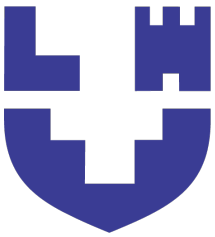
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ



АРХІТЕКТУРА КОМП'ЮТЕРІВ

Методичні вказівки до лабораторних занять
для здобувачів першого (бакалаврського) рівня вищої освіти
освітньо-професійної програми «Комп'ютерна інженерія»
галузь знань 12 (F) «Інформаційні технології»
спеціальність 123 (F7) «Комп'ютерна інженерія»
денної та заочної форм навчання

Луцьк 2025

УДК 004.451
А-87

Електронна копія друкованого видання передана для внесення в репозитарій ЛНТУ
Директор бібліотеки _____ Наталія ПОЛІЩУК

Рекомендовано до видання вченою радою факультету комп'ютерних та інформаційних технологій ЛНТУ,
протокол № від « __ » 2025 року

Голова вченої ради факультету КІТ _____ Інна КОНДІУС

Розглянуто і схвалено на засіданні кафедри комп'ютерної інженерії та безпеки ЛНТУ, протокол № від « __ » 2025 року

Завідувач кафедри комп'ютерної
інженерії та безпеки _____ Тарас ТЕРЛЕЦЬКИЙ

Укладач: _____ Наталія ХРИСТИНЕЦЬ, кандидат технічних наук, доцент кафедри комп'ютерної інженерії та безпеки ЛНТУ

Рецензент: _____ Наталія ЯКИМЧУК, кандидат технічних наук, доцент кафедри електроніки та телекомунікацій ЛНТУ

Відповідальний за випуск: _____ Тарас ТЕРЛЕЦЬКИЙ, доцент, завідувач кафедри комп'ютерної інженерії та безпеки ЛНТУ

А-87 **Архітектура комп'ютерів**: методичні вказівки до лабораторних занять для здобувачів першого (бакалаврського) рівня вищої освіти освітньо-професійної програми «Комп'ютерна інженерія» галузі знань 12 (F) «Інформаційні технології» спеціальності 123 (F7) Комп'ютерна інженерія денної та заочної форм навчання / уклад. Н. А. Христинець. Луцьк: Луцький НТУ, 2025. 92 с.

Методичні вказівки складено відповідно до діючої програми курсу «Архітектура комп'ютерів». Вони можуть бути використані студентами технічних спеціальностей при вивченні даної дисципліни.

Н.А. Христинець 2025

ЗМІСТ

ВСТУП

Методичні вказівки до лабораторних занять з дисципліни «Архітектура комп’ютерів» призначені для систематичного ознайомлення студентів із будовою, функціонуванням та принципами роботи сучасних комп’ютерних систем. Вони охоплюють різні аспекти архітектури комп’ютерів, починаючи від основних елементів материнської плати та системних шин, організації оперативної та кеш-пам’яті, до роботи блоків живлення, BIOS/UEFI та системного таймера. Таке поетапне опанування матеріалу дозволяє студентам закріплювати теоретичні знання практичними навичками через виконання лабораторних завдань.

Лабораторні роботи спрямовані на розвиток у студентів практичних компетентностей у сфері інформаційних технологій та комп’ютерної інженерії. Їх виконання дозволяє відпрацювати навички роботи з апаратними та програмними засобами, зрозуміти принципи взаємодії компонентів комп’ютера та механізми оптимізації їхньої роботи. Це сприяє формуванню професійних навичок, необхідних для проєктування, налагодження та аналізу комп’ютерних систем.

Окремий блок методичних вказівок присвячено сучасним процесорним архітектурам, включаючи аналіз Intel Core, AMD Ryzen та ARM. Порівняння різних рішень та оцінка перспектив розвитку комп’ютерних архітектур дозволяють студентам глибше зрозуміти тенденції розвитку апаратного забезпечення та його вплив на ефективність обчислювальних систем. Такий комплексний підхід забезпечує системне формування знань і вмінь, необхідних для професійної діяльності у сфері комп’ютерної інженерії.

Лабораторна робота №1. Будова та функції материнської плати.

Мета роботи: Провести огляд архітектури системного блоку комп'ютера. Визначити основні складові, їх характеристики. Вивчити будову, призначення та функціональність основних компонентів ПК.

Теоретичні відомості

Материнська плата (motherboard) – це основна друкована плата комп'ютера, яка забезпечує фізичне та логічне з'єднання всіх компонентів системи. Сучасні материнські плати (рис. 1.1) виконують роль комунікаційного центру, через який взаємодіють процесор, пам'ять, накопичувачі, графічні та інші пристрої. Від вибору материнської плати залежить сумісність із типом процесора, обсягом оперативної пам'яті, швидкістю шини та кількістю підключених пристроїв [1].

Рисунок 1.1 – Материнська плата MSI Z890 Gaming Plus Wi-Fi (s1851, Intel Z890, PCI-Ex16) ATX 2024 року [1]

Тобто, материнська плата – це сполучна ланка для всього комп’ютера. У неї встановлюється процесор, відеокарта, оперативна пам’ять, накопичувачі і вся периферія. При цьому кожен елемент має свій слот підключення. Для процесора – це сокет, для внутрішнього накопичувача даних – SATA, для відеокарти, звукової карти, ТВ-тюнера, вбудованого WI-FI адаптера PCI-слот. Основним елементом материнської плати є процесорний роз’єм (сокет). Він визначає, який саме процесор можна встановити. Наприклад, для процесорів Intel 14-го покоління використовується сокет LGA1700, а для новіших AMD Ryzen – AM5. Ці сокети відрізняються кількістю контактів, формою та електричними параметрами. Крім цього, материнські плати підтримують певний чипсет, який визначає функціональні можливості плати: кількість USB, підтримка PCIe 4.0/5.0, можливість розгону тощо.

Ще одним критичним вузлом є слоти оперативної пам’яті (DIMM). Вони призначені для встановлення модулів RAM, зазвичай стандарту DDR4 або DDR5. Кількість слотів (2–4 у звичайних платах, до 8 у HEDT/серверних) впливає на загальний обсяг пам’яті, яку можна встановити, а підтримка багатоканального режиму – на продуктивність системи.

Материнські плати також містять шини розширення, найпоширеніша з яких – PCI Express (PCIe). Слоти PCIe x16 використовуються переважно для відеокарт, а x1 – для додаткових плат (мережеві, звукові, тощо). З появою PCIe 5.0 та 4.0 зросла пропускна здатність, що дозволяє підключати високошвидкісні SSD та GPU.

Важливим параметром плати є її **форм-фактор** – фізичні розміри та розташування кріплень. Найпоширеніші: ATX, microATX та Mini-ITX. Форм-фактор визначає, який корпус можна використати, кількість слотів розширення та тип живлення. **Чипсет** – це набір мікросхем на материнській платі, який координує обмін даними між процесором, пам’яттю, відеокартою, накопичувачами та периферією. Його основне завдання – забезпечити взаємодію між усіма компонентами комп’ютера.

У класичній архітектурі ПК (до 2008–2010 років) чипсет складався з двох окремих мікросхем:

Північний міст (Northbridge) відповідав за високошвидкісні компоненти: зв'язок із процесором (через Front Side Bus – FSB); оперативна пам'ять (RAM); відеокарта (через AGP або PCIe). Працював на високій частоті, безпосередньо взаємодіючи з CPU.

Південний міст (Southbridge) обробляв повільніші пристрої: жорсткі диски (через IDE, SATA), USB, COM, LPT порти, аудіо, мережу, клавіатуру/мишу, PCI-слоти, CMOS, BIOS.

Сучасна модель. У сучасних системах архітектура змінена. Функції північного моста перенесено в сам процесор (CPU). Залишився лише один чип – Platform Controller Hub (PCH), який виконує роль сучасного «південного моста».

Наприклад, у платах з чипсетами Intel Z790 або AMD X670E, відео, пам'ять і PCIe x16 напряму підключаються до CPU, а решта пристроїв – до чипсета.

Сучасні материнські плати мають розвинену підсистему живлення (VRM), що відповідає за стабільне постачання напруги до процесора та пам'яті, особливо важливо при розгоні. До плати також підключаються накопичувачі через інтерфейси SATA або M.2 (NVMe), периферійні пристрої через USB, а для виведення відео – HDMI/DisplayPort, якщо є вбудоване графічне ядро.

Отож, сучасні материнки оснащуються слотами PCI-E x16, PCI-E x8, PCI-E x4, PCI-E x1. Кожен призначений для підключення певних комплектуючих. Зазвичай, для дискретної відеокарти відводитися верхній PCI-E x16. Що важливо: після підключення відеокарти, її система охолодження нерідко перекриває доступ до наступного PCI-слота або навіть оперативної пам'яті – зверніть увагу на розташування слота для відеокарти на фотографіях обраної материнки і розташування інших слотів стосовно нього. Фактор вільного простору дуже важливий, особливо при малих форм-факторах.

Порядок виконання роботи:

Завдання виконуються на стендових материнських платах.

1. Вказати назву та модель материнської плати, яка досліджується.

2. Визначити тип сокета та підтримувані процесори (наприклад, LGA1700 для Intel або AM5 для AMD).

3. Вказати форм-фактор плати (ATX, microATX) та обгрунтувати це.

4. Знайти слоти для оперативної пам’яті: кількість, стандарт (тип DDR), максимально підтримуваний обсяг.

5. Ідентифікувати головний слот PCI Express для відеокарти, вказати його стандарт (модифікацію PCIe).

6. Вказати слоти M.2 для SSD, вказати кількість та тип підтримуваних накопичувачів.

7. Визначити інтерфейси підключення живлення: основний 24-pin, CPU живлення (8+4 pin).

8. Вказати чипсет, встановлений на материнській платі, та коротко описати його можливості.

9. Показати елементи охолодження та VRM (система живлення процесора), вказати кількість фаз живлення.

10. Знайти задню панель введення/виведення і перерахувати доступні порти (USB, LAN, HDMI, аудіо тощо).

Контрольні питання:

1. Що таке материнська плата і яку роль вона виконує в комп’ютері?

2. Що таке чипсет і яке призначення його основних компонентів?

3. Яка основна функція чипсета, і як він впливає на функціональні можливості системи?

4. Яка різниця між DDR3, DDR4 і DDR5 оперативною пам’яттю? Який тип підтримують сучасні плати?

5. Що таке PCI Express, які версії існують та для чого використовується кожен канальний тип слота (x1, x4, x16)?

6. Які форм-фактори материнських плат існують і чим вони відрізняються між собою?

Література: [1-6, 10, 14]

Лабораторна робота №2. Дослідження основних шин комп'ютера

Мета роботи: вивчити характеристики шин PCIe, SATA, USB

Теоретичні відомості

У комп'ютерній архітектурі шина – це сукупність ліній зв'язку, які передають дані, адреси та сигнали керування між компонентами системи. У сучасних комп'ютерах основними типами шин є PCI Express (PCIe), SATA та USB, кожна з яких виконує специфічні функції та має свої характеристики пропускної здатності.

PCI Express (PCIe) – це високошвидкісна шина для підключення відеокарт, NVMe SSD, мережевих адаптерів та інших плат розширення. Вона використовує послідовну точка-точка архітектуру. Основні версії: PCIe 3.0 (до 1 ГБ/с на лінію), PCIe 4.0 (2 ГБ/с), PCIe 5.0 (до 4 ГБ/с), PCIe 6.0 (до 8 ГБ/с), а в майбутньому – PCIe 7.0. Наприклад, слот PCIe x16 версії 5.0 може забезпечити пропускну здатність до 64 ГБ/с.

SATA (Serial ATA) – це шина для підключення накопичувачів типу HDD, SSD (2.5"), оптичних приводів. Найпоширеніша версія – SATA III (SATA 6 Гбіт/с), що забезпечує реальну пропускну здатність до ~550 МБ/с. SATA є поступово застаріваючим інтерфейсом, який витісняється NVMe-пристроями на базі PCIe.

USB (Universal Serial Bus) – універсальна шина для підключення периферійних пристроїв (клавіатура, миша, флешка, принтер, камера тощо). USB має кілька поколінь: USB 2.0 (до 480 Мбіт/с), USB 3.2 Gen1 (до 5 Гбіт/с), Gen2 (до 10 Гбіт/с), Gen2x2 (до 20 Гбіт/с), USB4 (до 40 Гбіт/с). Новітні версії також передають відео та живлення, що дозволяє використовувати USB Type-C як єдиний кабель.

Кожна з шин має свою архітектуру, топологію та призначення. Наприклад, PCIe підключається безпосередньо до процесора або чипсета, SATA – через контролери чипсета, USB – як частина вводу/виводу. Для визначення типів шин та їх пропускної здатності на конкретній платі використовують документацію виробника або утиліти, такі як HWiNFO, AIDA64 або CPU-Z.

Порядок виконання роботи:

Завдання виконуються на стендових материнських платах.

1. Оберіть сучасну материнську плату (наприклад, ASUS Z790-A, MSI B650 Tomahawk, ASRock X670E Steel Legend) і визначте, які типи шин реалізовані на платі: PCIe, SATA, USB.

2. Знайдіть та зафіксуйте:

- кількість слотів PCIe, їхні версії та формати (x1, x4, x16);
- кількість SATA-портів та версію SATA;
- типи USB-портів (2.0, 3.2 Gen1/Gen2, USB-C, USB4) – як на платі, так і на задній панелі.

3. Обчисліть пропускну здатність кожної з шин.

4. Побудуйте таблицю «Назва шини – Призначення – Версія – Пропускна здатність».

5. Зробіть висновки щодо можливостей розширення системи.

Контрольні питання:

1. Що таке шина комп’ютера? Які функції вона виконує?
2. Чим PCIe відрізняється від старих шин типу PCI або AGP?
3. Які версії PCIe існують і чим вони відрізняються між собою?
4. Що таке конфігурація ліній PCIe (x1, x4, x8, x16) і як вона впливає на пропускну здатність?
5. Для чого використовується шина SATA і яка її максимальна швидкість?
6. Які обмеження має SATA SSD порівняно з NVMe SSD?
7. У чому різниця між USB 2.0, 3.2 Gen1, Gen2, Gen2x2 та USB4?
8. Яка реальна швидкість передачі даних по USB 3.2 Gen2?
9. Як визначити тип шини та її версію на материнській платі?
10. Які шини передають не тільки дані, але й живлення?

Література: [1-6, 12-15]

Лабораторна робота №3. Організація оперативної пам’яті

Мета роботи: дослідити структуру та принцип роботи ОЗП

Теоретичні відомості

Оперативна пам’ять (ОЗП, RAM) – це енергозалежна пам’ять, яка тимчасово зберігає дані та команди, необхідні процесору для виконання поточних завдань. Найпоширеніший тип в сучасних ПК – DRAM (Dynamic RAM), зокрема DDR (Double Data Rate) пам’ять. Сьогодні використовуються стандарти DDR4 та DDR5, які відрізняються тактовою частотою, пропускною здатністю, енергоспоживанням та внутрішньою архітектурою.

Кожен модуль ОЗП має SPD (Serial Presence Detect) – спеціальний мікрочип, в якому зберігається службова інформація про параметри модуля: назва виробника, тип пам’яті, об’єм, частота, таймінги, напруга, XMP-профілі тощо. Програмне забезпечення (наприклад, CPU-Z, HWiNFO, Thaiphoon Burner) дозволяє прочитати SPD-дані.

Частота та таймінги – це ключові характеристики швидкодії пам’яті. Частота (наприклад, 3200 МГц, 5600 МГц) визначає кількість переданих бітів за секунду, а таймінги (наприклад, CL16-18-18-36) – затримки при доступі до даних. Чим нижчі таймінги, тим швидше модуль реагує на запити.

Канальність пам’яті – ще один важливий параметр. Системи можуть підтримувати одно-, дво- або чотириканальний режим, що прямо впливає на ефективну пропускну здатність. Наприклад, при двоканальній конфігурації два модулі працюють паралельно, що підвищує продуктивність до 50–70%.

Оцінити реальну швидкодію ОЗП можна за допомогою бенчмарків, зокрема AIDA64, PassMark, UserBenchmark тощо. Вони вимірюють пропускну здатність (MB/s) і латентність (затримку доступу в наносекундах).

Порядок виконання роботи:

1. Встановити утиліти CPU-Z та AIDA64 (або аналогічні).
2. У CPU-Z перейти у вкладку «Memory» та «SPD», записати:
   - тип пам’яті (DDR4/DDR5);
   - об’єм модуля (у GB);

- частоту (DRAM Frequency ×2);
- таймінги (CL, tRCD, tRP, tRAS);
- виробника та серійний номер.

3. За допомогою AIDA64 або іншого бенчмарка виконати:
   - тест пропускної здатності пам’яті (читання/запис/копіювання);
   - вимірювання латентності.
4. Зробити скріншоти результатів та порівняти з номінальними характеристиками пам’яті.

5. Якщо доступно – змінити XMP-профіль у BIOS/UEFI і повторити тестування.

6. Побудувати таблицю: Параметр – Значення до – Значення після (з XMP)

Контрольні питання:

1. Яке призначення оперативної пам’яті у комп’ютері?
2. Які типи оперативної пам’яті використовуються в сучасних ПК?
3. Що таке SPD-інформація та як її можна переглянути?
4. Які основні характеристики містить SPD?
5. Що таке таймінги пам’яті? Як вони впливають на продуктивність?
6. Що таке XMP-профіль і як він використовується?
7. Що означає двоканальний або чотириканальний режим роботи пам’яті?
8. Як вимірюється пропускна здатність пам’яті?
9. Що таке латентність пам’яті і як її знизити?
10. Як підвищення частоти ОЗП впливає на загальну продуктивність системи?

Література: [1-2, 4, 7-9, 13]

Лабораторна робота №4. Структура кеш-пам’яті

Мета роботи: вивчити рівні кешу L1, L2, L3 і їх роль в обчислювальних можливостях процесорів

Теоретичні відомості

Кеш-пам’ять (cache memory) – це надшвидкий буфер між процесором і оперативною пам’яттю. Вона зберігає часто використовувані дані, щоб процесор міг звертатись до них без затримки, не чекаючи доступу до повільнішої оперативної пам’яті.

Сучасні процесори мають багаторівневу кеш-пам’ять.

L1 (Level 1) – це найменший, але найшвидший кеш. Зазвичай поділяється на інструкційний та кеш даних. Розмір – 16–128 КБ на ядро, затримка – 1-4 такти.

L2 (Level 2) – більший і повільніший, ніж L1. Розмір – 256 КБ – 1 МБ на ядро. Має нижчу пріоритетність, але зберігає більше інформації.

L3 (Level 3) – найбільший кеш, спільний для кількох або всіх ядер. Розмір – від 2 до 96 МБ. Знаходиться далі від ядра, але значно швидший за ОЗП.

Деякі сучасні процесори (наприклад, AMD Ryzen 7000 з технологією 3D V-Cache) мають додатковий **кеш-пул**, що істотно підвищує продуктивність в іграх і обчисленнях.

