МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Кафедра комп'ютерної інженерії

Дисципліна: Програмування
Спеціальність: 123 «Комп’ютерна інженерія»
Освітній рівень: Перший (бакалаврський)
Семестр: I

ЛАБОРАТОРНА РОБОТА № 4

Тема: Модульне програмування, низькорівневі порозрядні операції, бітові маски та командний процес розробки через Pull Request у Git

---

1. МЕТА РОБОТИ

- Опанувати принципи функціональної декомпозиції програм: проектування чистих підпрограм із чітким розділенням на прототипи (оголошення) та реалізацію (визначення).
- Дослідити механізм роботи автоматичних локальних змінних на стеку та використання специфікатора `static` для збереження внутрішнього стану функцій між послідовними викликами.
- Опанувати низькорівневе маніпулювання даними на рівні окремих розрядів за допомогою порозрядних операцій `&`, `|`, `^`, `~`, `<<`, `>>`.
- Засвоїти чотири канонічні інженерні операції над апаратними регістрами мікропроцесора: встановлення, скидання, інвертування та тестування стану бітів за допомогою динамічно сформованих масок.
- Опанувати командний процес інтеграції коду через створення, рецензування та злиття **Pull Request (PR)** на платформі GitHub.
- Підготувати підсумкову інженерну документацію та долучити **обов'язкову пояснювальну записку у форматі PDF** до звітності.

---

2. НЕОБХІДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

1. Текстовий редактор / середовище розробки: **Visual Studio Code** (із розширенням *C/C++* від Microsoft) або **CLion**.
2. Компілятор: **GCC** (GNU Compiler Collection).
3. Система контролю версій: **Git**.
4. Програма для підготовки технічної документації та експорту в PDF (**LaTeX**, **MS Word** або **LibreOffice Writer**).

---

3. ПОСЛІДОВНІСТЬ ВИКОНАННЯ РОБОТИ

Етап 0. Підготовка робочої гілки Git

1. Відкрийте термінал і перейдіть до кореневого каталогу вашого репозиторію:

```
cd ~/projects/programming-sem1
```

2. Переконайтеся, що ви перебуваєте в основній гілці `main`, та створіть нову робочу гілку `lab04`:

```
git checkout -b lab04
# або: git switch -c lab04
```

3. Створіть каталог для поточної лабораторної роботи та перейдіть у нього:

```
mkdir lab04
cd lab04
```

Етап 1. Функціональна декомпозиція та збереження стану (`task1.c`)

1. У каталозі `lab04` створіть файл `task1.c`.
2. Відповідно до свого номера варіанта з **Таблиці 1** реалізуйте модульну програму:
   - На початку файлу (до функції `main`) розмістіть обов'язкові **прототипи всіх функцій**.
   - Функція `main` повинна містити виключно інтерфейсну логіку (введення значень, виклик обчислювальних підпрограм та друк результатів).
   - Реалізуйте підпрограму розрахунку індивідуального завдання, яка приймає аргументи виключно за значенням (Pass-by-Value).

- Усередині допоміжної функції оголосіть **статичну локальну змінну (`static`)**, яка накопичує аналітичну інформацію (кількість викликів, середнє значення або екстремум) без використання глобальних змінних.

3. Скомпілюйте програму з контролем попереджень:

```
gcc -Wall -Wextra task1.c -o task1
./task1
```

4. Зафіксуйте створений сирцевий файл у робочій гілці Git:

```
git add task1.c
git commit -m "feat(lab4): implement modular function with static state
tracker"
```

Етап 2. Емуляція 32-бітного апаратного регістра GPIO (`task2.c`)

