**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
Київський національний університет будівництва і архітектури

# ОСНОВИ КОМП'ЮТЕРНО-ІНТЕГРОВАНОГО УПРАВЛІННЯ

Методичні вказівки
до виконання лабораторних і практичних робіт
для здобувачів першого (бакалаврського) рівня вищої освіти
за спеціальністю 174 «Автоматизація, комп'ютерно-інтегровані технології та робототехніка»

Київ 2025

УДК 62–5 [075.8]

О-75

Укладач М.І. Самойленко, асистент

Рецензент В.Ю. Луценко, канд. техн. наук, доцент

Відповідальний за випуск А.В. Запривода, канд. техн. наук, доцент

*Затверджено на засіданні кафедри автоматизації технологічних процесів, протокол № 5 від 2 грудня 2024 року.*

В авторській редакції.

**Основи** комп’ютерно-інтегрованого управління : методичні
О75 вказівки до лабораторних і практичних робіт / уклад. М.І. Самойленко. – Київ : КНУБА, 2025. – 44 с.

Містять вимоги до змісту лабораторних та практичних робіт з курсу «Основи комп’ютерно-інтегрованого управління».

Призначено для здобувачів першого (бакалаврського) рівня вищої освіти за спеціальністю 174 «Автоматизація, комп’ютерно-інтегровані технології та робототехніка».

# ЗАГАЛЬНІ ПОЛОЖЕННЯ

**Метою** виконання лабораторних та практичних робіт є закріплення студентами набутих під час вивчення дисципліни «Основи комп'ютерно-інтегрованого управління» теоретичних знань.

Виконання кожної лабораторної роботи передбачає написання та відлагодження програм в середовищі AVR Studio від ф. Atmel та їх запис в пам'ять контролера для перевірки роботи на лабораторному макеті.

Під час виконанні лабораторних робіт рекомендується використовувати в якості додаткових літературних джерел офіційний інтернет-сайт виробника контролерів https://www.microchip.com/, а також інтернет-сайт виробника лабораторних макетів *www.open-system.com*. Інтернет-сайти містять необхідну інформацію про AVR–контролери та периферійні пристрої, котрі входять до складу лабораторних макетів.

До курсу лабораторних робіт включені типові задачі, що виникають у процесі розробки мікропроцесорних систем керування, такі як взаємодія контролера зі статичним, динамічним та РК-дисплеями; внутрішніми та зовнішніми периферійними пристроями. Розглянуті задачі вводу-виводу дискретних сингалів, реалізація АЦП, перетворення ширини імпульсу на цифровий код, формування аналогових сигналів керування. Також розглянуті типові інтерфейси міжелементного послідовного обміну даними та послідовний асинхронний інтерфейс, їх фізична суть та програми роботи з ними.

Виконуючи лабораторні роботи рекомендується використовувати середовище розробки AVR Studio та мову асемблер чи С, що підтримує дане середовище. Середовище AVR Studio безкоштовно доступне на сайті виробника контролерів. За бажанням студента допускається використання інших засобів розробки та програмування AVR контролерів, наприклад, CodeVision AVR чи інші.

Для відлагодження програм рекомендується використовувати вбудований симулятор AVR Studio та лабораторні макети. Під час виконання лабораторних робіт онлайн (в разі переходу на дистанційну форму навчання) – відлагодження та перевірку роботи програм рекомендується проводити в програмі-симуляторі Proteus (є доступна пробна версія на сайті виробника); за бажанням студента дозволяється використовувати інші симулятори, в тому числі онлайн.

## *Лабораторна робота №1*

**Тема:** *схемні та програмні методи керування роботою семисегментного індикатора. Статична та динамічна індикація.*

**Мета:** *вивчити схеми статичної та динамічної індикації. Навчитись керувати роботою статичного та динамічного семисегментного індикатора.*

## *Хід роботи*

1. Вивчити принцип роботи динамічного та статичного методів відображення інформації.
2. Скласти та відлагодити програму, що відображає на статичному індикаторі число, наприклад, результат математичної дії з числами, що записані в регістри R16 та R17. Мікро-ЕОМ виконує дії з числами в шістнадцятирічному форматі, тому в програмі слід передбачити процедуру переводу чисел зі шістнадцятирічного в десятинний формат.
3. Записати програму в пам’ять мікро-ЕОМ та перевірити роботу програми на лабораторному макеті.
4. Доповнити програму п.2 підпрограмою, що керує роботою динамічного індикатора та відображає на ньому теж саме число.
5. Відлагодити програму з використання засобів AVR Studio, завантажити програму в пам’ять мікро-ЕОМ та перевірити її роботу на лабораторному макеті.