Попадання в кеш (cache hit) виникає тоді, коли процесор знаходить необхідні дані в одному з рівнів кеш-пам’яті (L1, L2 або L3), не звертаючись до повільнішої оперативної пам’яті. Це дозволяє значно прискорити виконання інструкцій, оскільки кеш працює в десятки разів швидше, ніж ОЗП. Чим вища ймовірність попадання в кеш, тим ефективніше використовуються ресурси процесора.

Промах у кеші (cache miss) трапляється, коли шуканих даних немає в кеші. У такому випадку процесор змушений звертатися до наступного рівня кешу або до основної пам’яті, що спричиняє затримку. Промахи в L1 частково компенсуються доступом до L2 і L3, але при промаху на всіх рівнях система втрачає час через повільнішу взаємодію з ОЗП. Це знижує загальну продуктивність, особливо в обчисленнях з випадковим або великим обсягом

даних. Чим ближчий кеш до ядра і чим менша його затримка, тим він ефективніший. У разі промаху в L1 дані шукаються в L2, потім у L3, і лише потім – в оперативній пам’яті. Чим менше таких «промахів» (cache misses), тим краща продуктивність системи.

Оновлення кешу відбувається автоматично за допомогою спеціальних алгоритмів заміщення (наприклад, LRU – least recently used), які визначають, які дані варто зберігати, а які замінити. При промаху нові дані завантажуються з оперативної пам’яті у кеш, і залежно від стратегії (write-through або write-back), кеш може відразу оновити основну пам’ять або зробити це пізніше. Така динамічна система дозволяє кешу адаптуватися до робочих навантажень і зберігати найбільш актуальні дані для прискорення доступу.

Порядок виконання роботи:

1. Запустіть утиліту CPU-Z, відкрийте вкладку «Cache»:
   - зафіксуйте розміри кешів L1, L2, L3;
   - зверніть увагу, який кеш поділений, а який – індивідуальний для ядер.
2. У AIDA64:
   - перейдіть у розділ «Центральний процесор» → «Кеш-пам’ять»;
   - перевірте кількість асоціативності (ways), розмір рядка кешу (cache line), типи кешів.
3. Проведіть **бенчмарк кешу** (AIDA64 → «Кеш і пам’ять»):
   - збережіть результати латентності (затримки) для L1, L2, L3;
   - порівняйте з показниками аналогічних процесорів (за базою CPU Benchmark або офіційним сайтом Intel/AMD).
4. Зробіть скріншоти таблиці кешів та графіка бенчмарку.
5. Складіть власну таблицю порівняння кешів.

Контрольні питання:

1. Яке призначення кеш-пам’яті в сучасних процесорах?
2. У чому полягає різниця між кешами L1, L2 та L3?
3. Чому кеш L1 має найменший розмір, але найвищу швидкість?
4. Що таке асоціативність кешу (наприклад, 8-way set associative)?

5. Як розподіляється кеш між ядрами в сучасних процесорах?
6. Які особливості кешування мають багатоядерні процесори?
7. Що таке “промах кешу” (cache miss) і як він впливає на продуктивність?
8. Як можна дізнатися характеристики кешу свого процесора?
9. Який рівень кешу найбільше впливає на швидкість виконання інструкцій?
10. Як технологія 3D V-Cache від AMD впливає на продуктивність?

Література: [1-5, 8-11]

Лабораторна робота №5. Робота блоку живлення комп’ютера

Мета роботи: ознайомитися з принципами роботи та видами БЖ

Теоретичні відомості

Блок живлення комп’ютера (БЖ) – це пристрій, який перетворює змінну напругу мережі 220 В у стабілізовану постійну напругу, необхідну для живлення внутрішніх компонентів ПК: материнської плати, процесора, накопичувачів, відеокарти тощо. Найпоширеніший стандарт – ATX, хоча існують також SFX, TFX, Flex ATX та інші форм-фактори.

Сучасні блоки живлення є імпульсними перетворювачами, які використовують високочастотні транзистори для підвищення ККД (до 80–95%) та зменшення розмірів трансформаторів. Напруги, які зазвичай видає БЖ: +12 В, +5 В, +3.3 В, а також – 12 В. На виході використовуються електролітичні та твердотільні конденсатори, дроселі та стабілізатори для згладжування пульсацій і стабілізації напруги.

Структура сучасного блоку живлення (для десктопних систем) подана на рисунку 5.1:

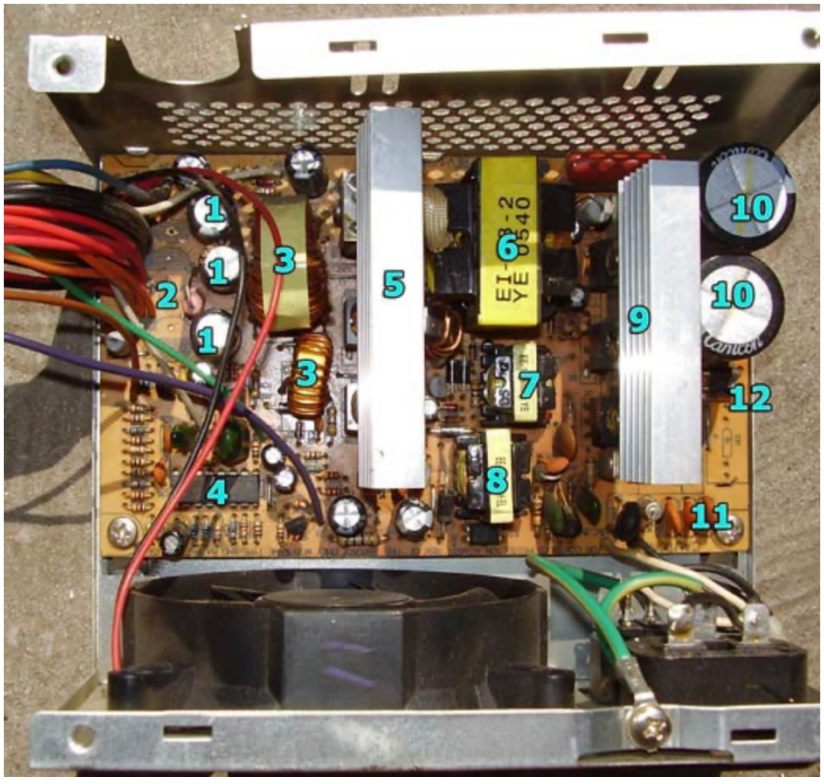


Рисунок 5.1 – Блок живлення ATX [2]

На рисунку показані:

1. Конденсатори, що виконують фільтрацію вихідних напруг.
2. Місце не розпаяних конденсаторів фільтра вихідних напруг.
3. Катушки індуктивності, що виконують фільтрацію вихідних напруг. Велика котушка відіграє роль не тільки фільтра, але ще працює як феромагнітний стабілізатор. Це дозволяє дещо знизити перекоси напруг при нерівномірному навантаженні різних вихідних напруг.
4. Мікросхема ШІМ-стабілізатора WT7520.
5. Радіатор на якому встановлені діоди Шотткі для напруг +3.3В і +5В, а для напруги +12В звичайні діоди. Необхідно відзначити, що часто особливо в старих блоках живленнях, на цьому ж радіаторі розміщаються додатково елементи. Це елементи стабілізації напруг +5В и +3,3В. У сучасних блоках живленнях розміщуються на цьому радіаторі тільки діоди Шотткі для всіх основних напруг або польові транзистори, які використовуються у якості випрямляча.
6. Основний трансформатор, що здійснює формування всіх напруг, а також гальванічну розв’язку з мережею.
7. Трансформатор, що формує керуючі напруги для вихідних транзисторів перетворювача.
8. Трансформатор перетворювача, що формує чергову напругу +5В.
9. Радіатор, на якому розміщені вихідні транзистори перетворювача, а також транзистор перетворювача формує чергову напругу.
10. Конденсатори фільтра мережної напруги. Їх не обов’язково повинно бути два. Для формування двополярної напруги та утворення середньої крапки встановлюють два конденсатори рівної ємності. Вони ділять випрямлену сіткову напругу навпіл, тим самим формуючи дві напруги різної полярності, з’єднані у загальній крапці. В схемах з однополярним живленням конденсатор один.
11. Елементи фільтра мережі від гармонік (перешкод), що генеруються блоком живлення.
12. Діоди діодного мосту, що здійснюють випрямлення змінної напруги мережі.

Якість та потужність блоку живлення оцінюється за такими параметрами: сумарна потужність (W), максимальний струм по кожній лінії (A), ефективність (сертифікати 80 PLUS), тип охолодження, наявність захистів (OVP, SCP, OCP, OTP тощо). Правильний розрахунок і вибір БЖ дозволяє забезпечити стабільну роботу комп'ютера та уникнути перевантажень чи виходу з ладу компонентів.

Оцінка ємності вхідних і вихідних конденсаторів допомагає визначити якість фільтрації пульсацій. Вона проводиться або за маркуванням (μF, V), або через вимірювання приладів. Також важливо перевірити розведення живлення для CPU (4/8 pin), GPU (6/8 pin), SATA, Molex.

Порядок виконання роботи:

1. Ознайомитися з реальним або демонстраційним блоком живлення (стендовим).
2. Візуально оцінити внутрішню структуру і визначити розташування основних компонентів:
   - трансформатор;
   - мости випрямлення;
   - радіатори;
   - високовольтні та низьковольтні конденсатори;
   - силові ключі (транзистори);
   - система охолодження.
3. Знайти маркування на електролітичних конденсаторах і визначити їх ємність і робочу напругу.
4. Розрахувати загальну ємність вихідного каскаду (за потреби – сумарно по каналах +12 В, +5 В).
5. Визначити кількість ліній живлення, типи роз'ємів та їх призначення.
6. Порівняти параметри з вимогами реального ПК (CPU + GPU + інші компоненти). Оцінити похибку вимірювання потужності.
7. Прикласти розрахунок потужності системи (наприклад, на основі калькулятора PSU).
8. Зробити висновки щодо придатності БЖ для сучасної системи.

Контрольні питання:

1. Яка основна функція блоку живлення комп’ютера?
2. Що таке імпульсний блок живлення і чим він відрізняється від лінійного?
3. Які напруги подає БЖ на материнську плату та інші пристрої?
4. Які форм-фактори блоків живлення існують?
5. Що означає сертифікат 80 PLUS і які є його рівні (Bronze, Gold тощо)?
6. Які види захисту передбачені в сучасних БЖ?
7. Яку роль виконують конденсатори у блоці живлення?
8. Як визначити загальну потужність БЖ і максимальне навантаження по лінії +12 В?
9. Чим небезпечне використання дешевих або несертифікованих БЖ?
10. Як здійснити підбір потужності блоку живлення для реальної конфігурації ПК?

Література: [1-3, 5-7, 10]

Лабораторна робота №6. Робота з BIOS/UEFI

Мета роботи: вивчити особливості налаштування BIOS/UEFI

Теоретичні відомості

BIOS (Basic Input/Output System) – це базова система вводу-виводу, що виконується першою після ввімкнення комп'ютера. Вона перевіряє основні апаратні компоненти (через POST – Power-On Self Test), а потім передає керування операційній системі.

UEFI (Unified Extensible Firmware Interface) – це сучасніша версія BIOS, з розширеним графічним інтерфейсом, підтримкою великих дисків (більше 2 ТБ), швидшого завантаження та розширених функцій безпеки (Secure Boot).

Основні розділи BIOS/UEFI:

- Boot – вибір черговості завантаження (USB, SSD, HDD, CD/DVD тощо)
- Advanced/AI Tweaker – налаштування процесора, пам'яті, оверклокінг
- Security – встановлення пароля BIOS, увімкнення Secure Boot
- Monitoring/Hardware – контроль температури, напруг, швидкості вентиляторів
- Exit – вихід із налаштувань, збереження або скасування змін.

У процесі адміністрування комп'ютерних систем та налагодження завантаження операційної системи важливо розуміти, яка саме прошивка використовується: традиційний BIOS чи сучасніший UEFI. Ці два режими мають різні можливості, інтерфейси та вимоги до завантаження ОС, зокрема щодо таблиць розділів дисків (MBR vs GPT) та підтримки функцій безпеки, таких як Secure Boot. Визначення типу прошивки дозволяє точніше налаштувати систему, забезпечити сумісність при встановленні ОС або виконанні відновлення системи.

Залежно від операційної системи, тип BIOS або UEFI можна визначити за допомогою відповідних команд у терміналі або консолі.

Щоб визначити тип BIOS/UEFI та отримати інформацію про версію прошивки на різних операційних системах, використовують різні команди (таблиця 6.1).

Таблиця 6.1 – Визначення типу базової системи вводу-виводу для ОС [3]

Операційна система	Команди перевірки, чи це BIOS, чи UEFI	Версія BIOS
Windows	PowerShell: *Get-WmiObject -Class Win32_BootConfiguration,* або *Confirm-SecureBootUEFI* Якщо команда Confirm-SecureBootUEFI повертає True – система завантажена через UEFI.	PowerShell: *Get-WmiObject win32_bios | Format-List* *wmic bios get smbiosbiosversion*
Linux (Ubuntu, Debian, etc.)	Bash: [-d /sys/firmware/efi] && echo «UEFI»|| echo «BIOS»	Bash: *sudo apt install dmidecode* *sudo dmidecode -t bios*
macOS	macOS використовує лише UEFI, тому прямої потреби визначати тип немає. Але можна переглянути інформацію про завантаження: Bash: *system_profiler SPHardwareDataType*	

Знання та вміння перевіряти тип прошивки є необхідними навичками для будь-якого фахівця, що працює з комп’ютерним обладнанням або операційними системами. У Windows для цього можна використати команди PowerShell або WMI, тоді як у Linux перевірка здійснюється через файлову систему /sys/firmware/efi, яка існує лише у випадку UEFI. У macOS питання типу завантаження практично не стоїть, оскільки всі сучасні пристрої Apple використовують UEFI.

Порядок виконання роботи:

1. Вхід до BIOS/UEFI. Перезавантажте комп’ютер або ввімкніть його. Під час запуску натискайте клавішу (залежить від виробника): Delete, F2, F10, ESC, F12. Зайдіть у BIOS/UEFI.

2. Зміна пріоритету завантаження: перейдіть у вкладку **Boot**, змініть порядок завантаження, встановивши, наприклад, **USB** першим, збережіть зміни (F10) і перезавантажте комп’ютер.

3. Діагностика компонентів. Перейдіть у вкладку Hardware Monitor або PC Health Status. Перегляньте: температуру процесора, напругу на CPU та RAM, оберти вентиляторів.

4. Оверклокінг (не зберігати вказані налаштування!). У вкладці AI Tweaker або Advanced знайдіть налаштування CPU Clock. Трохи змініть частоту (наприклад, з 3.2 GHz на 3.4 GHz).

5. Захист BIOS. У розділі Security встановіть пароль для входу до BIOS. Увімкніть Secure Boot, якщо планується використання Windows 10/11.

Контрольні питання:

1. Чим відрізняється BIOS від UEFI?
2. Які функції виконує POST?
3. Як змінити порядок завантаження пристроїв у BIOS?
4. Що таке Secure Boot і яка його мета?
5. Які ризики пов’язані з оверклокінгом?
6. Як переглянути температуру процесора в UEFI?
7. Чому важливо оновлювати прошивку BIOS?
8. Як встановити пароль на вхід у BIOS?
9. Який розділ BIOS/UEFI використовується для зміни параметрів CPU?
10. Як зберігаються зміни після налаштування BIOS?

Література: [1-4, 6-9, 11, 15]

Лабораторна робота №7. Системний таймер і годинник реального часу

Мета роботи: дослідити принцип роботи таймерів

Теоретичні відомості

У комп’ютері використовуються два основні типи таймерів: RTC (Real-Time Clock) та системний таймер. RTC– це апаратний модуль на материнській платі, який відліковує час навіть тоді, коли комп’ютер вимкнений. RTC зберігає дату та час за допомогою батарейки CMOS. Системний таймер (або PIT – Programmable Interval Timer) – це таймер, який використовується операційною системою для керування процесами, вимірювання часу виконання, керування подіями, затримками тощо. У сучасних системах його функції можуть виконуватись через HPET або TSC [4].

RTC представляє собою спеціалізований мікросхемний модуль, вбудований у материнську плату, який відповідає за безперервне відлічування часу. Його головною функцією є підтримка точної дати й часу незалежно від того, чи увімкнений комп’ютер, чи ні. Це стає можливим завдяки автономному джерелу живлення – батарейці CMOS, що підтримує роботу RTC у фоновому режимі навіть при повному вимкненні системи. Завдяки цьому користувач не втрачає налаштування часу між перезавантаженнями або при тривалому вимкненні комп’ютера.

Точність RTC зазвичай забезпечується за допомогою кварцового генератора, який формує стабільну частоту, необхідну для вимірювання часу. RTC працює незалежно від центрального процесора і операційної системи, що дає змогу точно синхронізувати системні події під час завантаження. Під час запуску комп’ютера BIOS або UEFI звертається до RTC, щоб зчитати актуальний час і передати його операційній системі. Надалі ОС використовує ці значення для журналювання подій, роботи планувальника завдань, цифрового підпису тощо.

Мікросхема RTC містить не тільки лічильник часу, але й невеликий обсяг енергонезалежної пам’яті CMOS, де можуть зберігатися базові налаштування BIOS (наприклад, порядок завантаження, конфігурація обладнання тощо). У випадку розрядження батарейки CMOS ці дані можуть бути втрачені, що призведе

до скидання налаштувань BIOS і необхідності повторного налаштування дати й часу.

У сучасних комп'ютерах, особливо ноутбуках і серверних платформах, RTC є невід'ємною частиною системного контролера (наприклад, чіпсета Intel PCH або AMD FCH) і може підтримувати синхронізацію з зовнішніми джерелами часу (через інтернет або мережу) після запуску ОС. Це особливо важливо для забезпечення точності таймштампів у критичних сферах – від бухгалтерії до системного моніторингу та безпеки.

Системний таймер і RTC взаємодіють між собою під час завантаження: BIOS/UEFI зчитує дані з RTC і передає ОС базове значення часу. Після запуску ОС переходить до використання власного механізму обліку часу.

Ось приклад простої Bash-програми, яка заміряє проміжок часу між двома довільними подіями, наприклад, початком і завершенням виконання певної команди, як-от копіювання файлу, або виконання циклу. Програма використовує вбудовану команду date +%s%N для отримання часу з точністю до наносекунд:

```
#!/bin/bash

echo "Початок вимірювання часу..."
start_time=$(date +%s%N)  # час у наносекундах

# Подія №1: наприклад, обробка циклу
for i in {1..100000}; do
  : # пуста команда, що не виконує дій
done

end_time=$(date +%s%N)  # час у наносекундах
elapsed=$((end_time - start_time))

# Переводимо наносекунди в мілісекунди
ms=$((elapsed / 1000000))

echo "Завершено. Пройшло часу: $ms мс"

```

Результат виконання програми:

```
$ bash timer.sh
Початок вимірювання часу...
Завершено. Пройшло часу: 7 мс

```

Порядок виконання роботи:

1. Після завантаження операційної системи перевірте системний час. У Windows це можна зробити за допомогою командного рядка, ввівши команди time та date, які відображають відповідно поточний час і дату. У Linux слід відкрити термінал і скористатися командами date для виведення поточного часу та timedatectl для перегляду розширеної інформації про часові параметри системи.