1. У каталозі `lab04` створіть файл `task2.c`.
2. Відповідно до свого номера варіанта з **Таблиці 2** реалізуйте набір незалежних підпрограм для маніпулювання окремими бітами умовного 32-бітного апаратного регістра (використовуйте беззнаковий тип `unsigned int` або `uint32_t` із `<stdint.h>`):
   - `set_bit`: примусове встановлення біта N в одиницю за допомогою операції побітового «АБО» з маскою (`1 << N`).
   - `clear_bit`: примусове скидання біта N у нуль за допомогою операції побітового «І» з інверсією маски (`~(1 << N)`).
   - `toggle_bit`: перемикання стану біта N за допомогою операції побітового «XOR» з маскою (`1 << N`).
   - `check_bit`: перевірка стану біта (повертає 1 або 0) за допомогою операції побітового «І».
   - `print_binary`: допоміжна функція, яка виводить стан усіх 32 бітів регістра в консоль у вигляді двійкового коду (з розбиттям по 4 або 8 бітів).
3. Продемонструйте у функції `main` роботу з бітами свого варіанта, виводячи двійковий стан регістра до та після кожної модифікації. Переконайтеся, що зміна цільового біта не порушує сусідні розряди.
4. Зафіксуйте код у Git:

```
gcc -Wall -Wextra task2.c -o task2
git add task2.c
git commit -m "feat(lab4): implement 4-canon hardware register bitwise
operations"
```

Етап 3. Пакування та розпакування телеметрії в бітове слово (`task3.c`)

1. У каталозі `lab04` створіть файл `task3.c` .
2. Відповідно до свого номера варіанта з **Таблиці 3** реалізуйте підпрограми для пакування декількох різних параметрів у єдине беззнакове слово:
   - Напишіть функцію пакування: за допомогою операцій зсуву ліворуч (`<<`) та побітового «АБО» (`|`) розташуйте кожне поле у відведеному діапазоні бітів.
   - Напишіть зворотну функцію розпакування: за допомогою відповідних бітових масок та зсувів праворуч (`>>`) виділіть кожне окреме значення назад у вихідні змінні.
3. Виведіть початкові параметри, сформоване слово пакета в шістнадцятковому форматі (HEX: `0x...`) та відновлені після розпакування параметри, підтвердивши їхню повну ідентичність.
4. Зафіксуйте зміни в Git:

```
gcc -Wall -Wextra task3.c -o task3
git add task3.c
git commit -m "feat(lab4): implement telemetry packet bitfield packing and
unpacking"
```

Етап 4. Оформлення звіту, відкриття Pull Request та злиття на GitHub

1. Підготуйте інженерну пояснювальну записку та збережіть її **виключно у форматі PDF** із точною назвою `Report_Lab04.pdf` у каталозі `lab04` .
2. Зафіксуйте звіт у робочій гілці та відправте гілку `lab04` на GitHub:

```
git add Report_Lab04.pdf
git commit -m "docs(lab4): add mandatory engineering PDF report"
git push -u origin lab04
```

3. Перейдіть на сайт GitHub у ваш репозиторій та створіть **Pull Request** із гілки `lab04` в гілку `main` :
   - Вкажіть заголовок: `Lab 4: Functions, Bitwise Operations and Register Emulation` .
   - В описі PR у форматі Markdown коротко зазначте мету роботи, свій варіант та перелік виконаних завдань.

4. Після перевірки або за вказівкою викладача виконайте злиття (**Merge pull request**) на сайті GitHub, після чого синхронізуйте локальну гілку `main`:

```
git checkout main
# або: git switch main
git pull origin main
```

---

4. ІНДИВІДУАЛЬНІ ВАРІАНТИ ЗАВДАНЬ

Номер варіанта визначається номером у списку академічного журналу групи:

$$\text{Варіант} = (\text{Номер у списку} - 1) \pmod{15} + 1$$

Таблиця 1. Варіанти до Завдання 1 (Модульність і static)