## Короткі теоретичні відомості

Для відображення цифрової інформації широко застосовуються семисегментні індикатори, в котрих для відображення контуру цифри

використовують сім лінійних світлодіодів, розміщених у вигляді цифри «8».

На основі таких індикаторів та додаткових світлодіодів будують дисплеї в мікропроцесорних системах керування ТП, для відображення інформації використовують статичний та динамічний методи індикації.

**Статична індикація** – кожний з розрядів індикатора постійно підсвічується від свого джерела інформації (рис. 1.1).

Рис. 1.1. Схема статичного індикатора

За такої схеми кожний розряд індикатора HL1-n підключається через свій дешифратор DC1-n та регістр RG1-n до шини даних контролера, вибір регістра здійснюється за допомогою селектора адреси SA. Апаратні затрати за такої реалізації становлять n-пар регістр +дешифратор для кожного з n-розрядів індикатора.

**Динамічна індикація** – розряди індикатора почергово в циклі підключаються до джерела інформації через загальну для них шину даних (див. рис. 1.2).

Адреса розряду індикатора зберігається в регістрі адреси RA, вибір розряду здійснює дешифратор DA, дані для відображення знаходяться в регістрі RD.

Під час динамічної реалізації значно менші апаратні затрати, однак необхідно забезпечити достатній час «підсвітки» кожного з розрядів для уникнення блимання та забезпечення потрібної яскравості світіння дисплею. Переваги даного методу зростають у разі збільшення кількості розрядів індикатора.

На стенді реалізовано обидва види індикаторів. Статична індикація – доступна як до зовнішньої пам’яті за адресою 0×A000

(ліва пара розрядів) та 0×B000 – права пара; наприклад, команда: st 0×A000, R17 запише в індикатор число, що було в регістрі R17.

Для відображення числа на динамічному індикаторі необхідно записати в регістр даних код символу числа, наприклад, для відображення символу «1» розряди a та b повинні бути лог. 1, решта лог. 0, код 0000 0011; для відображення символу «2» – a,b,g,e,d=1, решта – 0, код 0101 1011 (в двійковому коді, молодший біт праворуч) аналогічно – решта символів. Потім в регістр адреси розряду слід записати адресу того розряду, в котрому цей символ слід відобразити: перший, другий, і т. д., 0000 – перший, 0001 – другий і, відповідно, решта розрядів (на макеті розрядів 4).

Запис в регістри здійснюється аналогічно до запису в зовнішню пам’ять командою ST: за адресою 0×8001 – код символу, за адресою 0×8002 – адреса розряду, наприклад:

```
ldi R17,0x03
ST 0x8001,R17
ldi R17,0
ST 0x8002,R17    ; – завантажити в регістр даних 3 (код символу «1») та відобразити його в молодшому розряді індикатора.
```

Рис. 1.2. Схема динамічного індикатора

Для перетворення числа зі шістнадцятирічного в двійково-десятковий формат необхідно з вхідного числа виділити розряди сотні (ціле від ділення на 100), потім десятки – поділивши залишок від ділення на 100 на 10; ціле від ділення на 10 – десятки, залишок – одиниці. Контролер AVR не має команди ділення, тому ділення замінимо на віднімання, наприклад, як в процедурі перетворення, алгоритм котрої наведено на рис. 1.3. На початку процедури перевіряємо, чи більше вхідне число від 100, якщо так – віднімемо від нього 100 та збільшимо лічильник сотень на 1, та знову повторимо перевірку, більше – знову віднімемо 100 та збільшимо лічильник сотень на 1, і так доти, доки число залишатиметься більшим 100. Якщо число менше 100, порівняємо з 10, та аналогічно будемо віднімати 10 та збільшувати лічильник десятків доти, доки число не стане менше 10. Вміст лічильника сотень – розряд сотні, вміст лічильника десятків – десятки, залишок – одиниці.