2. Дослідити роботу системного таймера. У Windows це можна зробити за допомогою PowerShell, використовуючи команду Get-Date для отримання поточного часу, а також Measure-Command { Start-Sleep -Seconds 3 } – для вимірювання затримки виконання певної дії. У Linux для спостереження за часом у реальному часі застосовується команда watch -n 1 date, яка виводить поточний час щосекунди. Додатково можна переглянути інформацію про таймери ядра командою cat /proc/timer_list.

3. Виконайте тестування затримки виконання. Напишіть просту програму (наприклад, на Python, Bash або PowerShell), яка заміряє проміжки часу між двома подіями – наприклад, запуском програми та її завершенням, або між натисканнями клавіш. Це дозволяє побачити, як системні таймери працюють на практиці та яку точність забезпечують.

Контрольні питання:

1. Що таке RTC і яка його роль у комп’ютері?
2. Яке джерело живлення використовує RTC і чому?
3. Що таке системний таймер, і як він використовується ОС?
4. Як перевірити час у BIOS/UEFI?
5. Як ОС визначає поточний час під час запуску?

Література: [1-4, 7-9, 13]

Лабораторне заняття №8. Дебагінг апаратного забезпечення

Мета роботи: засвоїти основи реверс-інженірингу

Теоретичні відомості

В сучасній ІТ-галузі є ряд спеціального програмного забезпечення, що називають налагоджувачами або відладчиками. Серед широко відомих продуктів – Turbo Debugger (що застосовується для 32-бітних систем), а також OllyDbg для 64-бітних систем.

Це програмне забезпечення використовують і новачки, що вивчають асемблер, і фахівці для тестування ряду програм. Відладчики – це зручний та багатофункціональний інструмент для аналізу та модифікації скомпільованих у машинний код файлів. З OllyDbg можливий як злом програми, так і навпаки – удосконалення захисту від злому програмного забезпечення. Коли вихідний код недоступний, можна дизассемблювати та аналізувати програму «у процесі, на льоту». Реалізований у налагоджувачі аналізатор коду розпізнає процедури, константи, цикли, таблиці, текст та розгалуження.

Відладчик-дизассемблер OllyDbg розпізнає та розшифровує понад 2000 функцій мови С та Windows API [5]. Налагоджувач підтримує підключення додаткових плагінів, виконує евристичний аналіз стека, працює в режимі покрокового налагодження, підтримує hex, ASCII, Unicode, 16/32 бітні цілі числа та 32/64/80 бітні числа з плаваючою комою. Відладчик із простим і зрозумілим інтерфейсом, підсвічуванням коду, хорошими можливостями та з відкритим вихідним кодом. Завантажити його можна за посиланням: **https://www.x64bitdownload.com/download/t-64-bit-ollydbg-download-egyyuouy.html**

Насамперед, необхідно запустити відладчик.

Для початку роботи реверс-інженірингу зручно використовувати bat-файли.

Приклад 1, створіть у каталозі одного з системних дисків свого ПК текстовий файл, назвіть його, наприклад, «debug.bat». У нього треба записати всього один рядок:

C:\TD\td.exe <файл_програми>.com

(Тобто, якщо файл з текстом програми називається prog.asm, то скомпільована програма буде називатися prog.com. В bat-файл треба написати: C:\TD\td.exe prog.com).

Приклад 2, можна створити файл .cpp за зразком:

```
#include <iostream.h>
#include <string.h>
#include <conio.h>
void main()
{
char *a,*pass="password";
clrscr();
cout<<"Enter the password: "<<endl;
cin>>a;
if(!strcmp(a,pass))
cout<<"OK";
else
cout<<"Password incorrect";
cin.get();
cin.get();
}
```

Нижче наведено фрагмент роботи відладчика Turbo Debugger після запуску у ньому виконавчого файлу. На рисунку представлено вигляд з рядком основного меню і областями редагування (налагодження) коду.

Після запуску виконавчого файлу у Turbo Debugger відкривається вікно з інтерфейсом, який складається з кількох основних областей, включаючи рядок головного меню, вікно коду, регістри та стан процесора. У верхній частині розташовано головне меню з пунктами File, Run, View, Break, Options, Window, Help, які дозволяють виконувати основні дії налагодження – запуск і зупинку програми, перегляд пам’яті, встановлення точок зупину, зміни регістрів та інші дії. Цей інтерфейс є характерним для текстових налагоджувачів у середовищі DOS і дозволяє зручно взаємодіяти з програмою на низькому рівні.

Основна область екрана відображає машинні інструкції та відповідний асемблерний код. Тут можна побачити адреси команд, коди байтів і розбір інструкцій, які виконує процесор. Крім того, вікно регістрів праворуч дозволяє

відстежувати значення регістрів загального призначення (AX, BX, CX тощо), покажчика інструкцій (IP), сегментних регістрів і прапорців. Turbo Debugger надає можливість покрокового виконання коду (Step Into, Step Over), що особливо корисно при пошуку помилок у низькорівневій логіці програми. Такий підхід до налагодження дозволяє глибоко аналізувати поведінку програми на рівні машинного коду, що критично важливо для системного програмного забезпечення або драйверів (рис. 8.1).

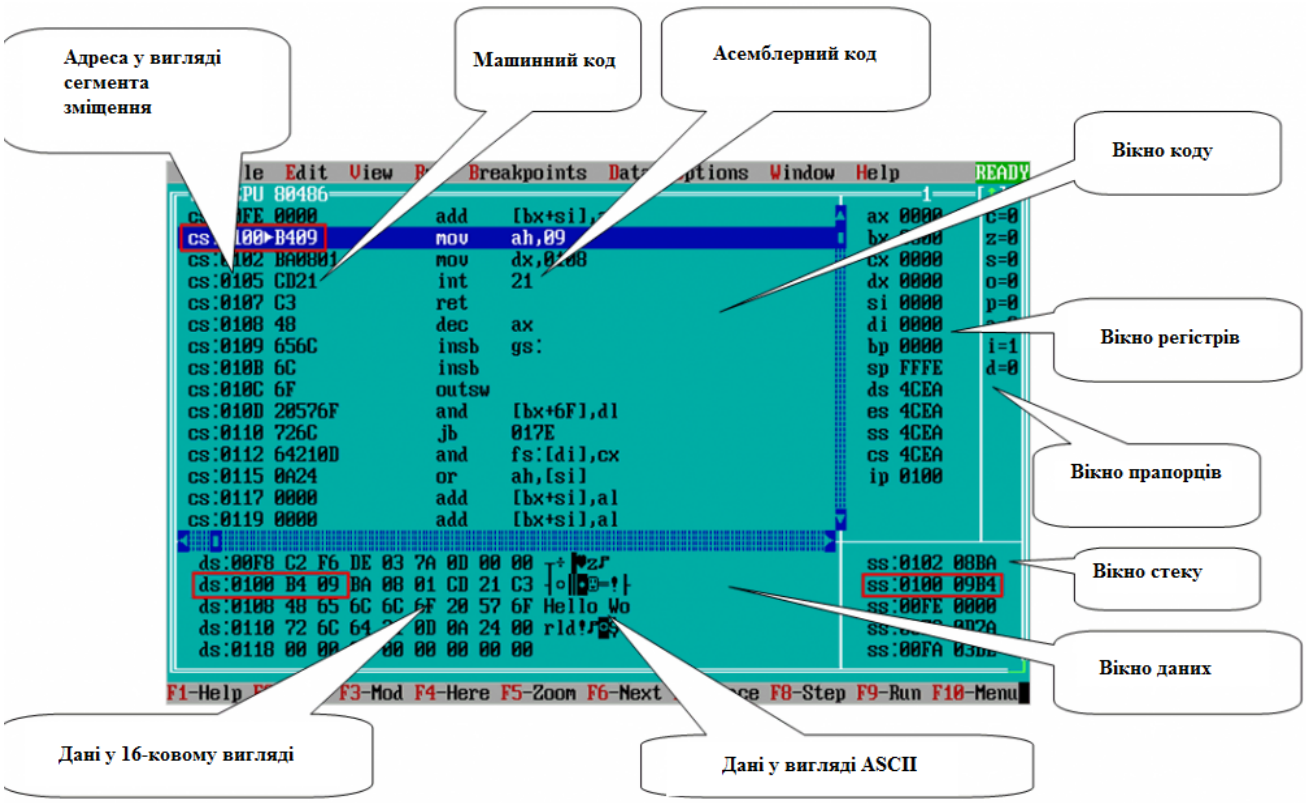


Рисунок 8.1 – Вікно відладчика

У якості експериментального робочого файлу у програмі *td.exe* можна запускати будь який виконавчий файл. Після запуску цього виконавчого файлу ви побачите приблизно таке вікно (рис. 8.2):

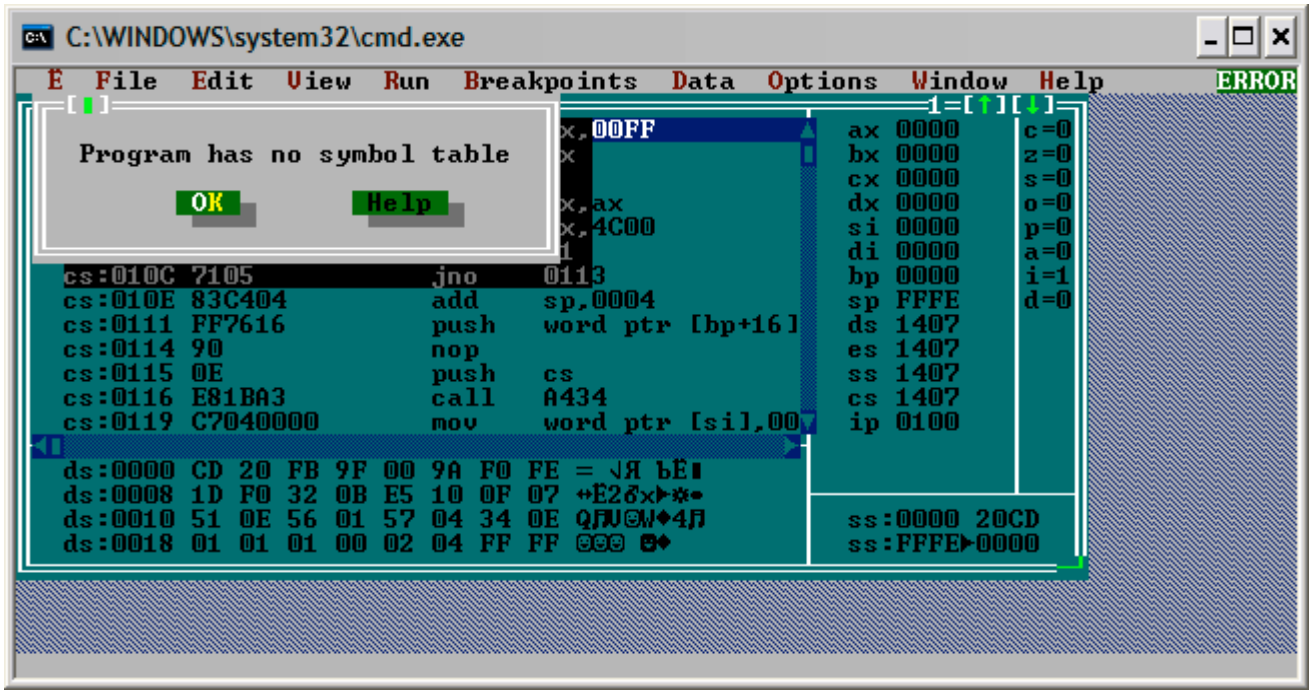


Рисунок 8.2 – Запуск програми у відладчику

Повідомлення означає, що у виконуваному файлі немає спеціальних даних для налагодження. Але нам ці дані не потрібні, тому що програма проста і зрозуміла. Натискаємо OK. Turbo Debugger відображає вікно CPU, в якому можна побачити, як виконується програма.

У наступному зображенні (OllyDbg) у великій області ми бачимо код нашої програми. Самий лівий стовпець – адреси, праворуч відображаються байти машинного коду, а ще правіше – символічне позначення команд.

Повідомлення про відсутність спеціальних даних для налагодження означає, що виконуваний файл не містить вбудованих символів, таких як імена змінних, функцій чи вихідного коду, які полегшують аналіз програми в налагоджувачі. Проте, якщо програма невелика або написана з мінімальною кількістю логіки, то навіть без цих символів можна ефективно стежити за її виконанням. Після підтвердження повідомлення кнопкою «OK», Turbo Debugger відкриває головне вікно CPU, у якому відображається поточний стан процесора, регістрів, сегментів пам’яті та виконуваний машинний код – усе необхідне для покрокового аналізу.

На прикладі іншого налагоджувача OllyDbg інтерфейс також орієнтований на аналіз машинного коду. Центральна частина вікна містить дизасембльований

код програми: зліва видно адреси команд, далі – послідовності байтів, які відповідають машинним інструкціям, а праворуч – символічний (асемблерний) еквівалент цих команд. Такий вигляд дозволяє легко аналізувати послідовність виконання інструкцій, стежити за переходами, викликами функцій та змінами в регістрах. OllyDbg автоматично виділяє активну інструкцію, дає змогу встановлювати точки зупину (breakpoints) та переглядати стан пам'яті в реальному часі, що робить його зручним інструментом для зворотного інжинірингу й налагодження низькорівневого коду (рис. 8.3).

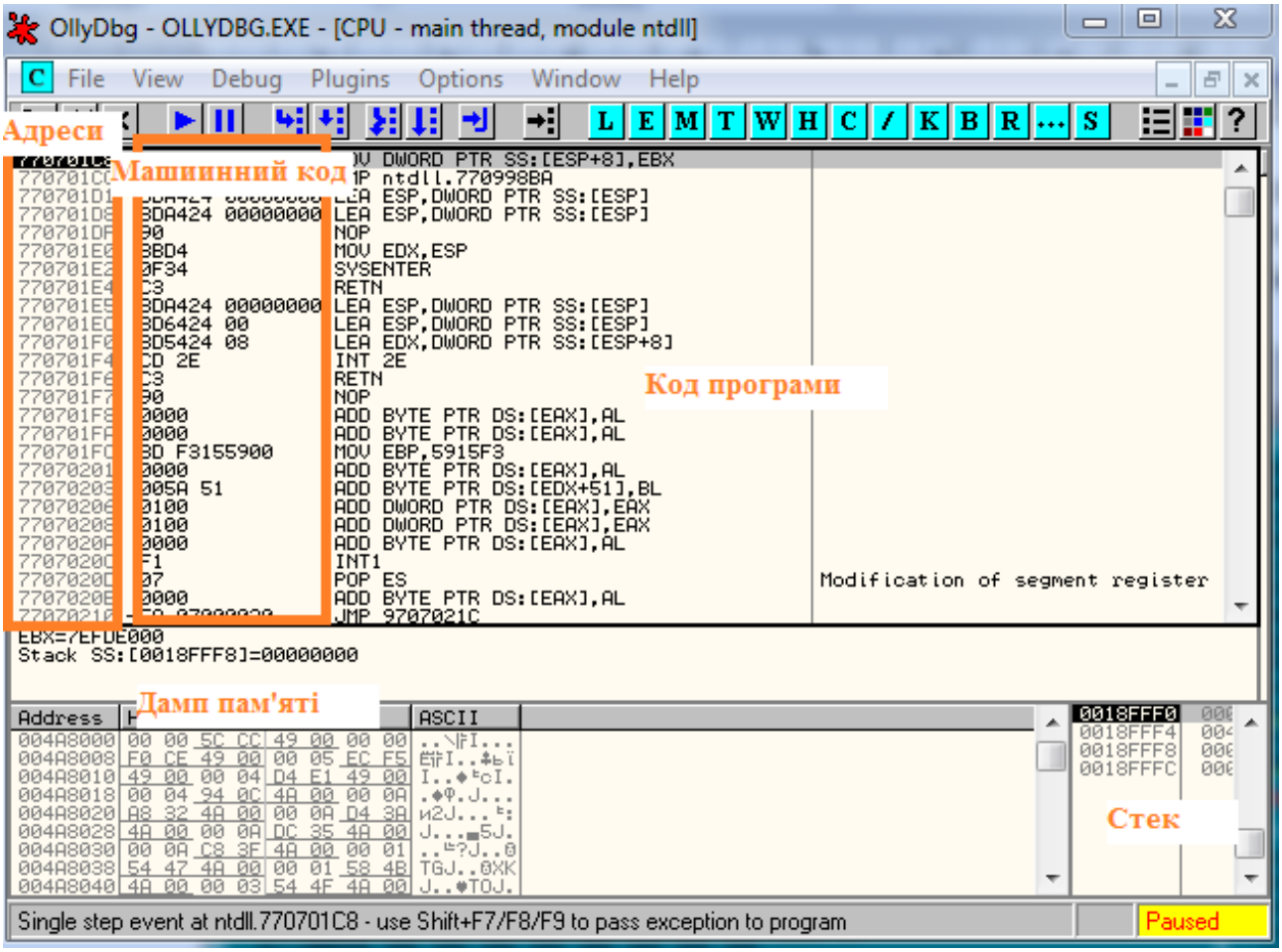


Рисунок 8.3 – Вікно дизасемблювання

У викликаній програмі *td.exe* (рисунок нижче) всього 6 машинних команд, а за ними в пам'яті знаходиться випадкове сміття (точні значення невідомі). Зверніть увагу, що відладчик показує адреси і значення в шістнадцятковому вигляді. У правій частині вікна CPU відображаються регістри процесора і

прапорці. У нижній частині можна побачити дамп пам'яті і стек. Стек – це спеціальна структура даних, з якою працюють деякі команди процесора. Адреса поточної машинної команди визначається регістрами CS і IP, ця команда показана виділеним рядком і стрілкою. Тепер натисніть F8, щоб виконати першу команду (рис. 8.4):