Вар.	Індивідуальне розрахункове завдання та роль статичної змінної
1	Функція розрахунку факторіала числа. Статична змінна рахує сумарну кількість операцій множення за всі виклики.
2	Функція обчислення N-го числа Фібоначчі. Статична змінна відстежує максимальне значення числа, яке будь-коли запитувалося.
3	Функція перевірки числа на простоту. Статична змінна рахує, скільки всього простих чисел було виявлено за сесію.
4	Функція перетворення температури з Цельсія у Фаренгейт. Статична змінна рахує кількість викликів із від'ємною температурою.
5	Функція обчислення гіпотенузи за двома катетами. Статична змінна накопичує середнє значення всіх розрахованих гіпотенуз.
6	Функція розрахунку суми цифр цілого числа. Статична змінна фіксує загальну кількість оброблених цифр за всі виклики.
7	Функція знаходження найбільшого спільного дільника (НСД). Статична змінна рахує сумарну кількість кроків алгоритму Евкліда.
8	Функція розрахунку кінетичної енергії $E = \frac{mv^2}{2}$. Статична змінна сигналізує, якщо швидкість перевищила швидкість звуку.
9	Функція обчислення площі трикутника за формулою Герона. Статична змінна рахує кількість викликів із виродженими трикутниками.
10	Функція генерації псевдовипадкового сигналу з шумом. Статична змінна зберігає внутрішній стан генератора (seed) між викликами.
11	Функція обчислення полінома $Y = AX^2 + BX + C$. Статична змінна рахує кількість проходів графіка через нуль ($Y == 0$).

Вар.	Індивідуальне розрахункове завдання та роль статичної змінної
12	Функція визначення відстані між двома точками на площині. Статична змінна накопичує сумарну пройдену довжину маршруту.
13	Функція округлення дробового числа до найближчого парного цілого. Статична змінна фіксує накопичену похибку округлення.
14	Функція перевірки парності та ділення числа на два. Статична змінна рахує кількість чисел, які виявилися непарними.
15	Функція обчислення опору паралельних резисторів. Статична змінна фіксує мінімальний еквівалентний опір за всю сесію.

Таблиця 2. Варіанти до Завдання 2 (Регістри та чотири канони)

Вар.	Номери бітів регістра, над якими виконуються операції
1	Встановити біт 3; скинути біт 7; інвертувати біт 12; перевірити стан біта 15.
2	Встановити біт 0; скинути біт 5; інвертувати біт 9; перевірити стан біта 31.
3	Встановити біт 4; скинути біт 8; інвертувати біт 14; перевірити стан біта 2.
4	Встановити біт 1; скинути біт 6; інвертувати біт 10; перевірити стан біта 18.
5	Встановити біт 7; скинути біт 2; інвертувати біт 15; перевірити стан біта 20.
6	Встановити біт 5; скинути біт 11; інвертувати біт 3; перевірити стан біта 8.
7	Встановити біт 2; скинути біт 9; інвертувати біт 16; перевірити стан біта 24.
8	Встановити біт 6; скинути біт 1; інвертувати біт 8; перевірити стан біта 30.
9	Встановити біт 10; скинути біт 4; інвертувати біт 13; перевірити стан біта 7.
10	Встановити біт 8; скинути біт 3; інвертувати біт 11; перевірити стан біта 19.
11	Встановити біт 12; скинути біт 0; інвертувати біт 5; перевірити стан біта 22.
12	Встановити біт 9; скинути біт 15; інвертувати біт 1; перевірити стан біта 16.
13	Встановити біт 11; скинути біт 6; інвертувати біт 4; перевірити стан біта 25.
14	Встановити біт 13; скинути біт 8; інвертувати біт 2; перевірити стан біта 27.
15	Встановити біт 14; скинути біт 10; інвертувати біт 6; перевірити стан біта 28.

Таблиця 3. Варіанти до Завдання 3 (Пакування телеметрії в 16/32-бітне слово)