Статичний індикатор на лабораторному макеті передбачає одночасне відображення двох розрядів – тисячі і сотні за адресою доступу 0×a000 та десятки і одиниці за адресою 0×b000, тому десятки та одиниці слід об’єднати в один байт, в якому старша тетрада – десятки, молодша – одиниці. Для цього поміняємо місцями тетради в десятках (результат перетворення займає лише молодшу тетраду), потім додамо молодшу тетраду одиниць – отримаємо байт у складі: старша тетрада – десятки від перетворення, молодша – одиниці. Такий байт можна безпосередньо записати в індикатор для відображення:

```
Swap D10          ;поміняти тетради в D10 місцями
Add D10, D1       ;додати одиниці, результат в D10
Sts 0xb000, D10   ;записати за адресою індикатора.
```

Рис. 1.3. Підпрограма перетворення 16-річного числа
на двійково-десятковий. Схема алгоритму

Під час відображення чисел на динамічному індикаторі в регістр даних індикатора записується код символу для кожного розряду, тому подібна операція непотрібна.

## *Лабораторна робота №2*

**Тема:** *керування засобами вводу інформації. Читання клавіатури.*

**Мета:** *закріпити знання з теоретичного курсу, навчитись зчитувати сигнали з дискретних датчиків.*

## *Хід роботи*

1. В середовищі AVR Studio скласти та відлагодити програму, що включає в себе:

1.1) підпрограму читання клавіатури;

1.2) підпрограму усунення брязкоту контактів;

1.3) підпрограму ідентифікації натиснутої кнопки;

1.4) підпрограму відображення символу натиснутої кнопки на дисплеї.

2. Записати програму в пам’ять ЕОМ та перевірити її роботу на лабораторному макеті.

3. Доповнити програму можливістю зчитування кількох кнопок з одночасним відображенням на дисплеї чотирьох символів натиснутих кнопок, наприклад, зі зсувом вліво, подібно до вводу числа в калькуляторі.

4. Записати програму в пам’ять ЕОМ та відлагодити роботу на макеті.

## Короткі теоретичні відомості

Для вводу інформації широко використовуються кнопкові перемикачі та контактні клавіатури. Сигнал з них формується замиканням контактної пари та супроводжується брязкітом контактів, тривалість брязкоту залежить від механічних особливостей клавіатури і становить близько 8..15мс. Для усунення брязкоту контактів використовують апаратні та програмні засоби. Апаратні – фільтри-формувачі сигналу, частіше це RS-тригери; програмні – повторне опитування контакту із затримкою, і в разі збігу результату кнопка вважається натиснутою, інакше – був брязкіт чи помилкове зчитування.

Під час зчитування клавіатури слід враховувати високу швидкодію контролера, зчитування відбувається десятки разів за секунду, тому для уникнення повторного зчитування однієї кнопки часто необхідно чекати готовності клавіатури – поки кнопка буде відпущена чи жодна з кнопок не натиснута.

Для зменшення кількості електричних з’єднань на платі контролера клавіатуру об’єднують в матрицю, на макеті – 4×3, три стовпчики по чотири кнопки в рядку (рис. 2.1).

Рис. 2.1. Схема підключення клавіатури

Клавіатура доступна для читання як зовнішня пам’ять за адресами 0×9006 – перший стовпчик, 0×9005 – другий стовпчик та 0×9003 – третій. Чотири кнопки, при цьому, читаються як молодша тетрада в байті, біти 0..3, для першого стовпчика адреса 0×9006, це кнопки з символами «1», «4», «7», «*»; для другого адреса 0×9005 – «2», «5», «8», «0»; для третього адреса 0×9003 – «3», «6», «9», «#». У разі натиснутої кнопки відповідний біт в тетраді буде рівний нулю, решта лог. 1, якщо результат зчитування всі одиниці – кнопка в цьому стовпчику не натиснута.

Підпрограма читання клавіатури, алгоритм якої наведено на рис. 2.2, почергово зчитує дані з трьох стовпчиків, порівнює з 0×0F (всі одиниці в молодшій тетраді), якщо так – кнопки не натиснуті, переходимо до наступного стовпчика; інакше – повторна перевірка для усунення дріб’язку. Кнопка нажата – визначаємо її номер та символ; було помилкове спрацювання – переходимо до наступного стовпчика.