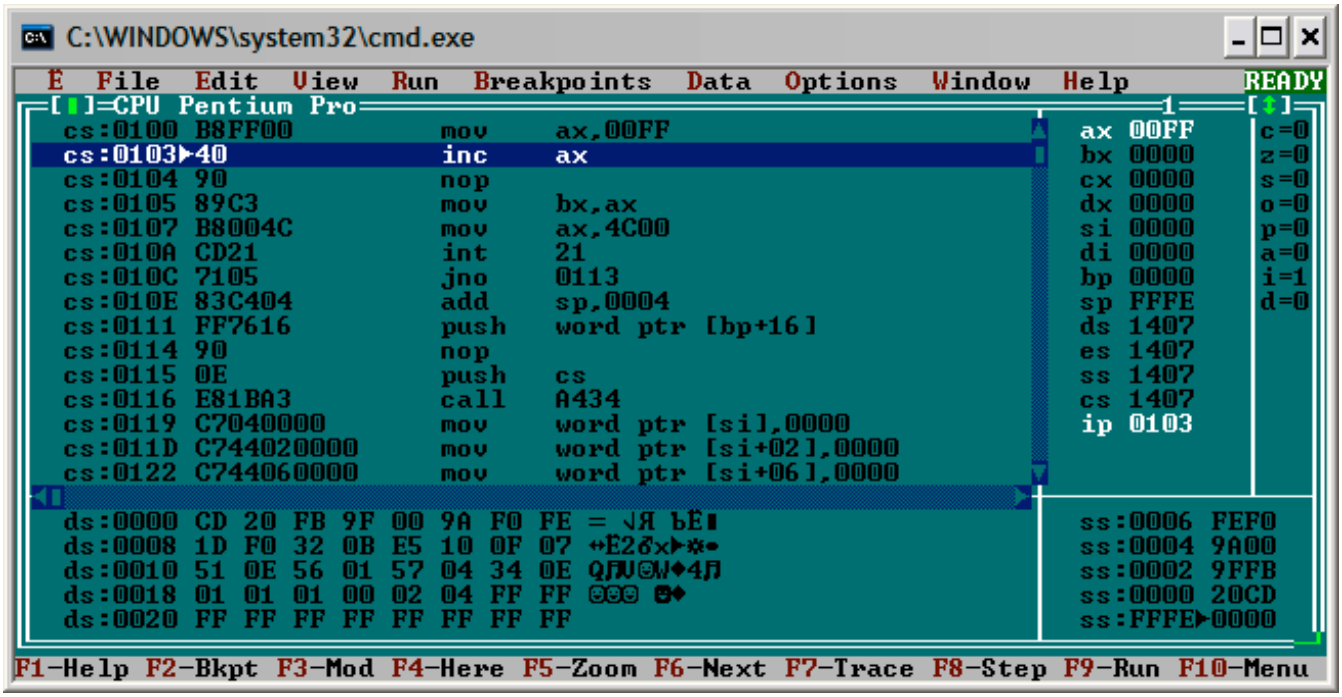


Рисунок 8.4 – Трасування в дебагері

Тепер стрілка вказує на другу команду. Змінені регістри виділені білим кольором. Регістр AX тепер містить значення 00FFh (тобто 255, чого ми хотіли від команди «mov ax,255»):

Також, змінилося значення регістра IP – воно збільшилося на розмір виконаної машинної команди, а саме на 3. Тепер 3S:IP вказує на таку команду. Знову натискаємо F8. Значення регістра AX збільшилася на 1 і стало рівним 0100h (256). Значення IP теж збільшилася на 1, тому що довжина команди «inc ax» – 1 байт. Процесор виконує програму послідовно, одну команду за одною.

Коли ви виконуєте першу команду, процесор прочитує інструкцію з поточної адреси (визначеної сегментом CS і регістром IP). Після її виконання IP автоматично збільшується на кількість байтів, яку займає ця команда – у цьому

випадку на 3 байти. Це означає, що наступна команда знаходиться на три байти далі у пам’яті. Далі, при натисканні F8, виконується інструкція inc ax, яка збільшує вміст регістра AX на 1. Це команда довжиною 1 байт, тож IP знову зміщується, але цього разу лише на 1. Така послідовність – типовий приклад того, як відладчик дозволяє аналізувати програму на рівні машинних інструкцій, показуючи зміни в регістрах і рух інструкцій крок за кроком (рис. 8.5).

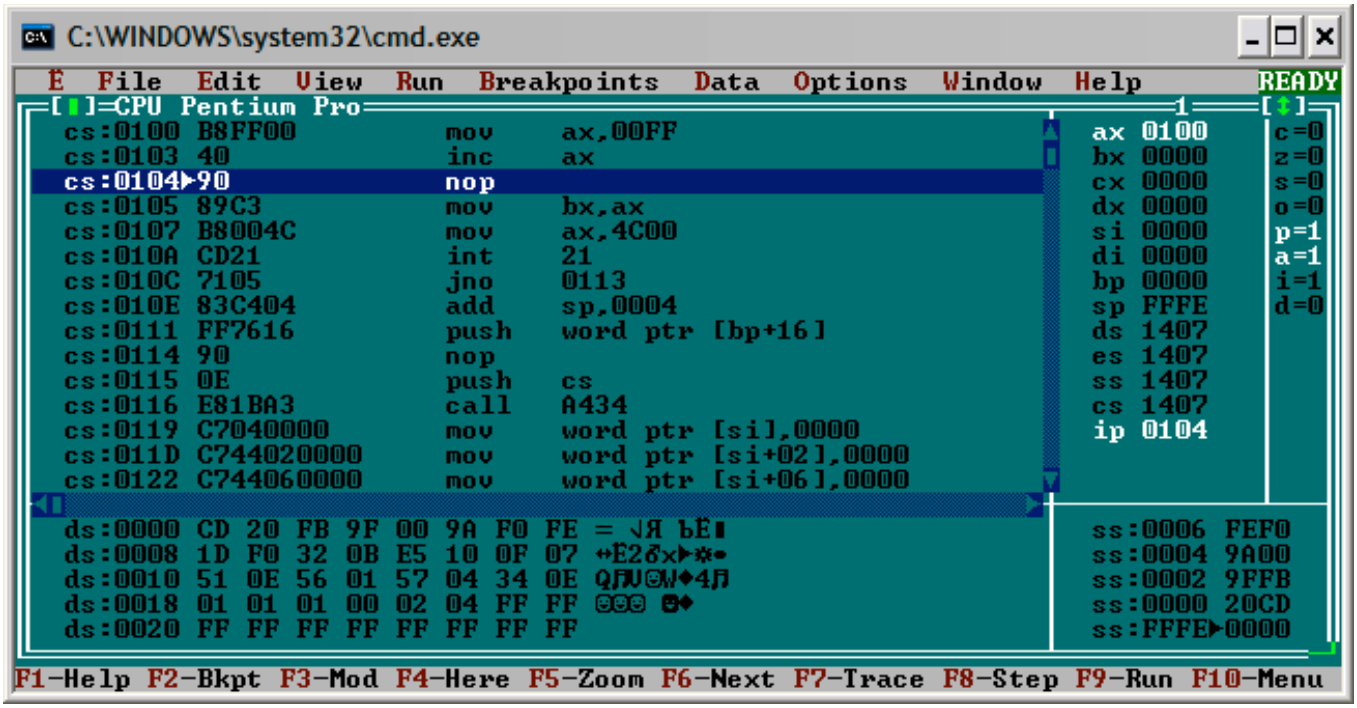


Рисунок 8.5 – Скрин роботи відладчика і зміни значень регістрів

Процес скінченний і після ще кількох натискань F8 програма завершується. Закрити відладчик можна з допомогою меню File – Quit.

Порядок виконання роботи:

1. Відкрити з допомогою відлагоджувача Debugger виконуваний файл
2. Оформити виконане завдання покроково, з поясненнями роботи CPU в процесі відлагодження.

1) Пояснити призначення кожного з 5ти вікон програми Debugger (що у цьому вікні вказується? Для чого?)

2) Виконати послідовне трасування програми

3) Описати поведінку регістрів процесора, значення машинних кодів, прапорців стану тощо.

Контрольні питання:

1. Що таке налагоджувач і яку роль він відіграє в процесі розробки програмного забезпечення?
2. Які основні функції надає Turbo Debugger для налагодження виконуваних програм?
3. Що таке регістр IP (Instruction Pointer), і яку роль він відіграє під час налагодження?
4. Як можна покроково виконувати програму в Turbo Debugger (які для цього використовуються команди)?
5. Як працює команда F8 в Turbo Debugger, і яке її призначення?
6. Яким чином можна побачити значення регістрів процесора під час виконання програми в Turbo Debugger?
7. Що таке breakpoint і як його встановити в налагоджувачі?
8. Як в Turbo Debugger відображаються машинні команди та їхній асемблерний код?
9. Чим відрізняється відображення команд у Turbo Debugger та OllyDbg?
10. Яким чином можна відслідковувати зміни в регістрах і пам’яті під час виконання програми в налагоджувачі?

Література: [1-6, 9-14]

Лабораторна робота №9. Кодування машинних команд

Мета роботи: вивчити принципи формування машинних кодів у процесі реверс-інженірингу

Теоретичні відомості

Машинна команда – це інструкція, яка подається на виконання центральному процесору у вигляді послідовності байтів. Її структура визначається архітектурою процесора та входить до Instruction Set Architecture ISA. У x86 архітектурі інструкції мають змінну довжину від 1 до 15 байтів. Структура типової машинної команди включає^

Opcode – байт або кілька байтів який визначає операцію наприклад MOV ADD JMP.

ModR/M – байт що описує адресацію операндів регістр або пам’ять.

SIB =Scale-Index-Base – додатковий байт адресації при використанні складних адресних виразів.

Displacement – зміщення якщо використовується непряма адресація.

Immediate – безпосереднє значення наприклад константа

Для аналізу відповідності між асемблерною інструкцією і машинним кодом зручно використовувати дизасемблери зокрема OllyDbg – інструмент що дозволяє відстежувати виконання програми покроково переглядати байти інструкцій та змінні у регістрах.

Порядок виконання роботи:

1. Ознайомтеся з базовим форматом машинної інструкції у x86 визначте які байти відповідають за команду операнди адресацію та значення
2. Напишіть у будь-якому текстовому редакторі або в компіляторі асемблері невелику програму що містить прості інструкції типу MOV AX BX ADD AL 1 SUB [BX] AL JMP short INC CX XOR AX AX
3. Зберігши програму у форматі exe відкрийте її в середовищі OllyDbg
4. Покроково проаналізуйте кожну інструкцію
5. Який машинний код у байтах відповідає кожній з них

6. Як змінюються регістри після виконання інструкцій
7. Як OllyDbg відображає структуру інструкції
8. Розшифруйте вручну 3-5 інструкцій визначивши
9. Де знаходиться opcode
10. Чи є байт ModR/M або SIB
11. Яке зміщення або безпосереднє значення кодується
12. Зіставте ваш аналіз із довідковими таблицями інструкцій x86 Opcode Map переконайтесь у правильності декодування.

Контрольні питання:

1. Що таке машинна команда і з яких компонентів вона складається
2. Що означає термін opcode і яку функцію він виконує
3. Які функції виконують байти ModR/M і SIB у x86 інструкціях
4. Чим відрізняється безпосередній операнд immediate від адресного
5. Чому інструкції в архітектурі x86 мають змінну довжину
6. Яке призначення дизасемблера OllyDbg у дослідженні машинних кодів
7. Як визначити яка частина машинної команди відповідає за адресу або операнд
8. Наведіть приклад простої інструкції MOV ADD тощо з її машинним кодуванням і поясніть кожен байт.

Література: [1-2, 4, 7-13]

Лабораторна робота №10. Організація шин пам’яті і шин розширення

Мета роботи: вивчити відмінності між шинами процесора та периферії

Теоретичні відомості

У комп’ютері для обміну даними між основними компонентами використовуються шини – спеціалізовані канали передачі інформації. Основні типи шин: адресна, даних та керуюча. Усі разом вони формують системну шину, яка з’єднує процесор, оперативну пам’ять і контролери.

Шина пам’яті призначена для передачі даних між процесором та оперативною пам’яттю. Її ключові параметри – ширина шини (у бітах) та тактова частота, від яких залежить пропускна здатність.

Шини розширення, такі як PCI, PCIe, ISA, використовуються для підключення зовнішніх пристроїв – звукових карт, мережевих адаптерів, відеокарт тощо. Вони мають свою структуру, характеристики та протоколи обміну. Шина PCIe, наприклад, використовує серійну передачу даних і забезпечує високі швидкості обміну, маючи різну кількість ліній x1, x4, x8, x16.

Порядок виконання роботи:

1. Вивчіть теоретичні основи організації шин у комп’ютерних системах. Опрацюйте матеріал щодо типів шин, які використовуються у комп’ютерах: системна шина, шина пам’яті, шини розширення. Зверніть увагу на їхню структуру, функціональне призначення та складові – шина адреси, шина даних, шина керування. Особливо зверніть увагу, як шини пов’язують між собою процесор, оперативну пам’ять і периферійні пристрої.

2. Проведіть аналітичне порівняння шин пам’яті та шин розширення. Охарактеризуйте обидва типи шин за такими параметрами: пропускна здатність, швидкодія, призначення, гнучкість у підключенні пристроїв, підтримка гарячого підключення, тип передачі даних. Оформіть результати у вигляді порівняльної таблиці або текстового аналізу. Поясніть, чому певні шини краще підходять для високошвидкісного обміну даними, а інші – для периферійних пристроїв.

3. Зробіть короткий технічний висновок.
На основі попереднього аналізу сформулюйте висновок про принципові відмінності між шинами, які використовуються для взаємодії з пам’яттю, та шинами для розширення системи. Опишіть, як вибір типу шини впливає на загальну продуктивність та масштабованість комп’ютерної системи.

4. Підготуйте та запустіть віртуальну машину в середовищі QEMU. Створіть конфігурацію віртуальної машини, яка міститиме щонайменше один пристрій пам’яті та один пристрій периферії на прикладі USB або мережевої карти. Зверніть увагу на параметри, які відповідають за підключення пристроїв до шин – опції -device, -drive, -usb, -net.

5. Проаналізуйте шини пристроїв у запущеній системі.
За допомогою інструментів lspci, lsusb, lshw, або dmesgзберіть інформацію про те, які пристрої було виявлено, через які шини вони взаємодіють із системою, і які драйвери для цього використовуються. Зафіксуйте типи шин, їхню структуру адресації та ідентифікатори.

6. Класифікуйте пристрої за типами шин.
Розподіліть пристрої, знайдені у віртуальному середовищі, на дві групи: ті, що використовують шини пам’яті (що з’єднують CPU і RAM), та ті, що працюють через шини розширення PCI, USB, SATA. Поясніть принципові відмінності у способі взаємодії цих пристроїв із процесором.

Контрольні питання:

1. Що таке шина в комп’ютерній архітектурі? Які основні функції вона виконує?
2. Які основні типи шин використовуються в комп’ютерних системах? Назвіть і охарактеризуйте їх.
3. У чому полягає відмінність між шиною даних, шиною адреси та шиною керування?
4. Які особливості системної шини? Для чого вона використовується?
5. Що таке шина розширення? Які пристрої підключаються через неї?

6. Порівняйте пропускну здатність і швидкодію шин пам’яті та шин розширення.

7. Назвіть сучасні типи шин розширення і коротко охарактеризуйте кожну з них: PCI, PCIe, USB, SATA, Thunderbolt.

8. Які шини використовуються для підключення оперативної пам’яті в сучасних комп’ютерах?

9. Що таке адресація пристроїв на шині? Як визначається, до якого пристрою звертається процесор?

10. Які інструменти в операційній системі Linux дозволяють визначити типи шин, до яких підключені пристрої?

11. Яку інформацію можна отримати за допомогою команд lspci, lsusb і lshw?

12. Як впливає тип шини на продуктивність пристроїв і всієї комп’ютерної системи загалом?

13. Що таке гаряче підключення і які шини його підтримують?

14. У чому полягають переваги шин типу PCI Express над традиційними PCI?

15. Які шини були задіяні у вашій конфігурації в QEMU? Як ви це з’ясували?

Література: [1-6, 11, 14-15]

Лабораторна робота №11. Підключення і тестування інтерфейсів SATA та M.2

Мета роботи: дослідити особливості підключення накопичувачів, оцінити їхню продуктивність за допомогою відповідних програмних засобів.

Теоретичні відомості

Інтерфейс SATA, або Serial ATA – це один з основних інтерфейсів підключення накопичувачів у персональних комп'ютерах. Він використовується для підключення жорстких дисків HDD та твердотільних накопичувачів SSD. SATA підтримує гаряче підключення, має просту структуру кабелів, але порівняно обмежену пропускну здатність (до 6 Гбіт/с у версії SATA III).

Інтерфейс M.2 – це сучасний стандарт роз'єму, що дозволяє підключати високошвидкісні SSD-накопичувачі без використання кабелів. M.2 може використовувати інтерфейси SATA або PCIe NVMe. Пристрої з підтримкою PCIe NVMe M.2 мають значно вищу швидкість обміну даними, оскільки передають дані напряму через шину PCI Express, минаючи контролери SATA.

Основні відмінності між SATA SSD і M.2 NVMe SSD полягають у швидкодії, способі підключення, форм-факторі та енергоспоживанні. Також M.2 може підтримувати різну кількість ліній PCIe (x2, x4), що прямо впливає на продуктивність.

Порядок виконання роботи:

1. Ознайомтеся з технічними характеристиками накопичувачів, доступних для тестування – SATA SSD, M.2 SATA SSD, M.2 NVMe SSD. Визначіть форм-фактор, інтерфейс, тип роз'єму, підтримку протоколів.
2. Перевірте, чи визначаються накопичувачі у BIOS/UEFI та в системі.
3. За допомогою засобів операційної системи визначте тип інтерфейсу для кожного накопичувача. Визначте, який контролер використовується – SATA або PCIe.
4. Встановіть або запустіть програму для тестування швидкодії дисків. Для кожного типу накопичувача проведіть тест читання і запису. Зафіксуйте результати.

5. Побудуйте порівняльну таблицю з результатами: тип накопичувача, інтерфейс, максимальна швидкість читання та запису, тип контролера.

6. Проаналізуйте отримані дані. Поясніть, чому SSD з інтерфейсом M.2 NVMe демонструє вищу продуктивність порівняно з SATA SSD. Зробіть висновок щодо доцільності використання тих чи інших накопичувачів у різних системах – ноутбук, сервер, ігровий ПК тощо.

Контрольні питання:

1. Які основні відмінності між інтерфейсами SATA та M.2?
2. У чому полягає перевага SSD-накопичувачів з інтерфейсом M.2 NVMe над SATA SSD?
3. Що таке протокол NVMe і як він впливає на швидкодію накопичувача?
4. Які типи роз'ємів та форм-факторів використовуються для M.2 SSD?
5. Чим відрізняються інтерфейси M.2 SATA та M.2 PCIe?
6. Як за допомогою програмного забезпечення визначити тип інтерфейсу підключеного накопичувача?
7. Які утиліти можна використовувати для тестування швидкості читання та запису диска?
8. У яких випадках доцільніше використовувати SATA SSD, а коли M.2 NVMe?

Література: [1-2, 4, 7-9, 13]

Лабораторна робота №12-13. Типи архітектур процесорів

Мета роботи: ознайомитися з архітектурними підходами (Фон Нейманівська архітектура, Гарвардська архітектура, архітектура x86, архітектура ARM)

Теоретичні відомості

Фон Нейманівська архітектура – це класичний підхід до побудови комп'ютера, за якого пам'ять програми і даних спільна, а обробка здійснюється послідовно. Основною проблемою цієї архітектури є «вузьке місце фон Неймана» – обмеження на швидкість обміну даними між процесором і пам'яттю.