Вар.	Структура полів для пакування в єдине беззнакове слово
1	[Біти 0..7] Код сенсора (0..255); [Біти 8..11] Режим роботи (0..15); [Біт 12] Прапорець тривоги (0..1); [Біти 13..15] Номер каналу (0..7).
2	[Біти 0..9] Значення АЦП (0..1023); [Біт 10] Готовність даних (0..1); [Біти 11..15] Температура чіпа від 0 до 31.
3	[Біти 0..3] Пріоритет (0..15); [Біти 4..11] Швидкість вентилятора (0..255); [Біти 12..13] Код помилки (0..3); [Біт 14] Живлення (0..1).
4	[Біти 0..6] Рівень освітленості (0..100); [Біт 7] Реле увімкнено (0..1); [Біти 8..15] Ідентифікатор пристрою (0..255).
5	[Біти 0..11] Напруга батареї в мілівольтах (0..4095); [Біти 12..14] Стан заряду (0..7); [Біт 15] Прапорець заряджання (0..1).
6	[Біти 0..4] День місяця (1..31); [Біти 5..8] Місяць (1..12); [Біти 9..14] Рік від 2000 (0..63); [Біт 15] Прапорець валідності дати.
7	[Біти 0..5] Секунди (0..59); [Біти 6..11] Хвилини (0..59); [Біти 12..16] Години (0..23); [Біт 17] Літній час (0..1) *(32-бітне слово)*.
8	[Біти 0..7] Швидкість вузла (0..200); [Біти 8..15] Кут повороту керма (0..255); [Біт 16] Гальма активні; [Біт 17] Фари увімкнено *(32 біти)*.
9	[Біти 0..9] Дистанція до перешкоди (0..1023); [Біти 10..13] Тип перешкоди (0..15); [Біт 14] Екстрена зупинка (0..1).
10	[Біти 0..7] Червоний канал кольору R (0..255); [Біти 8..15] Зелений канал G (0..255); [Біти 16..23] Синій канал B (0..255) *(24 біти в 32)*.
11	[Біти 0..3] Версія протоколу (0..15); [Біти 4..11] Довжина корисного пакета (0..255); [Біти 12..15] Контрольна сума CRC (0..15).
12	[Біти 0..10] Висота польоту (0..2047 м); [Біт 11] Двигун запущено; [Біти 12..15] Рівень сигналу GPS від 0 до 15.
13	[Біти 0..5] Рівень звуку в дБ (0..63); [Біт 6] Мікрофон увімкнено; [Біти 7..14] Частотний діапазон (0..255); [Біт 15] Шумозаглушення.
14	[Біти 0..8] Атмосферний тиск (зміщення від норми); [Біти 9..14] Вологість повітря (0..100%); [Біт 15] Сигнал штормового попередження.
15	[Біти 0..3] Код операції АЛП (0..15); [Біти 4..9] Номер першого регістра; [Біти 10..15] Номер другого регістра *(формат команди CPU)*.

---

5. ВИМОГИ ДО ЗВІТНОСТІ ТА ЗДАЧІ РОБОТИ

Робота захищається очно в комп'ютерному класі.

Обов'язковою умовою допуску до очного захисту є наявність у репозиторії на GitHub

пояснювальної записки у форматі PDF (`Report_Lab04.pdf`), яка містить титульний аркуш, індивідуальні формулювання, опис масок і розрахунків, повні лістинги програм `task1.c` , `task2.c` , `task3.c` , скріншоти тестування та висновки.

Структура репозиторію на GitHub перед захистом повинна мати вигляд:

```
programming-sem1/
├── .gitignore
├── README.md
├── lab01/
├── lab02/
├── lab03/
└── lab04/
    ├── task1.c
    ├── task2.c
    ├── task3.c
    └── Report_Lab04.pdf
```

Примітка щодо GitHub: Викладач перевіряє наявність відкритого та злитого **Pull Request** на сторінці репозиторію зі статусом `Merged` .

---

6. КОНТРОЛЬНІ ЗАПИТАННЯ ДЛЯ ОЧНОГО ЗАХИСТУ

1. Чому компілятору мови С необхідний прототип функції, якщо тіло підпрограми описане нижче у тому самому файлі?
2. Що таке стековий кадр (Stack Frame) і які апаратні регістри процесора фіксують його межі під час виклику функції?
3. Чим фізично відрізняється локальна змінна зі специфікатором `static` від звичайної автоматичної локальної змінної на стеку?
4. Чому спроба змінити значення змінної всередині функції через передачу за значенням (Pass-by-Value) не змінює оригінальну змінну в головній програмі?
5. Як за допомогою побітових операцій зсуву та оператора «HI» сформувати маску для гарантованого скидання обраного біта регістра в нуль?
6. Чому операцію перевірки стану прапорця виконують через побітове «І», а не через перевірку на рівність усьому числу?
7. У чому полягає апаратна небезпека циклу Read-Modify-Write при роботі з регістрами мікроконтролера в умовах апаратних переривань?

8. Що таке Pull Request у системі контролю версій Git і яку інженерну роль він відіграє в процесі рецензування коду (Code Review)?