомляє , що код програми (як і зміщення всередині сегменту CSEG) необхідно відраховувати з 100h . За цією адресою в пам'ять завжди завантажуються програми типу * . Com .

Запускаємо програму з Прикладу № 3 в відладчик . Припустимо , вона завантажилася у вільний сегмент 1234h . Перша команда у рядку (04) буде розташовуватися за такою адресою :

1234h : 0100h (тобто CS = 1234h , а IP = 0100h) (подивіться в відладчик на регістри CS і IP).

Перейдемо до наступної команді (в відладчикCodeView натисніть клавішу F8 , у AFD - F1 , в іншому - подивіться , яка клавіша потрібна ; буде написано щось на кшталт " F8 -Step " або " F7 - Trace ") . Тепер ви бачите , що змінилися такі регістри :

- **AX** = 0900h (точніше , AH = 09h , а AL = 0, тому ми завантажили командою movah , 9 число 9 в регістр AH , при цьому не чіпаючи AL . Якби AL дорівнював , скажімо , 15h , то після виконання даної команди AX б дорівнював 0915h)
- **IP** = 102h (тобто вказує на адресу наступної команди. З цього можна зробити висновок , що команда movah , 9 займає 2 байти: 102h - 100h = 2).

Наступна команда (натискаємо клавішу F8 / F1) змінює регістри DX і IP. Тепер DX вказує на зміщення нашої рядки (" Dima \$ ") відносно початку сегмента , тобто 109h , а IP дорівнює 105h (тобто адреса наступної команди). Неважко порахувати , що команда movdx , offsetMy_name займає 3 байти (105h - 102h = 3).

Зверніть увагу , що в Асемблері ми пишемо :

movdx , offsetMy_name

а в відладчик бачимо наступне :

movdx , 109; (109 - шістнадцяткове число , але CodeView і багато інших відладчики символ ' h ' не ставлять. Це треба мати на увазі) .

Чому так відбувається? Справа в тому , що при асемблюванні програми , програма - асемблер (MASM / TASM) підставляє замість offsetMy_name реальна адреса

рядка з ім'ям My_name в пам'яті (її зсув) . Можна , звичайно , записати одразу :
movdx , 109h

Програма буде працювати нормально. Але для цього нам потрібно вирахувати самим цю адресу. Спробуйте вставити наступні команди , починаючи з рядка (07) у Прикладі№ 3:

```
...  
( 07) int 20h  
( 08) int 20h  
(09) My_name db ' Dima $'  
( 10 ) CSEG ends  
( 11 ) end_start
```

Просто продублюємо команду int 20h (хоча , як ви вже знаєте , до рядка (08) програма не дійде) .

Тепер асемблює програму заново. Запускайте її під відладчиком . Ви побачите , що в DX завантажуюмо не 109h , а інше число . Подумайте , чому так відбувається. Це просто!

У вікні " Memory " (" Пам'ять") відладчикаCodeView (у AFD щось подібне) ви повинні побачити приблизно наступне:

1234	:	0000	CD 20 00 A0 00 9A F0 FE	= .a.
№1		№2	№3	№4

Позиція № 1 (1234) - сегмент , в який завантажилася наша програма (може бути будь-яким) .

Позиція № 2 (0000) - зміщення в даному сегменті (сегмент і зсув відокремлюються двокрапкою (:)) .

Позиція № 3 (CD 20 00 ... F0 FE) - код в шістнадцятковій системі , який розташовується з адреси 1234:0000 .

Позиція № 4 (= . A .) - Код в ASCII (нижче розглянемо) , відповідний шістнадцятиричним числах з правого боку.

У Позиції № 2 (зміщення) введіть значення , яке знаходиться в регістрі DX після виконання рядки (5). Після цього в Позиції № 4 ви побачите рядок " Dima \$", а в Позиції № 3 - коди символів " Dima \$" в шістнадцятковій системі ... Так ось що завантажуюється в DX ! Це не що інше , як АДРЕСА (зміщення) нашої рядка в сегменті !

Але повернемося. Отже , ми завантажили в DX адреса рядка в сегменті , який ми назвали CSEG (рядки (01) і (09) у Прикладі № 3). Тепер переходимо до наступної команді: int 21h . Викликаємо переривання DOS з функцією 9 (movah , 9) та

адресою рядка в DX (movdx , offsetMy_name) .

Для використання переривань у програмах , в АН заноситься номер функції .

Номери функцій бажано запам'ятовувати, щоб постійно не шукати в довідниках яка функція що робить.

Функція 09h переривання 21h виводить рядок на екран , адреса якої вказана в регістрі DX.

Взагалі , будь-яка рядок, що складається з ASCII символів , називається ASCII - рядок . ASCII символи - це символи від 0 до 255 в DOS , куди входять літери російського і латинського алфавітів , цифри , знаки пунктуації та пр.

Функція 09h переривання 21h - висновок рядка символів на екран в поточну позицію курсору:

Вхід	АН = 09h DX=адрес ASCII- рядки символів, закінчившись
Вихід	Нічого

У полі "**Вхід**" ми вказуємо , в які регістри що завантажувати перед викликом переривання , а в полі "**Вихід**" - що повертає функція . Порівняйте цю таблицю з Прикладом № 3 .

Завдання для практичного виконання.

Виводить у верхній лівий кут екрану « веселу пику» на синьому фоні:

(01) CSEG segment

(02) org 100h

(03) _beg :

(04) movax , 0B800h

(05) moves , ax

(06) movdi , 0

(07)

(08) movah , 31

(09) moval , 1

(10) moves : [di] , ax

(11)

```
( 12 ) movah , 10h
( 13 ) int 16h
( 14 )
( 15 ) int 20h
( 16 )
( 17 ) CSEG ends
( 18 ) end _beg
```

У даному прикладі , для виведення символу на екран , ми використовуємо метод прямого відображення в відеобуфер .

У рядках (04) і (05) завантажуюмо в сегментний регістр ES число 0B800h , яке відповідає сегменту дисплея в текстовому режимі (запам'ятайте його !) . У рядку (06) завантажуюмо в регістр DI нуль. Це буде зсув щодо сегмента 0B800h . У рядках (08) і (09) в регістр AH заноситься атрибут символу (31 - яскраво -білий символ на синьому тлі) і в AL - ASCII -код символу (01 - це пика) .

У рядку (10) заносимо за адресою 0B800 : 0000h (тобто перший символ в першому рядку дисплея - верхній лівий кут) атрибут і ASCII –

код символу (31 і 01 відповідно).

Зверніть увагу на запис регістрів в рядку (10). Квадратні дужки ([]) вказують на те , що треба завантажити число не в регістр , а за адресою , який міститься в цьому регістрі (в даному випадку , як уже зазначалося , - це 0B800 : 0000h) .

ПЕРЕЛІК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. Корнійчук В. і., Тарасенко В. П, О.В. Тарасенко-Клятченко. Основи комп'ютерної арифметики. - Київ, «Корнійчук». - 2007.
2. Мельник А. М, Архітектура комп'ютера. -Луцьк. -2008 р. -470 с.
3. Таненбаум З. Архитектура комп'ютера. С-Пб: Питер. - 2003 р. - 697 с.
4. Бройдо В. Л., Ильина О. П. Архитектура ЭВМ и систем. - С-Пб.: Питер. -2006. -718 с.
5. В.Л. Григорьев. Микропроцессор i486. Архитектура и программирование (в 4-х книгах). – М., ГРАНАЛ, 1993 – 346 с., ил.
6. Бройдо В.Л., Ильина О.П. Архитектура ЭВМ и систем: Учебник для вузов – Питер, 2006.–718 с.