Гарвардська архітектура передбачає розділення пам'яті для програм і для даних, що дозволяє одночасно зчитувати інструкції та обробляти дані. Така архітектура часто застосовується в мікроконтролерах і вбудованих системах.

Архітектура x86 застосовується при виробництві мікропроцесорів у компаніях Intel та AMD. Обидва виробники використовують модифіковану фон Нейманівську архітектуру з елементами паралельної обробки, кешування, мультипоточності, інтегрованої графіки та інших сучасних технологій.

Останні архітектури Intel (станом на 2024–2025):

1. Meteor Lake – перехід на модульну (tile-based) архітектуру, розділення SoC на окремі блоки, вперше використовує технологію Intel Foveros 3D Packaging.
2. Raptor Lake – гібридна архітектура з ядрами P-core (продуктивні) та E-core (ефективні), підтримка DDR5, PCIe 5.0.
3. Alder Lake – перша масова реалізація гібридної архітектури (x86 + ARM-подібна концепція), Intel Thread Director.
4. Tiger Lake – мобільна архітектура з ядрами Willow Cove, графіка Iris Xe.
5. Ice Lake – 10-нм архітектура, покращена підтримка AI-обчислень, AVX-512.

Ключові технології Intel:

1. Тактовий цикл «Tick-Tock»: поперемінне впровадження нових технологічних процесів («Tick») і нових мікроархітектур («Tock»).

2. Intel Thread Director: керування потоками між P-core та E-core.

3. Hyper-Threading, Turbo Boost, Foveros 3D.

Останні архітектури AMD:

1. Zen 5 (Ryzen 9000) – підвищення IPC, енергоефективності, підтримка AI-інструкцій.

2. Zen 4 – 5-нм техпроцес, підтримка DDR5, PCIe 5.0.

3. Zen 3 – суттєве зростання продуктивності на 1 такт, новий CCX-дизайн.

4. Zen 2 – розділення на чиплети, незалежна I/O секція.

5. Zen+ – оптимізація техпроцесу, зменшення затримок у кеші та RAM.

Ключові технології AMD: Infinity Fabric – високошвидкісна внутрішня шина, що зв'язує ядра, кеші та периферійні компоненти; Chiplet design – модульна структура процесора; Simultaneous Multithreading (SMT) – аналог Intel Hyper-Threading.

Порівняльна характеристика процесорів Intel Core наведена у таблиці 12.1.

Таблиця 12.1 – Особливості та характеристики процесорів серій Core i3-i9

Серія процесора	Призначення	Кількість ядер/потоків	Особливості
Core i3	Базовий рівень, офісна робота	4–8	Низьке енергоспоживання, бюджетні моделі
Core i5	Середній рівень, ігри, мультимедіа	6–12	Оптимальний баланс ціни і продуктивності
Core i7	Висока продуктивність	8–16	Підтримка Hyper-Threading, вища кеш-пам'ять
Core i9	Преміум-сегмент, контент, рендеринг	12–24	Максимальна кількість ядер, підтримка AI, PCIe Gen5

Архітектура ARM (на прикладі Apple Silicon).

Apple розробила власну серію ARM-процесорів для комп'ютерів – Apple Silicon, починаючи з M1, яка стала першим ARM-чипом у MacBook. На відміну

від x86, ARM має RISC-архітектуру (Reduced Instruction Set Computing), що означає простіші інструкції та нижче енергоспоживання.

Серії Apple Silicon:

1. Apple M1 – 8 ядер (4 продуктивні + 4 енергоефективні), інтегрована графіка, пам’ять у SoC.

2. M1 Pro / M1 Max – більше ядер, покращена графіка, розширена шина пам’яті.

3. Apple M2 – покращення IPC, більша пропускна здатність пам’яті, нові медіаблоки.

4. Apple M3 (2023) – 3-нм техпроцес, ще більша енергоефективність, нова графіка Dynamic Caching.

5. Apple M4 (2024) – високопродуктивний RISC-процесор, ще тісніша інтеграція компонентів SoC.

Ключові особливості Apple Silicon:

1. Уніфікована пам’ять (Unified Memory Architecture) – процесор і графіка використовують спільну оперативну пам’ять.

2. Neural Engine – окремий блок для обробки AI-інструкцій.

3. Інтеграція SoC – усі компоненти (CPU, GPU, RAM, I/O) у єдиному кристалі.

4. Низьке тепловиділення при високій продуктивності.

Порядок виконання роботи

1. Ознайомитися з принципами побудови фон Нейманівської та Гарвардської архітектур. Побудувати схематичні зображення кожної з них.

2. Зібрати інформацію про сучасні архітектури процесорів Intel та AMD (п’ять останніх поколінь кожного виробника), вказавши техпроцес, кількість ядер, кеш, підтримувані технології. Дані подати у вигляді таблиці.

3. Дослідити архітектуру ARM на прикладі Apple Silicon: M1–M4. Вказати основні відмінності від x86, переваги у мобільних і ноутбучних платформах.

4. Провести аналіз, у яких типах систем краще використовувати архітектури x86, а в яких – ARM. Зробити висновки.

Контрольні питання:

1. У чому полягає принципова різниця між фон Нейманівською та Гарвардською архітектурами?
2. Які переваги та недоліки має фон Нейманівська архітектура?
3. Які сучасні архітектури використовують компанії Intel та AMD? Назвіть останні покоління процесорів.
4. Що таке архітектура Zen і які її ключові особливості? Які версії архітектури Zen реалізовані в AMD Ryzen?
5. Що таке Intel Thread Director і як він працює в гібридній архітектурі?
6. Поясніть, що таке Infinity Fabric. Яку роль він виконує в архітектурі AMD?
7. Чим відрізняються процесори ARM від x86? У чому полягають переваги ARM у мобільних і ноутбучних системах?
8. Які ключові інновації реалізовані в процесорах Apple Silicon M1–M4? Назвіть відмінності у порівнянні з традиційними x86-процесорами.
9. У чому полягає концепція уніфікованої пам’яті (Unified Memory Architecture) в Apple Silicon?
10. Які особливості має архітектура Intel Core i3/i5/i7/i9? Як відрізняються ці серії між собою за продуктивністю та призначенням?

Література: [1-7, 10-12, 15]

Лабораторна робота №14. Архітектури CISC та RISC

Мета роботи: визначити функціональні особливості процесорів CISC та RISC

Теоретичні відомості

Архітектура процесора визначає набір інструкцій, структуру регістрів, принципи обробки команд та особливості реалізації програмного коду. Два основні підходи – CISC (Complex Instruction Set Computer) та RISC (Reduced Instruction Set Computer) – сформували дві протилежні парадигми у побудові процесорів.

CISC-архітектура характерна великою кількістю складних інструкцій, які можуть виконувати декілька операцій за один машинний цикл. Такий підхід зменшує обсяг коду, дозволяє напряму працювати з пам’яттю та спрощує компілятори. Класичним прикладом є архітектура x86 Intel та AMD.

RISC-архітектура використовує прості, однотипні, швидкі інструкції з фіксованою довжиною, орієнтована на більшу кількість команд за цикл. Всі операції проводяться з регістрами, доступ до пам’яті здійснюється лише через завантаження та збереження. Представники – ARM, RISC-V, Apple Silicon.

Переваги CISC: компактний код, складні операції за одну інструкцію, зручність для високорівневих мов. Переваги RISC: простіша архітектура, ефективність у виконанні, зручність для оптимізації та паралелізму.

Порядок виконання роботи:

1. Опрацюйте теоретичні матеріали щодо принципів архітектур CISC і RISC. Зверніть увагу на те, як обробляються інструкції, які типи регістрів використовуються, як організовано роботу з пам’яттю та адресацією.

2. Ознайомтеся з прикладами типових інструкцій у стилі CISC (наприклад, x86: MOV AX, [BX+4], ADD AL, 5, LOOP label) та RISC (наприклад, ARM або RISC-V: LDR, ADD, BNE).

Візьміть просту задачу, наприклад: обчислення C = A + B, де A, B і C – змінні у пам’яті. Реалізуйте її у вигляді машинних інструкцій у двох стилях: у CISC-

архітектурі (наприклад, x86-синтаксис), а у RISC-архітектурі (наприклад, у стилі ARM або псевдокод RISC-V).

Для цього знадобиться таке ПЗ: для CISC/x86 – асемблер або дизасемблер чи DOSBox. Для RISC/ARM/RISC-V: онлайн-емулятор або середовище для ARM: ARM Keil Studio, онлайн-платформа ARMulator (рекомендовано), або емулятор RISC-V: Venus для RISC-V Ripes– RISC-V процесорний симулятор.

3. Проаналізуйте, скільки інструкцій потрібно для виконання задачі у кожному підході, які з них використовують пам’ять, які регістри залучаються, як організована адресація.

Контрольні питання:

1. У чому полягає основна ідея CISC-архітектури?
2. Які ключові риси притаманні RISC-архітектурі?
3. Як організована адресація пам’яті в CISC та в RISC?
4. Які переваги надає кожна з архітектур при написанні низькорівневого коду?
5. Які процесори на базі CISC та RISC вам відомі?
6. У чому полягає різниця між інструкціями MOV AX, [BX+4] і комбінацією LDR + ADD у RISC?
7. Як архітектурний стиль впливає на продуктивність комп’ютера?
8. Чому ARM-процесори широко використовуються в мобільних пристроях, тоді як x86 – у ПК?

Література: [1-3, 5, 7-9, 14]

Лабораторна робота №15. Інструкційний ISA набір процесора

Мета роботи: дослідити типи інструкцій: арифметичні, логічні, керування

Теоретичні відомості

ISA (Instruction Set Architecture) – це набір команд, який процесор розуміє і виконує. Команди поділяються на групи за функціональністю: арифметичні, логічні, керуючі, обмін з пам'яттю, спеціальні, тощо. Кожна архітектура – x86, ARM, чи RISC-V – має власний ISA, хоча типові інструкції часто подібні за призначенням.

Арифметичні інструкції виконують базові обчислення: додавання, віднімання, множення, ділення тощо (наприклад: ADD, SUB, MUL, DIV).

Логічні інструкції виконують побітові операції: AND, OR, XOR, NOT, а також зсуви (SHL, SHR).

Інструкції керування змінюють порядок виконання програми: умовні та безумовні переходи JMP, JE, JNE, JG, JL, виклики процедур CALL і повернення RET.

ISA x86 – це складна CISC-архітектура з великою кількістю команд, підтримкою різних адресних режимів і регістрів. У середовищі MASM (Microsoft Macro Assembler) інструкції подаються у зрозумілому для людини форматі і можуть використовуватись для написання як низькорівневого коду, так і частково оптимізованого машинного.

Порядок виконання роботи:

1. Ознайомтеся з базовим синтаксисом асемблера MASM та структурою програми (сегменти data, code, директиви, макроси).

2. Розгляньте приклади програм, що містять арифметичні – ADD, SUB, INC, DEC, MUL, DIV та логічні інструкції AND, OR, XOR, NOT, SHR, SHL. Поясніть їх роботу.

3. Вивчіть приклади інструкцій переходу та керування виконанням коду JMP, JE, JNE, CALL, RET.

4. Запишіть власні приклади фрагментів коду для кожної групи інструкцій. Виконайте їх у MASM, спостерігаючи за зміною регістрів і результатами.

5. Уточніть, як впливають інструкції на прапори (флаги) процесора: Zero Flag (ZF), Carry Flag (CF), Sign Flag (SF), Overflow Flag (OF).

6. Зробіть порівняння роботи логічних і арифметичних інструкцій з однаковими операндами.

Контрольні питання:

1. Що таке ISA та яку роль він відіграє у роботі процесора?
2. Які основні типи інструкцій ISA ви можете назвати? Які задачі вирішує кожен тип?
3. Назвіть приклади арифметичних інструкцій у x86-архітектурі та поясніть їхню дію.
4. Що виконують логічні інструкції AND, OR, XOR і які вони мають застосування?
5. Які інструкції використовуються для організації переходів у програмі? Чим відрізняються умовні та безумовні переходи?
6. Які регістри та прапори процесора залучаються під час виконання арифметичних і логічних інструкцій?
7. Що таке стекова інструкція і як вона пов’язана з викликами процедур (CALL, RET)?
8. У чому полягає відмінність між інструкцією INC і ADD, якщо обидві збільшують значення?

Література: [1-5, 7-12, 14]

Лабораторна робота №16. Поняття конвеєризації виконання команд

Мета роботи: вивчити принципи роботи конвеєра команд

Теоретичні відомості

Конвеєризація виконання команд – це метод підвищення продуктивності процесора шляхом розподілу виконання команд на окремі стадії. Кожна стадія обробляє певний етап команди (наприклад, отримання команди, її декодування, виконання, доступ до пам’яті та запис результату). Завдяки цьому одночасно обробляються декілька команд на різних стадіях.

Стадії конвеєра: типовий конвеєр включає 4-5 стадій – IF, Instruction Fetch – отримання команди з пам’яті; ID, Instruction Decode – декодування команди та підготовка операндів; EX, Execute – виконання операцій; MEM, Memory Access – доступ до комірок пам’яті (лише якщо потрібно); WB, Write Back – запис результату у регістр. Конвеєр дозволяє кожній стадії працювати незалежно, що скорочує простої процесора.

Основною перевагою є підвищення пропускної здатності процесора – кількість виконаних команд за одиницю часу зростає. Крім того, конвеєр дозволяє оптимізувати використання ресурсів процесора, зменшуючи час простою його окремих модулів.

Конвеєризація має обмеження: при виникненні залежностей між командам, наприклад, коли одна команда використовує результат попередньої, то тоді можуть виникати простої, або потрібно застосовувати методи перепланування та предиктивного виконання. Також збільшується складність апаратної реалізації.

Розглянемо приклад задачі на конвеєризацію в MASM.

Виконати послідовність базових команд над регістрами і прослідкувати проходження команд через конвеєр: AND AX, BX – OR CX, DX – ADD AX, CX.

```
.MODEL SMALL
.STACK 100h
.DATA
   ; Ініціалізація регістрів
   val1 DW 0F0Fh
   val2 DW 00FFh
   val3 DW 1234h
```

```
    val4 DW 0F0Fh
.CODE
MAIN PROC
    MOV AX, val1   ; AX = 0F0Fh
    MOV BX, val2   ; BX = 00FFh
    MOV CX, val3   ; CX = 1234h
    MOV DX, val4   ; DX = 0F0Fh

    ; Базові операції
    AND AX, BX     ; AX = AX AND BX
    OR CX, DX      ; CX = CX OR DX
    ADD AX, CX     ; AX = AX + CX

    MOV AH, 4Ch
    INT 21h
MAIN ENDP
END MAIN
```

У цій програмі спочатку виконується ініціалізація регістрів. Задаються початкові значення для AX, BX, CX та DX, щоб підготувати процесор до виконання операцій. Це дозволяє впевнено відстежувати зміни в регістрах під час моделювання команд.

Далі виконується команда AND AX, BX**,** яка здійснює логічну операцію AND над регістрами AX та BX. Результат записується в регістр AX. У конвеєрі ця команда проходить через усі стадії: IF → ID → EX → MEM → WB, що ілюструє поетапне виконання команди та підготовку даних для наступних операцій.

Наступною виконується команда OR CX, DX, яка здійснює логічну операцію OR над регістрами CX і DX. Результат записується в CX. Ця команда може виконуватися паралельно з наступною командою на конвеєрі, якщо немає залежності від регістра AX, що дозволяє збільшити продуктивність за рахунок конвеєризації.

Команда ADD AX, CX виконує арифметичне додавання значень AX та CX. Тут виникає залежність від попередніх команд: AX вже був змінений командою AND, а CX – командою OR. У конвеєрі це може спричинити затримку (data hazard), оскільки потрібно дочекатися, поки попередні результати стануть доступними для обчислення.

Узагальнюючи, у процесорі команди виконуються у конвеєрі, тобто частини наступної команди можуть починати виконання, поки попередня ще не завершена. При правильному плануванні команд, наприклад, використовуючи їх переупорядкування або вставку NOP, можна значно зменшити затримки через залежності та підвищити ефективність виконання.

Порядок виконання роботи:

1. Ознайомитися з теоретичними відомостями про конвеєризацію та стадії виконання команд.
2. Розглянути приклад виконання команд без конвеєра та з конвеєром на схемі.
3. Проаналізувати часові діаграми (таймінги) виконання команд на конвеєрі.
4. Визначити можливі конфлікти (залежності даних, конфлікти ресурсів) та способи їх усунення.
5. Написати програми на MASM для виконання базових операцій:
   - операція AND над регістрами;
   - операція OR над регістрами;
   - операція ADD над регістрами;
6. Відстежити порядок виконання команд та проаналізувати їх проходження через конвеєр.
7. Підвести підсумки: порівняти ефективність виконання команд з конвеєром та без нього, зробити висновки щодо оптимізації виконання.

Контрольні питання:

1. Що таке конвеєризація виконання команд?
2. Які стадії входять до класичного конвеєра процесора?
3. Як конвеєр підвищує продуктивність процесора?
4. Що таке залежності між командами, і як вони впливають на конвеєр?
5. Що таке таймінгова діаграма конвеєра?
6. Які типи конфліктів можуть виникати при конвеєризації?
7. Що таке стадія Write Back і яка її роль у конвеєрі?

8. Як вирішуються конфлікти даних у конвеєрі?

9. У чому різниця між пропускною здатністю та часом виконання однієї команди?

10. Наведіть приклад ситуації, коли конвеєр не підвищує продуктивність.

Література: [1-6, 12-15]

Лабораторна робота №17. Організація введення-виведення

Мета роботи: дослідити режими введення/виведення на рівні архітектури процесора

Теоретичні відомості

Введення-виведення у середовищі MASM організовується переважно за допомогою переривань DOS, зокрема INT 21h. Цей механізм дозволяє зчитувати символи з клавіатури, виводити символи на екран, а також працювати з файлами.

Команди введення-виведення поділяються на кілька типів: з буферизацією та без буферизації. Наприклад, функція INT 21h, AH=01h зчитує один символ з клавіатури без відображення на екрані, а AH=08h – з відображенням. Для виведення символу використовується AH=02h. Символи в MASM представлені у вигляді коду ASCII. Це дозволяє програмно виконувати операції над символами, порівнювати їх, змінювати регістри, та виводити відповідні коди на екран. Правильна організація введення-виведення дозволяє ефективно керувати взаємодією користувача з програмою та забезпечує наочність для навчальних цілей.

Розглянемо приклад задачі в MASM: написати програму на зчитування одного символу та його виведення.

```
.MODEL SMALL
.STACK 100h
.CODE
MAIN PROC
   MOV AH, 01h      ; Функція: зчитати символ з клавіатури
   INT 21h          ; Виклик DOS
   MOV DL, AL       ; Копіюємо символ у DL для виведення
   MOV AH, 02h      ; Функція: вивести символ на екран
   INT 21h          ; Виклик DOS
   MOV AH, 4Ch      ; Завершення програми
   INT 21h
MAIN ENDP
END MAIN
```

Спочатку у регістр AH записується код функції DOS 01h, яка відповідає за зчитування одного символу з клавіатури. Після виконання переривання INT 21h введений символ опиняється у регістрі AL.

Далі виконується переривання DOS 21h, яке обробляє введення з клавіатури. Програма чекає, поки користувач натисне клавішу, після чого введений символ зберігається у AL.

Потім значення введеного символу копіюється з AL у DL, оскільки функція виведення символу використовує регістр DL для визначення символу, який потрібно вивести на екран.

У регістр AH записується код функції DOS 02h, яка відповідає за виведення одного символу на екран. Викликається переривання DOS 21h для виконання цієї операції, і символ, що міститься у DL, з'являється на екрані. Наприкінці у регістр AH записується код функції DOS 4Ch, яка відповідає за завершення програми та повернення управління операційній системі. Виконується переривання INT 21h, після чого програма коректно завершує роботу.

Порядок виконання роботи:

1. Ознайомитися з основами організації введення-виведення в MASM та функціями DOS для роботи з клавіатурою і екраном.

2. Створити програму для зчитування одного символу з клавіатури та його виведення на екран.

3. Розширити програму для виведення ASCII-коду введеного символу.

4. Проаналізувати роботу програми за допомогою відстеження значень регістрів.

5. Спробувати зчитати кілька символів послідовно та вивести їх коди на екран.

6. Підвести підсумки: порівняти різні методи введення-виведення символів та зробити висновки щодо їх ефективності.

Контрольні питання:

1. Які функції DOS використовуються для введення та виведення символів у MASM?

2. Чим відрізняється функція AH=01h від AH=08h?
3. Як відображається символ на екрані за допомогою MASM?
4. Як визначити ASCII-код введеного символу?
5. Що таке регістр AL і яку роль він відіграє при введенні символу?
6. Як організувати послідовне зчитування кількох символів?
7. Як вивести повідомлення на екран у MASM?
8. Які обмеження має введення символів через INT 21h?
9. Що таке буферизація введення і навіщо вона потрібна?
10. Як завершити програму у MASM правильно, щоб повернути управління DOS?

Література: [1-4, 6-7, 10-12]

Лабораторна робота №18. Паралельне виконання та багатопоточність

Мета роботи: дослідити принципи організації паралельних обчислень

Теоретичні відомості

Паралельне виконання дозволяє одночасно виконувати кілька частин програми, що значно підвищує продуктивність на багатоядерних процесорах. У сучасних системах паралельне виконання реалізується за допомогою багатопоточності, коли один процес містить декілька потоків, кожен із яких виконує свою задачу.

Потік – це окремий контекст виконання всередині процесу, який має власний стек і регістри, але ділиться ресурсами процесу, такими як пам’ять. Багатопоточність дозволяє більш ефективно використовувати CPU, розподіляючи роботу між ядрами.

Для запуску багатопоточних програм використовують різні бібліотеки та фреймворки: у Windows це API потоків Win32, у C/C++ – std::thread, у Java – клас Thread та ExecutorService, у Python – threading або multiprocessing. Кожна реалізація дозволяє створювати потоки, управляти ними та синхронізувати їхню роботу.

Аналіз навантаження CPU допомагає зрозуміти ефективність багатопоточності: можна спостерігати, як процесорні ядра завантажені, чи відбувається блокування потоків, та виявити можливі «вузькі місця» у програмі.

Приклад багатопоточної програми на C++, яка демонструє паралельне виконання та дозволяє аналізувати навантаження CPU:

```
#include <iostream>
#include <thread>
#include <vector>

void worker(int id) {
    std::cout << "Потік " << id << " почав роботу\n";
    long long total = 0;
    for (long long i = 0; i < 100000000; ++i) {
        total += i;
    }
    std::cout << "Потік " << id << " завершив роботу. Сума = " << total <<
"\n";
```

```
}

int main() {
    const int numThreads = 4; // Кількість потоків
    std::vector<std::thread> threads;

    // Створення та запуск потоків
    for (int i = 0; i < numThreads; ++i) {
        threads.push_back(std::thread(worker, i));
    }

    // Очікування завершення всіх потоків
    for (auto &t : threads) {
        t.join();
    }

    std::cout << "Всі потоки завершили роботу\n";
    return 0;
}
```

Програма створює 4 потоки, кожен з яких виконує обчислювальну задачу – підсумовує числа від 0 до 100 000 000.

Функція worker отримує ідентифікатор потоку, виводить повідомлення про початок роботи, виконує обчислення та повідомляє про завершення.

У main ми створюємо вектор потоків threads і додаємо до нього кожен новий потік, використовуючи std::thread. Потім за допомогою join() чекаємо, поки всі потоки завершать роботу, після чого програма завершується.

Після запуску програми можна відкрити монітор ресурсів або диспетчер задач, щоб побачити, як навантажуються ядра CPU під час виконання потоків.

Ця ж програма на мові Python має такий лістинг:

```
import threading
import time

def worker(number):
    print(f"Потік {number} почав роботу")
    total = 0
    for i in range(10000000):
        total += i
    print(f"Потік {number} завершив роботу")
```

```
threads = []
for i in range(4):  # Створюємо 4 потоки
    t = threading.Thread(target=worker, args=(i,))
    threads.append(t)
    t.start()

for t in threads:
    t.join()

print("Всі потоки завершили роботу")
```

Програма створює 4 потоки, які виконують однакову обчислювальну задачу паралельно. Після запуску можна відкрити диспетчер завдань, щоб побачити завантаження CPU на різних ядрах.

Порядок виконання роботи:

1. Ознайомитися з основами паралельного виконання та багатопоточності у вибраній мові програмування.

2. Підготувати багатопоточну програму, у якій декілька потоків виконують обчислювальні або симуляційні задачі паралельно.

3. Запустити програму та спостерігати за виконанням потоків у середовищі розробки.

4. Відкрити системний монітор або диспетчер завдань, щоб проаналізувати навантаження на CPU та завантаження окремих ядер.

5. Зробити висновки щодо ефективності багатопоточності: чи всі ядра задіяні, чи немає простоїв через блокування потоків, наскільки швидше виконуються задачі порівняно з однопотоковим виконанням.

6. Підготувати звіт з результатами: скріншоти монітору CPU, опис програми, кількість потоків та спостереження.

Контрольні питання:

1. Що таке поток у контексті багатопоточності?

2. Чим багатопотоковий процес відрізняється від багатопроцесорного?

3. Які ресурси процесу розділяють потоки?

4. Як створити новий потік у Python чи C++?

5. Що таке синхронізація потоків і чому вона потрібна?

6. Що таке «data race» і як його уникнути?

7. Як перевірити завантаження CPU під час виконання багатопоточної програми?

8. Які методи дозволяють покращити ефективність багатопоточності?

9. Чим відрізняється паралельне виконання від послідовного?

10. Як оцінити, що програма використовує всі доступні ядра процесора ефективно?

Література: [1-2, 5-7, 9]

Лабораторна робота №19. Введення-виведення через порти та прямий доступ до пам’яті

Мета роботи: вивчити принципи взаємодії пристроїв вводу-виводу

Теоретичні відомості

Введення-виведення через порти використовується для обміну даними між центральним процесором та периферійними пристроями. Кожен порт має свою унікальну адресу у просторі вводу-виводу. Процесор передає дані пристрою за допомогою команд IN для зчитування з порту та OUT для запису у порт.

Прямий доступ до пам’яті DMA дозволяє периферійному пристрою безпосередньо записувати дані в оперативну пам’ять, минаючи CPU. Це підвищує продуктивність системи, оскільки процесор може виконувати інші задачі під час передачі даних. DMA керується спеціальним контролером, який встановлює адресу, розмір блоку та напрямок передачі.

У системах з введенням-виведення через порти зазвичай використовується портовий ввід-вивід. Для програміста важливо правильно визначити адресу порту та формат даних, щоб уникнути помилок під час зчитування чи запису.

Логічна схема вводу-виводу через порти показує взаємодію між CPU, контролером портів та периферійним пристроєм. Вона включає шину даних, шину адреси, сигнали управління та регістри портів для буферизації даних.

Приклад програми на C для зчитування даних з порту (для ОС Windows):

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>

int main() {
    unsigned short port = 0x378; // приклад порту (паралельний порт)
    unsigned char data;

    data = _inp(port); // зчитування одного байту з порту
    printf("Дані з порту: %02X\n", data);

    _getch();
    return 0;
}
```

Програма зчитує один байт з обраного порту і виводить його значення у шістнадцятковому форматі. На практиці адреса порту підбирається відповідно до підключеного пристрою.

Порядок виконання роботи:

1. Ознайомитися з принципами роботи вводу-виводу через порти та прямого доступу до пам’яті (DMA).
2. Розглянути логічну схему передачі даних через порти, включаючи CPU, шину адреси, шину даних і регістри портів.
3. Написати програму на MASM або C/C++ для зчитування даних з порту та виведення їх на екран.
4. Виконати програму та перевірити правильність зчитування даних з обраного порту.
5. Моделювати передачу даних через DMA, аналізуючи, як процесор звільняється від обробки безпосереднього копіювання даних.
6. Підготувати звіт з описом програми, схемою вводу-виводу та висновками щодо роботи портів та прямого доступу до пам’яті.

Контрольні питання:

1. Що таке порт у контексті введення-виведення?
2. Як процесор передає дані пристрою через порти?
3. Що таке команди IN і OUT?
4. Що таке прямий доступ до пам’яті (DMA)?
5. Які переваги використання DMA?
6. Що таке портовий ввід-вивід (I/O-mapped I/O)?
7. Які сигнали включені в логічну схему порту?
8. Як забезпечується буферизація даних у портах?
9. Як визначити адресу порту для конкретного пристрою?
10. Чим відрізняється доступ через порти від доступу до пам’яті безпосередньо?

Література: [1-5, 7-9, 13]

Лабораторна робота №20. Система переривань: типи і пріоритети

Мета роботи: ознайомитися з обробкою переривань у процесорі x86, навчитися працювати з емулятором EMU8086 та моделювати ситуації з пріоритетними перериваннями.

Теоретичні відомості

Система переривань у процесорі x86 дозволяє тимчасово перервати виконання поточної програми для обробки більш важливих подій. Переривання можуть виникати апаратно або програмно. Програмні переривання, як правило, викликаються командою INT, а апаратні – зовнішніми пристроями через контролер переривань.

Кожне переривання має свій номер і таблицю обробки, яка вказує адресу процедури обробника. Процедура обробника виконує необхідні дії, після чого процесор повертається до виконання перерваної програми за допомогою команди IRET.

Переривання поділяються за типами: масковані, які можна тимчасово заборонити, немасковані, які не можна заборонити та програмні. Крім того, існує система пріоритетів: якщо одночасно виникають кілька переривань, процесор обробляє перше за пріоритетом. Контролер переривань PIC забезпечує правильну черговість обробки.

В емуляторі EMU8086 можна моделювати переривання, викликаючи INT n для програмних переривань або використовуючи емуляцію апаратних сигналів. Це дозволяє наочно побачити, як обробляються пріоритетні переривання і як CPU зберігає контекст поточної програми.

Приклад програми на EMU8086 для реалізації переривань:

```
.model small
.stack 100h
.code
org 100h

start:
    mov ah, 09h
    lea dx, msg1
```

```
int 21h          ; виклик переривання для виводу першого повідомлення

int 3            ; програмне переривання №3 (тестове, демонстраційне)

mov ah, 09h
lea dx, msg2
int 21h          ; вивід другого повідомлення

mov ah, 4Ch
int 21h          ; завершення програми

msg1 db 'Перше повідомлення $'
msg2 db 'Друге повідомлення $'
end start
```

Ця програма демонструє виклик декількох переривань, включаючи тестове INT 3, і дозволяє прослідкувати порядок їх обробки та вплив пріоритетів.

Порядок виконання роботи:

1. Ознайомитися з принципами роботи системи переривань у процесорі x86 та її типами.
2. Вивчити таблицю переривань та номери стандартних переривань у EMU8086.
3. Написати програму, яка викликає кілька програмних переривань за допомогою команди INT і відслідковує їх виконання.
4. Змоделювати ситуацію з одночасним виникненням переривань різного пріоритету.
5. Використати регістри CPU для аналізу збереження контексту під час обробки переривань.
6. Підготувати звіт з результатами, включаючи приклади коду та схему пріоритетів переривань.

Контрольні питання:

1. Що таке переривання і які його типи існують?
2. Чим відрізняються апаратні та програмні переривання?
3. Як працює таблиця векторів переривань?
4. Що таке масковане та немасковане переривання?

5. Як процесор визначає пріоритет між одночасними перериваннями?
6. Яку роль відіграє контролер переривань (PIC)?
7. Як EMU8086 дозволяє моделювати переривання?
8. Яка команда повертає процесор до перерваної програми після обробки переривання?
9. Як відстежити збереження контексту під час обробки переривань?
10. Наведіть приклад ситуації, коли пріоритетні переривання необхідні в системі.

Література: [1-4, 6-7, 10-12]

Лабораторна робота №21. Емуляція простого процесора: робота регістрів та АЛП

Мета роботи: змоделювати роботу процесора на прикладі базових інструкцій

Теоретичні відомості

Емуляція процесора дозволяє наочно вивчати роботу внутрішніх компонентів, таких як регістри, АЛП і шини даних, без використання фізичного обладнання. Регістри процесора x86 використовуються для зберігання даних, адрес та результатів обчислень. Основні регістри включають AX, BX, CX, DX, SP, BP, SI, DI та сегментні регістри CS, DS, ES, SS.

Арифметико-логічний пристрій АЛП виконує операції додавання, віднімання, множення, поділу, а також логічні операції. Взаємодія між регістрами та АЛП забезпечує виконання команд процесора.

У емуляторі EMU8086 можна моделювати виконання інструкцій покроково, спостерігаючи зміни регістрів, прапорів та пам’яті. Це дозволяє аналізувати, як окремі інструкції впливають на стан процесора і як АЛП обробляє обчислення та логічні операції.

Моделювання обробки інструкцій включає такі етапи: вибір інструкції, її декодування, виконання в АЛП, запис результату в регістр та оновлення прапорів. Програма покрокової емуляції, яка допомагає зрозуміти внутрішню логіку роботи процесора на EMU8086:

```
.model small
.stack 100h
.code
org 100h

start:
    mov ax, 5       ; записуємо 5 у регістр AX
    mov bx, 3       ; записуємо 3 у регістр BX

    add ax, bx      ; AX = AX + BX (5 + 3 = 8)
    sub bx, 1       ; BX = BX - 1 (3 - 1 = 2)
    and ax, bx      ; AX = AX AND BX (8 AND 2 = 0)
```

```
    or bx, ax       ; BX = BX OR AX (2 OR 0 = 2)

    mov ah, 4Ch
    int 21h         ; завершення програми

end start
```

Ця програма демонструє роботу регістрів та АЛП при виконанні арифметичних і логічних операцій, дозволяє відстежити зміну значень AX і BX у емуляторі.

Порядок виконання роботи:

1. Ознайомитися з регістрами та АЛП процесора x86, їх призначенням і функціями.
2. Вивчити інструкції для роботи з регістрами та арифметично-логічні операції у EMU8086.
3. Написати просту програму на EMU8086, яка виконує декілька арифметичних і логічних інструкцій над регістрами.
4. Запустити програму в емуляторі та покроково відстежувати зміни регістрів та прапорів.
5. Проаналізувати роботу АЛП під час виконання операцій, звертаючи увагу на зміну значень у регістрах та прапорі переносу.
6. Підготувати звіт з результатами роботи, включаючи скріншоти стану регістрів та короткий опис кожної інструкції.

Контрольні питання:

1. Які основні регістри процесора x86 та їх призначення?
2. Що таке АЛП і які операції воно виконує?
3. Як взаємодіють регістри та АЛП під час виконання інструкцій?
4. Що відбувається під час виконання команди ADD у регістрі?
5. Як працюють логічні операції AND, OR, XOR у регістрах?
6. Що таке прапори процесора і які зміни вони відображають після операцій?

7. Як покрокова емуляція допомагає зрозуміти внутрішню роботу процесора?

8. Як в EMU8086 спостерігати стан регістрів та прапорів під час виконання програми?

9. Які етапи проходить інструкція від вибору до виконання?

10. Наведіть приклад ситуації, де важливо контролювати роботу АЛП та регістрів у програмі.

Література: [1-5, 8-10, 13]

Лабораторна робота №22. Емуляція процесора з підтримкою конвеєра

Мета роботи: змоделювати простий конвеєр команд

Теоретичні відомості

Конвеєризація виконання команд дозволяє підвищити продуктивність процесора, розділивши виконання команди на кілька стадій: отримання команди, декодування, виконання, доступ до пам’яті та запис результату. Завдяки цьому одночасно можуть оброблятися декілька команд на різних стадіях конвеєра.

Основна перевага конвеєризації полягає у збільшенні пропускної здатності процесора – кількість виконаних команд за одиницю часу зростає, навіть якщо час виконання однієї команди залишається тим самим. Конвеєр дозволяє скоротити простої функціональних блоків процесора та оптимізувати використання ресурсів.

У процесорі x86 можуть виникати проблеми при конвеєризації, пов’язані з залежностями між командами, конфліктами ресурсів або переходами. Для їх усунення застосовують методи перепланування команд або вставки NOP.

Емулятор EMU8086 дозволяє моделювати покрокове виконання команд, спостерігати зміни регістрів, прапорів та пам’яті, а також перевіряти правильну послідовність виконання команд без конфліктів. Це допомагає контролювати ефективність конвеєризації.

Приклад програми конвеєрної обробки на EMU8086:

```
.model small
.stack 100h
.code
org 100h

start:
   mov ax, 5      ; AX = 5
   mov bx, 3      ; BX = 3
   add ax, bx     ; AX = AX + BX
   sub bx, 1      ; BX = BX - 1
   and ax, bx     ; AX = AX AND BX
   or bx, ax      ; BX = BX OR AX

   mov ah, 4Ch
   int 21h        ; завершення програми
```

end start

Ця програма містить кілька арифметичних та логічних інструкцій, що дозволяє відстежити конвеєрне виконання та вплив залежностей між командами.

Порядок виконання роботи:

1. Ознайомитися з принципами конвеєризації команд у процесорі та її стадіями.
2. Відкрити емулятор EMU8086 і підготувати просту програму з декількома командами над регістрами.
3. Запустити програму покроково та спостерігати за станом регістрів і прапорів.
4. Визначити порядок проходження команд через стадії конвеєра та перевірити правильність їх виконання.
5. Проаналізувати можливі затримки або конфлікти між командами та запропонувати способи їх усунення.
6. Підготувати звіт з результатами роботи, включаючи скріншоти стану регістрів і пояснення проходження команд через конвеєр.

Контрольні питання:

1. Що таке конвеєризація виконання команд і які її основні стадії?
2. Які переваги дає конвеєризація для процесора?
3. Що таке залежності між командами і як їх усувають?
4. Які стадії проходить команда в конвеєрі x86?
5. Як вставка NOP допомагає при конфліктах команд?
6. Що таке конфлікт переходів і як його можна уникнути?
7. Як в EMU8086 можна відстежити проходження команд через конвеєр?
8. Як визначити правильність виконання команд у конвеєрі?
9. Чим відрізняється виконання команд у конвеєрі від послідовного виконання?

Література: [1-4, 6-7, 10-12]

Лабораторна робота №23. Оптимізація доступу до кеш-пам’яті

Мета роботи: ознайомитися з принципами роботи кеш-пам’яті

Теоретичні відомості

Кеш-пам’ять є високошвидкісним проміжним буфером між центральним процесором і оперативною пам’яттю, який зберігає недавно використані дані або дані, що ймовірно будуть потрібні в найближчому майбутньому. Основна мета кеша полягає у зменшенні часу доступу до пам’яті та підвищенні продуктивності системи. Кеш дозволяє процесору отримувати необхідні дані швидше, ніж безпосередньо звертаючись до оперативної пам’яті, що особливо важливо при виконанні обчислень, які потребують частого доступу до даних.

Сучасні процесори використовують багаторівневу кеш-пам’ять, яка включає рівні від L0 до L3. Кеш L0 є найшвидшим і зазвичай найменшим за обсягом. Він інтегрований безпосередньо у ядро процесора і служить для дуже швидкого доступу до даних і команд, які найбільш часто використовуються. Кеш L1 розташований поруч із ядром і забезпечує трохи більший обсяг, але трохи більший час доступу. Кеш L2 є спільним для одного або декількох ядер, має більший обсяг і помірний час доступу. Кеш L3 зазвичай є загальним для всіх ядер процесора, має найбільший обсяг і повільніший час доступу порівняно з попередніми рівнями, але все одно значно швидший за основну оперативну пам’ять. Типові ємності кеш-пам’яті сучасних процесорів можуть виглядати так: для L0 від кількох кілобайт, для L1 від 32 до 128 кілобайт, для L2 від 256 кілобайт до декількох мегабайт і для L3 від кількох мегабайт до десятків мегабайт.

Наприклад, у сучасних процесорах Intel Core i7 кеш L0 може мати ємність близько 4 кілобайт для команд і 4 кілобайт для даних на кожне ядро, кеш L1 – 32 кілобайти для даних і 32 кілобайти для команд на ядро, кеш L2 – 256 кілобайт на ядро, а кеш L3 – спільний для всіх ядер і може становити 8 мегабайт. У процесорах AMD Ryzen кеш L0 може мати 64 кілобайти на ядро, кеш L1 – 128 кілобайт на ядро, кеш L2 – 512 кілобайт на ядро, а кеш L3 – загальний для кристалу і доходить до 16 мегабайт. Такі розміри дозволяють забезпечити високу

швидкість доступу до даних і ефективно використовувати локальність пам’яті при обчисленнях (рис. 23.1).

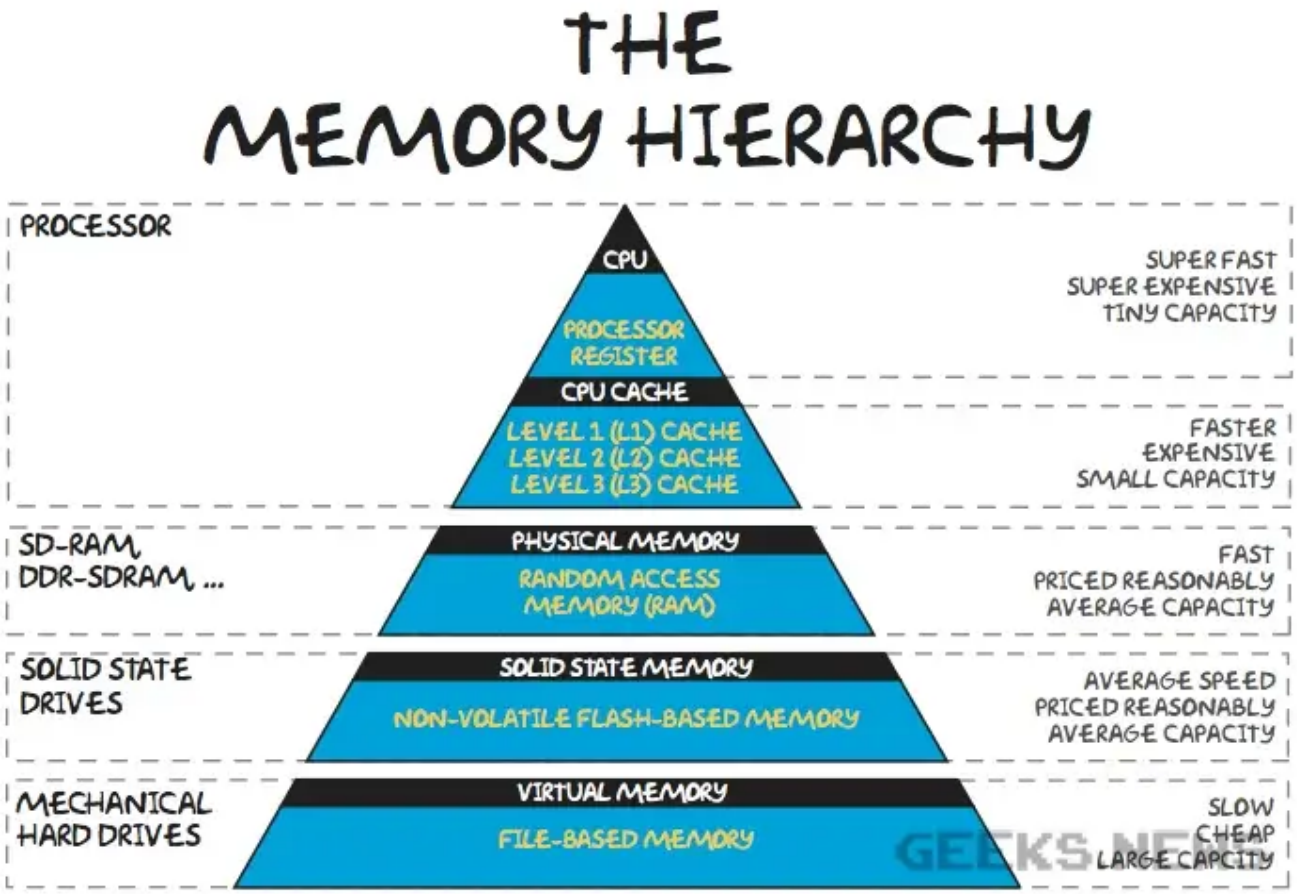


Рисунок 23.1 – Рівні і ієрархія пам’яті процесора [7]

Ефективність кеша значною мірою залежить від локальності даних і команд. **Лінійний доступ** до масиву або послідовна обробка блоків даних дозволяє процесору максимально використовувати кеш, оскільки сусідні елементи часто потрапляють у один блок кеша і зчитуються разом. Стрибкоподібний доступ, коли звернення здійснюються через великі проміжки пам’яті, призводить до частих промахів кеша, тобто ситуацій, коли потрібні дані відсутні у кеші і процесору доводиться звертатися до повільної оперативної пам’яті. **Попадання в кеш**, навпаки, забезпечує миттєвий доступ до даних і підвищує продуктивність. Показники ефективності кеша оцінюються через частоту попадань і промахів. Частота попадань показує відсоток запитів, оброблених без звернення до основної

пам’яті, а частота промахів показує відсоток запитів, які потребують доступу до повільнішої пам’яті.

Кеш-пам’ять характеризується такими параметрами, як час доступу, розмір блоків даних, організація по множинах і асоціативність. Час доступу визначає, скільки тактів процесора потрібно для отримання даних з кеша. Розмір блоку показує, скільки сусідніх байтів завантажується у кеш під час одного запиту. **Асоціативність кеша** визначає, скільки різних місць у кеші може приймати один і той же блок пам’яті, що впливає на частоту конфліктів. Для підвищення ефективності кеша застосовують оптимізацію доступу до даних, таку як впорядкування обчислень по рядках або блоках, що дозволяє зменшити промахи, уникнути конфліктів і максимально використовувати локальність даних.

Кеш-пам’ять дозволяє також покращувати продуктивність при паралельному виконанні програм і багатоядерній роботі, оскільки кожне ядро може використовувати власний кеш L1 і L0, зменшуючи навантаження на спільні ресурси. При правильно організованому доступі до кеша зменшується затримка виконання команд і забезпечується стабільна висока продуктивність системи в цілому.

Приклад команди для аналізу кеша у Windows:

wmic cpu get Name,L2CacheSize,L3CacheSize

Приклад команди для аналізу кеша у Linux:

lscpu | grep -i cache

Порядок виконання роботи:

1. Ознайомитися з технічними характеристиками кеш-пам’яті різних процесорів через прайси або документацію виробника.

2. Розшифрувати записи кеша в характеристиках процесорів, визначити обсяги для рівнів L0, L1, L2, L3.

3. На реальному комп’ютері або в емуляторі перевірити конфігурацію кеша за допомогою системних команд або утиліт.

4. Проаналізувати, який кеш є загальним для всіх ядер, а який виділений для конкретного ядра.

5. Виміряти вплив порядку доступу до масиву на ефективність кеша, реалізувавши лінійний і стрибкоподібний доступ у простій програмі.

6. Підготувати звіт з розшифровкою прайсів, скріншотами конфігурації кеша на комп'ютері та висновками щодо його ефективності.

7. *Засобами VS продемонструвати простий лінійний та стрибкоподібний доступ до масиву, імітуючи кеш-дизайн доступу.

Контрольні питання:

1. Що таке кеш-пам'ять і яка її роль у роботі процесора?
2. Чим відрізняються рівні L0, L1, L2 і L3 кеша?
3. Як визначити обсяг кеша у документації або прайсі процесора?
4. Що таке промах і попадання кеша?
5. Як порядок доступу до даних впливає на ефективність кеша?
6. Як можна отримати інформацію про кеш у Windows та Linux?
7. Який кеш загальний для всіх ядер, а який виділений для конкретного ядра?
8. Що таке локальність даних і як вона впливає на продуктивність кеша?
9. Як великі блоки даних і асоціативність кеша впливають на частоту промахів?
10. Наведіть приклад оптимізації алгоритму для кращого використання кеша.

Література: [1-2, 4-6, 15]

Лабораторна робота №24. Синхронізація кешу в багатопроцесорних системах

Мета роботи: дослідити протоколи когерентності кешу

Теоретичні відомості

У багатопроцесорних системах кожне ядро зазвичай має власну кеш-пам’ять. Для забезпечення правильної роботи спільних даних необхідно підтримувати когерентність кешів. Когерентність кешу означає, що всі процесори бачать одні й ті ж значення даних, навіть якщо вони зберігаються в різних кешах.

Протоколи когерентності визначають правила оновлення кешів і передачі інформації між ними. Найпоширенішими є протоколи MSI, MESI, MOESI та MESIF. Вони регулюють стан кеш-блоків і порядок оновлення при зміні даних у будь-якому ядрі. Наприклад, у протоколі MESI блоки даних можуть перебувати в станах Modified, Exclusive, Shared або Invalid. Якщо одне ядро змінює дані, інші кеші отримують сигнал про недійсність або оновлення, щоб запобігти конфліктам.

Багатопоточне програмування дозволяє моделювати доступ до спільних даних і досліджувати поведінку кешів. Несинхронізований доступ може призводити до умов гонки, коли одне ядро читає або змінює дані, які вже були змінені іншим ядром, що демонструє необхідність протоколів когерентності.

Ефективне використання багатоядерних систем вимагає не лише правильного доступу до спільних даних, але й знання механізмів синхронізації, таких як м’ютекси, семафори та атомарні операції. Це дозволяє програмі забезпечувати правильність даних і уникати непередбачуваних результатів.

Приклад програми на C++ з використанням потоків:

```
#include <iostream>
#include <thread>
#include <vector>
#include <atomic>

const int SIZE = 1000000;
std::vector<int> data(SIZE, 0);
std::atomic<int> sum(0);

void threadFunc(int id, int step) {
```

```
    for (int i = id; i < SIZE; i += step) {
        sum += data[i]; // атомарна операція
    }
}

int main() {
    const int numThreads = 4;
    std::vector<std::thread> threads;

    for (int i = 0; i < numThreads; ++i) {
        threads.emplace_back(threadFunc, i, numThreads);
    }

    for (auto& t : threads) {
        t.join();
    }

    std::cout << "Сума елементів: " << sum << std::endl;
    return 0;
}
```

У цій програмі використовується атомарна операція для забезпечення синхронізації при паралельному доступі до спільного ресурсу. При несинхронізованому доступі результати могли б бути некоректними через умови гонки, що ілюструє важливість когерентності кешу.

Порядок виконання роботи:

1. Ознайомитися з принципами когерентності кешу у багатопроцесорних системах та протоколами MESI, MSI, MOESI.

2. Розробити багатопоточну програму, яка використовує спільні дані і демонструє доступ з декількох потоків.

3. Реалізувати синхронізацію потоків за допомогою м’ютексів або атомарних операцій.

4. Провести експерименти зі зміною стану кеш-блоків і спостерігати ефект когерентності через час виконання та правильність результатів.

5. Порівняти результати синхронізованого та несинхронізованого доступу до спільних даних.

Контрольні питання:

1. Що таке когерентність кешу і чому вона важлива у багатопроцесорних системах?
2. Які протоколи когерентності кешу існують і чим вони відрізняються?
3. Що означає стан Modified, Shared, Exclusive у протоколі MESI?
4. Як багатопоточні програми впливають на стан кешів?
5. Що таке умова гонки і чому вона виникає при несинхронізованому доступі?
6. Які механізми синхронізації потоків дозволяють уникати некоректних результатів?
7. Як атомарні операції допомагають підтримувати когерентність кешу?
8. Чим відрізняється синхронізований доступ від несинхронізованого з точки зору кеша?
9. Як можна моделювати зміну стану кеш-блоків у багатопоточних програмах?
10. Наведіть приклад ситуації, коли порушення когерентності кешу може призвести до помилок у програмі.

Література: [1-3, 9, 12-15]

Лабораторна робота №25. Дослідження ієрархії пам’яті сучасних процесорів

Мета роботи: вивчити розташування кешів і типи пам’яті

Теоретичні відомості

Ієрархія пам’яті сучасних процесорів створена з метою зменшення середнього часу доступу до даних. Чим ближче рівень пам’яті до процесора, тим менший час доступу, але й менший її обсяг. На найвищому рівні знаходяться регістри, які забезпечують безпосередню взаємодію з арифметико-логічним пристроєм. Кеш-рівні L0, L1, L2 і L3, які згадані у попередніх лабораторних роботах, поступово збільшують загальний обсяг пам’яті комп’ютерної системи, але мають більший час затримки.

Кеш L0 використовується для швидкої підготовки мікрооперацій, тоді як L1 і L2 зберігають найбільш часто використовувані інструкції та дані. L3 виконує функцію спільного буфера між усіма ядрами і синхронізує обмін із оперативною пам’яттю. DRAM є основною робочою пам’яттю системи, а доступ до неї регулюється контролером, який часто інтегрований у сам процесор.

Далі в ієрархії знаходяться пристрої постійного зберігання даних – SSD і HDD. SSD-пам’ять працює за принципом флеш-технології і має меншу затримку, ніж жорсткі диски, однак усе ще значно повільніша за кеш або оперативну пам’ять.

Ефективна взаємодія між цими рівнями є основою продуктивності системи. Висока швидкість обробки даних можлива лише тоді, коли більшість звернень потрапляє до кешу, а частота промахів залишається низькою. Для аналізу таких характеристик використовуються утиліти моніторингу та мікробенчмарки, які вимірюють латентність доступу до пам’яті.

Порядок виконання роботи:

1. Ознайомитися з теоретичними аспектами ієрархії пам’яті процесора, зрозуміти її рівневу структуру, взаємозв’язок між кешами різних типів і оперативною пам’яттю, а також принципи обміну даними між ними. Це допоможе

усвідомити логіку побудови експерименту та значення кожного рівня у загальній продуктивності системи.

2. За допомогою спеціалізованих утиліт, таких як CPU-Z, AIDA64 або lscpu, визначити характеристики процесора. Слід звернути увагу на обсяг кешів L1, L2, L3, їхню організацію – поділ на кеш даних і кеш інструкцій – та кількість ядер, які спільно використовують певні рівні пам’яті. Отримані дані потрібно занотувати для подальшого аналізу.

3. Після цього варто застосувати утиліту LatencyMon або аналогічний інструмент для вимірювання затримки пам’яті. Цей етап дозволить оцінити час доступу до різних рівнів пам’яті й побачити, як зростає латентність при переході від кеша L1 до оперативної пам’яті.

4. Наступним кроком є проведення експерименту за допомогою програми, що послідовно звертається до елементів великого масиву. Потрібно запустити програму кілька разів, змінюючи крок доступу до елементів масиву, і фіксувати отримані результати часу виконання. Це дозволить простежити, при якому розмірі даних відбувається вихід за межі кеша певного рівня.

5. Після проведення вимірювань необхідно побудувати графік залежності часу доступу від розміру даних. За формою графіка можна визначити межі кешів L1, L2 і L3, оскільки у цих точках зазвичай спостерігається різкий стрибок часу доступу.

6. На завершальному етапі слід провести аналіз отриманих даних, зробити висновки про ефективність роботи кеш-пам’яті.

Контрольні питання:

1. Як локальність даних впливає на продуктивність програми?
2. Які інструменти можна використовувати для вимірювання затримки доступу до пам’яті?
3. Як змінюється час виконання при збільшенні розміру даних понад обсяг кеша?
4. Яка роль контролера пам’яті у сучасних CPU?
5. У чому різниця між DRAM і SRAM?

6. Як результати експерименту можна інтерпретувати для визначення меж кешів L1-L3?

Література: [1-4, 6-7, 10-12]

Лабораторна робота №26. Аналіз архітектури процесорів Intel Core

Мета роботи: вивчити специфіку сучасних архітектур Intel

Теоретичні відомості

У цій лабораторній роботі досліджується архітектура сучасних процесорів Intel Core, зокрема серії Core i3, i5, i7 та i9 на прикладі архітектур Raptor Lake та Arrow Lake. Метою роботи є ознайомлення з особливостями сучасних процесорів, їхньою структурою, техпроцесом, кількістю ядер, кеш-пам'яттю та типами сокетів. Особлива увага приділяється аналізу відмінностей між різними лінійками процесорів і оцінці їхніх характеристик з точки зору продуктивності, енергоефективності та сумісності з платформою.

Архітектура Raptor Lake від Intel побудована на гібридній концепції з продуктивними (P-ядер) і ефективними (E-ядер) ядрами. Вона випущена на техпроцесі Intel 7 і сумісна із сокетом LGA 1700. Ця архітектура включає до 8 P-ядер і до 16 E-ядер, що дозволяє досягти до 24 ядер у певних моделях.

Для архітектури Arrow Lake Intel застосувала новий техпроцес (зокрема TSMC N3B для частини) і новий сокет LGA 1851. Сокет LGA 1851 має більше контактів у порівнянні з попереднім LGA 1700, і він передбачає використання материнських плат із чіпсетами серії 800.

У межах серій Core i3, i5, i7, i9 відрізняються не лише кількістю ядер й потоків, але й кеш-пам'яттю, максимальною частотою (boost), енергоспоживанням та іншими характеристиками. Наприклад, для Raptor Lake у Core i9 можуть бути до 24 ядер і до 32 потоків. Для Arrow Lake флагманська серія Core Ultra 9 (що еквівалентно i9) має конфігурацію 8 P-ядер + 16 E-ядер і підтримує розгін до ~5.7 GHz.

Оцінка техпроцесу і сокетів дає розуміння платформи: новий техпроцес означає покращену енергоефективність, можливість вищих частот, менший розмір транзисторів. Нові сокети означають зміни у сумісності материнських плат, контролерів пам'яті, підтримці DDR5, PCIe, тощо. Для споживача це означає, що перехід на нову серію може вимагати оновлення також материнської плати чи пам'яті.

Порядок виконання роботи:

Спочатку оберіть два процесори Intel: один із серії Core i3 або i5 на архітектурі Raptor Lake, та один із серії Core i7 або i9 на архітектурі Arrow Lake. Далі дослідіть технічні характеристики кожного: техпроцес (наприклад Intel 7 чи N3B), сокет (LGA 1700 чи LGA 1851), кількість P- і E-ядер, кеш-пам’ять, максимальна частота.

Після збору даних проаналізуйте: у чому полягає відмінність між техпроцесами, яку роль відіграє новий сокет, як зміни архітектури впливають на продуктивність і сумісність з платформою.

Складіть таблицю порівняння обраних процесорів із зазначенням параметрів і зробіть висновки: який із процесорів кращий за архітектурою, на що варто звернути увагу при виборі та чи виправданий перехід на нову лінійку з точки зору платформи.

Контрольні питання:

1. Який техпроцес використовує архітектура Raptor Lake і яку роль відіграє він у продуктивності?
2. Які зміни у сокеті відбулися при переході від LGA 1700 до LGA 1851?
3. У чому різниця між серіями Core i3, i5, i7, i9 з точки зору архітектури?
4. Як конфігурація P-ядер і E-ядер впливає на багатопоточну продуктивність?
5. Чому при зміні архітектури потрібно часто оновлювати материнську плату чи пам’ять?
6. Які переваги дає новий техпроцес у архітектурі Arrow Lake?
7. Які обмеження чи недоліки можуть виникнути при переході на нову платформу?
8. Як кеш-пам’ять та максимальна частота впливають на результати у задачах обробки даних та ігор?
9. Що означає сумісність архітектури з платформою і чому вона важлива?

Література: [1-5, 7, 9-10, 14]

Лабораторна робота №27. Аналіз архітектури процесорів AMD Ryzen

Мета роботи: вивчити особливості системної шини Infinity Fabric процесорів AMD Ryzen

Теоретичні відомості

Системна шина Infinity Fabric є ключовим елементом архітектури процесорів AMD Ryzen, забезпечуючи високошвидкісну комунікацію між різними компонентами, такими як ядра, кеш-пам’ять, контролери пам’яті та інші периферійні пристрої. Вона дозволяє AMD реалізувати модульну архітектуру, де різні функціональні блоки, як наприклад, ядра та I/O можуть бути виготовлені окремо та з’єднані за допомогою цієї шини, що забезпечує гнучкість та масштабованість процесорів.

Архітектура Zen 2 стала першою, що впровадила Infinity Fabric у вигляді "chiplet" дизайну, де обчислювальні ядра та I/O блок виготовляються окремо та з’єднуються через Infinity Fabric. Це дозволило AMD значно збільшити кількість ядер у процесорах без значного збільшення вартості виробництва.

З кожною новою архітектурою AMD вдосконалювала Infinity Fabric, збільшуючи пропускну здатність та зменшуючи латентність. Наприклад, у Zen 3 була покращена ефективність комунікації між ядрами та кеш-пам’яттю, що призвело до значного підвищення продуктивності в багатозадачних та багатопоточних навантаженнях.

У Zen 4 AMD впровадила підтримку DDR5 пам’яті та PCIe 5.0, що вимагало подальшого вдосконалення Infinity Fabric для забезпечення необхідної пропускної здатності. Крім того, була покращена інтеграція з графічними прискорювачами, що дозволило досягти кращої продуктивності в задачах, що вимагають високої обчислювальної потужності.

У Zen 5 AMD продовжила вдосконалення Infinity Fabric, зокрема, зменшивши енергоспоживання та покращивши масштабованість для серверних рішень. Це дозволило досягти ще більшої продуктивності при збереженні енергоефективності.

Порядок виконання роботи:

У рамках цієї лабораторної роботи необхідно провести порівняльний аналіз архітектур AMD Zen 2, Zen 3, Zen 4 та Zen 5, зокрема, зосередившись на особливостях реалізації системної шини Infinity Fabric у кожній з них. Для цього слід:

1. Вивчити технічні характеристики кожної архітектури, зокрема, пропускну здатність та латентність Infinity Fabric.
2. Оцінити, як зміни в Infinity Fabric вплинули на загальну продуктивність процесорів у різних архітектурах.
3. Порівняти ефективність комунікації між ядрами та іншими компонентами в кожній з архітектур.
4. Проаналізувати, як вдосконалення Infinity Fabric сприяло покращенню масштабованості та енергоефективності процесорів.

Результати аналізу слід представити у вигляді порівняльної таблиці, що відображатиме основні характеристики Infinity Fabric у кожній з архітектур, а також у вигляді текстового пояснення, яке розкриватиме вплив цих характеристик на загальну продуктивність та ефективність процесорів.

Контрольні питання:

1. Що таке системна шина Infinity Fabric і яку роль вона відіграє в архітектурі процесорів AMD Ryzen?
2. Як архітектура Zen 2 реалізує Infinity Fabric і які переваги це дає?
3. Які основні вдосконалення Infinity Fabric були впроваджені в архітектурах Zen 3, Zen 4 та Zen 5?
4. Як зміни в Infinity Fabric вплинули на продуктивність процесорів у різних архітектурах?
5. Які переваги та недоліки має модульна архітектура з використанням Infinity Fabric порівняно з традиційною монолітною архітектурою?

Література: [1-5, 8, 11-13]

Лабораторна робота №28. Архітектура ARM

Мета роботи: вивчити підходи формування сучасних архітектур на М-чипах компанії Apple

Теоретичні відомості

У цій лабораторній роботі ми досліджуємо еволюцію ARM-архітектур, розроблених компанією Apple, зокрема серії М-чипів – від М1 до М4. Ці системи-на-чипі SoC стали основою для нових поколінь MacBook, Mac mini, iPad Pro та інших пристроїв Apple, забезпечуючи значний приріст продуктивності та енергоефективності.

Apple M1, представлений у листопаді 2020 року, став першим чипом Apple, побудованим на ARM-архітектурі. Він об'єднує CPU, GPU, Neural Engine та інші компоненти в одному чипі, що дозволяє досягти високої продуктивності при низькому енергоспоживанні. М1 виготовлений за 5-нм технологією та має 8 ядер CPU – 4 продуктивних та 4 енергоефективних, до 8 ядер GPU та 16-ядерний Neural Engine.

Apple M2, анонсований у червні 2022 року, є наступником М1 і пропонує покращену продуктивність та підтримку більшої кількості пам'яті. Він виготовлений за 5-нм технологією другого покоління та має до 10 ядер GPU, що забезпечує до 35% кращу графічну продуктивність порівняно з М1.

Apple M3, представлений у червні 2023 року, виготовлений за 3-нм технологією та пропонує ще більшу продуктивність та енергоефективність. М3 має до 12 ядер CPU та до 18 ядер GPU, що забезпечує значний приріст продуктивності в порівнянні з попередніми поколіннями.

Apple M4, представлений у травні 2024 року, продовжує тенденцію до покращення продуктивності та енергоефективності. Він має до 16 ядер CPU та до 40 ядер GPU, що забезпечує ще більший приріст продуктивності в порівнянні з М3.

Порядок виконання роботи:

Провести порівняльний аналіз чотирьох поколінь М-чипів Apple (М1, М2, М3, М4) за наступними параметрами:

1. Технологічний процес: порівняти розмір транзисторів та їх вплив на продуктивність та енергоефективність.
2. Кількість ядер: оцінити кількість продуктивних та енергоефективних ядер у кожному поколінні.
3. Графічна продуктивність: порівняти кількість ядер GPU та їх вплив на графічну продуктивність.
4. Пам’ять: оцінити максимальний обсяг та тип пам’яті, підтримуваний кожним чипом.
5. Neural Engine: порівняти характеристики та продуктивність Neural Engine у кожному поколінні.

Результати аналізу слід представити у вигляді порівняльної таблиці.

Контрольні питання:

1. Які основні відмінності між чипами M1, M2, M3 та M4?
2. Як технологічний процес виготовлення чипів впливає на їх продуктивність та енергоефективність?
3. Яким чином збільшення кількості ядер у чипах впливає на їх багатозадачність та загальну продуктивність?
4. Як зміни в архітектурі GPU впливають на графічну продуктивність пристроїв?
5. Яку роль відіграє Neural Engine у загальній продуктивності чипів?
6. Як зміни в обсязі та типі пам’яті впливають на ефективність роботи пристроїв?
7. Які переваги та недоліки має кожне покоління чипів M у контексті реальних застосувань?

Література: [1-2, 9, 11]

Лабораторна робота №29-30. Порівняння сучасних архітектур. Оцінка переспектив розвитку

Мета роботи: розуміти особливості архітектур процесорів брендів та поколінь, навчитися характеризувати процесор за його основними параметрами, включаючи кількість ядер, сокети, техпроцес, стратегію побудови та набори інструкцій.

Теоретичні відомості

Сучасні процесори різних брендів демонструють значні відмінності в архітектурі та підходах до оптимізації продуктивності. Архітектури Intel Core, AMD Ryzen та Apple M-чипів використовують різні концепції побудови ядер, гібридні дизайни, системи взаємодії кешів та контролери пам’яті. Архітектура Intel Core останніх поколінь Raptor Lake та Arrow Lake поєднує продуктивні та енергоефективні ядра у гібридній моделі, що дозволяє досягти високої швидкодії в багатопоточних задачах при збереженні енергоефективності. Сокети LGA 1700 та LGA 1851 визначають сумісність процесора з материнською платою та впливають на можливості апгрейду.

Архітектура AMD Ryzen на базі Zen 2–Zen 5 використовує модульний підхід із системною шиною Infinity Fabric, яка забезпечує швидку взаємодію між ядрами, кешем та контролером пам’яті. З кожним поколінням збільшуються кількість ядер, пропускна здатність шини та ефективність енергоспоживання, що дозволяє масштабувати продуктивність без значного збільшення теплового пакету.

Архітектури Apple M1–M4 реалізовані як системи-на-чипі на ARM-ядрах, де CPU, GPU, Neural Engine та інші компоненти інтегровані в одному чипі. Використання продуктивних і енергоефективних ядер разом із високопродуктивними графічними та нейронними блоками дозволяє досягти високої продуктивності при мінімальному енергоспоживанні, що особливо важливо для мобільних та компактних пристроїв.

При порівнянні архітектур важливо звертати увагу не лише на кількість ядер чи частоту, але й на техпроцес, який визначає щільність транзисторів та

енергоефективність, а також на підтримку наборів інструкцій, які впливають на сумісність програмного забезпечення та оптимізацію алгоритмів. Знання цих характеристик дозволяє оцінювати перспективи розвитку процесорів і робити обґрунтований вибір платформи для різних типів задач.

Порядок виконання роботи:

1. Обрати представників архітектур Intel Core, AMD Ryzen та Apple M-чипів різних поколінь і зібрати дані про їхню кількість ядер, конфігурацію сокетів, техпроцес, стратегію побудови та підтримку наборів інструкцій.

2. Зробити порівняльний аналіз цих параметрів, щоб виявити сильні та слабкі сторони кожної архітектури та оцінити потенційні напрямки розвитку процесорів.

3. Скласти таблицю, де відобразити основні характеристики кожного процесора: ядра, сокети, техпроцес, стратегія побудови та набори інструкцій.

4. Зробити висновки щодо переваг і обмежень кожної архітектури та яка архітектура найбільш перспективна для різних типів застосувань.

5. Провести узагальнення отриманих результатів і сформулювати рекомендації щодо подальшого розвитку процесорних архітектур з урахуванням підвищення продуктивності, енергоефективності та сумісності з сучасними технологіями пам’яті та периферії.

Контрольні питання:

1. Які ключові відмінності між архітектурами Intel Core, AMD Ryzen та Apple M?

2. Як гібридна архітектура впливає на продуктивність і енергоефективність процесорів?

3. Яким чином техпроцес впливає на швидкодію та тепловий пакет?

4. Яку роль відіграють набори інструкцій у сумісності та оптимізації програмного забезпечення?

5. Які стратегії побудови ядер є найбільш ефективними для мобільних та стаціонарних платформ?

6. Які тенденції розвитку процесорів можна прогнозувати на основі аналізу сучасних архітектур?

7. Як конфігурація сокетів впливає на сумісність із материнськими платами?

8. Як різні архітектури впливають на можливості апгрейду та масштабування системи?

9. Які характеристики визначають перспективність архітектури для серверних, робочих і мобільних застосувань?

Література: [1-2, 6-7, 10]

СПИСОК РЕКОМЕНДОВАНИХ ДЖЕРЕЛ

1. Христинець Н. А. Архітектура комп’ютерів. Конспект лекцій для здобувачів першого (бакалаврського) рівня вищої освіти освітньо-професійної програми «Комп’ютерна інженерія» галузь знань 12 «Інформаційні технології» спеціальності 123 Комп’ютерна інженерія денної та заочної форм навчання. Луцьк: Луцький НТУ, 2025. 108 с.

2. Христинець Н. А. Архітектура комп’ютерів. Методичні вказівки до самостійної роботи для здобувачів першого (бакалаврського) рівня вищої освіти освітньої програми «Комп’ютерна інженерія» галузі знань 12 «Інформаційні технології» спеціальності 123 Комп’ютерна інженерія денної та заочної форм навчання. Луцьк: ЛНТУ, 2023. 24 с.

3. Chakraborty P. Computer Organisation and Architecture. New York: *Imprint Chapman and Hall*, 2024. 751p. URL: https://www.cse.iitd.ac.in/~srsarangi/archbook/archbook.pdf (Last accessed: 27.09.2025).

4. Ковальчук М. Л., Ушенко Ю. О., Угрин Д. І. Архітектура комп’ютерів: навчальний посібник. Чернівці: Чернівецький національний університет ім. Ю. Федьковича, 2022. 188 с. URL: http://surl.li/unqtg (дата звернення: 8.05.2025).

5. Задерейко О. В., Логінова Н. І., Трофименко О. Г. Комп’ютерна схемотехніка та архітектура комп’ютерів: навч.-метод. посіб. та ін. 2-ге вид. перероб. і допов. Одеса: Нац. ун-т «Одеська юридична академія», 2022. 288 с. URL: http://dspace.onua.edu.ua/handle/11300/22720 (дата звернення: 8.10.2025).

6. Крупельницький Л. В., Снігур А. В., Богомолов С. В. Архітектура комп’ютерів. Частина 1. Вінниця: ВНТУ, 2020. 104 с. URL: https://pdf.lib.vntu.edu.ua/books/IRVC/Krupelnitskij_P1_2020_104.pdf (дата звернення: 8.05.2024).

7. Що таке кеш процесора та як він впливає на продуктивність в іграх. *GeeksNEWS: огляди, новини технологій і компю'терного світу*. URL: https://surl.li/oaezco (дата звернення: 11.10.2025).

8. Bindal A. Fundamentals of Computer Architecture and Design and Edition. Springer, 2020. 592 p. URL: https://www.twirpx.com/file/2301543/ (date of access: 26.09.2025).

9. Гололобов Д.О. Основи комп'ютерної техніки та програмування мікропроцесорів: навчальний посібник. Київ: Видавничий центр Державного університету телекомунікацій, 2020. 58с. URL: https://www.twirpx.com/file/2966275/ (дата звернення: 14.10.2025).

10. McDowall J. Complex Enterprise Architecture: A New Adaptive Systems Approach Apress. 2019. URL: https://1lib.eu/book/3686670/a1f895 (дата звернення: 14.10.2025).

11. Автоматизація та комп'ютерні технології систем управління: наук.-допом. бібліогр. покажч. Нац. ун-т харч. технол. Київ, 2021. 171 с. URL: http://surl.li/unqxb (дата звернення: 30.09.2025).

12. Христинець Н. А., Михалик А. В., Міскевич О. І. Продуктивність технології Crossfire X при навантаженні відеоадаптерів мікропроцесорів AMD. *Науковий журнал «Комп'ютерно-інтегровані технології: освіта, наука, виробництво»*. Луцьк: Видавництво Луцький НТУ, 2020. Вип. 39. С. 213-217.

13. Христинець Н. А. Реалізація багатопоточності на архітектурі мультимедійних процесорів Nexperia. *Науковий журнал «Інформаційні технології та комп'ютерна інженерія»*. Вінниця: ВНТУ, 2022. С. 59-64.

14. Khrystynets N., Melnyk K., Lavrenchuk S., Miskevych O., Kostiuchko S. Multiprocessing as a Way to Optimize Queries. Advances in Transdisciplinary Engineering, 2024. №48. pp. 455–464. URL: https://doi.org/10.3233/ATDE231357 (дата звернення: 09.09.2025).

15. Smruti R. Basic Computer Architecture. Springer, 2020. 670 p. URL: https://www.cse.iitd.ac.in/~srsarangi/archbook/archbook.pdf (date of access: 10.09.2025).

Архітектура комп’ютерів: методичні вказівки до лабораторних занять для здобувачів першого (бакалаврського) рівня вищої освіти освітньо-професійної програми «Комп’ютерна інженерія» галузі знань 12 (F) «Інформаційні технології» спеціальності 123 (F7) Комп’ютерна інженерія денної та заочної форм навчання / уклад. Н. А. Христинець. Луцьк: ЛНТУ, 2025. 92 с.

Комп’ютерний набір: Н. А. Христинець
Редактор: Н. А. Христинець

Підпис до друку _____2025 р. Формат 60х84/16. Папір офс.
Гарн. Таймс. Ум. друк. арк.___
Тираж __ прим.

Інформаційно-видавничий відділ
Луцького національного технічного університету
43018, Луцьк-18, вул. Львівська,75.