Приклад підпрограми для визначення номера кнопки та символу, зображеному на ній:

```
Key_D:
        set             ;встановимо прапорець T
        com a           ;побітова  інверсія A
        clc             ;очистить C
        clr Key         ;------ Key---
k1:     inc Key         ;лічильник біту Key=Key+1
        ror a           ;зсунемо код нат. клавіші вправо
        brcc k1         ;перенос був? Ні повторить!
        ldi tmp2, 04    ;кнопок 4
        mul row1, tmp2  ;помножимо номер стовпчика на 4
```

```
sub r0,Key      ;віднімемо номер натиснутої кнопки
mov Zl,r0       ;отримали адрес коду в таблиці
lpm             ;читаємо код кнопки з таблиці
mov tmp2,r0     ;значення в tmp2
ret
```

Рис. 2.2. Процедура читання клавіатури. Схема алгоритму

Спочатку помножимо послідовний номер стовпчика на чотири, потім віднімемо номер кнопки, отримане число – адреса символу в таблиці.

## *Лабораторна робота №3*

**Тема:** *використання модулів на рідких кристалах в якості знакосинтезуючого символьного дисплею.*

**Мета:** *освоїти прийоми роботи з РК – модулем, навчитись використовувати різні режими роботи модуля, закріпити знання з теоретичного курсу.*

### *Хід роботи*

1. В середовищі AVR Studio скласти та відлагодити програму, що реалізує взаємодію контролера з РК-модулем. Програма повинна включати в себе:

1.1) підпрограму ініціалізації РК-модуля;

1.2) підпрограму, що виводить на екран повідомлення, рядок символів;

1.3) підпрограму, що демонструє різні режими відображення курсору;

1.4) підпрограму, що відображає символьні імена та числові значення параметрів, наприклад: R16=12, R17=3, R16*R17=36;

1.5) підпрограму створення власного символу, наприклад, «Я», ± чи інший.

2. Записати програму в пам’ять ЕОМ та перевірити її роботу на лабораторному макеті, в разі потреби удосконалити програму.

## Короткі теоретичні відомості

В якості символьних дисплеїв широко використовуються РК-модулі, побудовані на базі контролера ф. Hitachi HD44780, він став стандартом для більшості виробників модулів. Перед використанням РК-модуля, як і решту периферійних пристроїв, необхідно виконати його ініціалізацію, схему алгоритму ініціалізації модуля наведено на рис. 3.1.

Рис. 3.1. Ініціалізація РК-модуля. Схема алгоритму реалізації

Під час використання модуля слід враховувати його невисоку швидкодію, тому для коректної роботи необхідно застосовувати програмні затримки.

На лабораторному макеті РК-модуль доступний як зовнішня пам’ять для запису команд за адресою 0×8004 та для запису даних за адресою 0×8005. Службові сигнали E, RS, R/W формуються самостійно апаратною частиною, це спрощує написання програм. Читання прапорця Busy (модуль зайнятий внутрішніми операціями) на макеті не передбачено, тому для коректної роботи після кожної

операції з РК-модулем необхідно передбачити затримку у часі (близько 2 мс) для виконання внутрішніх команд всередині модуля.

Система команд РК-модуля наведена в табл. 3.1.

*Таблиця 3.1*

**Система команд РК-модуля HD44780**

| **Команда** | **R S** | **R/ W** | **D7** | **D6** | **D5** | **D4** | **D3** | **D2** | **D1** | **D0** | **Опис** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Очис-тить дисплей | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | Очищує пам’ять DD шляхом запису пробілів $20 та встановлює лічильник пам’яті в початкове значення $00 |
| Курсор на початок | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | * | Встановлює курсор на адресу $00 (початок). Всі відміняє зсуви тексту |
| Режим вводу даних | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | I / D | S | Інкремент (I/D=1) або декремент (I/D=0) лічильника адреси на 1 під час операції запису в пам’ять DD/CG чи читанні з неї. Активізується курсор та функція мерехтіння. Якщо S=1, то вміст дисплея зсувається вправо (I/D=1) чи вліво (I/D=0) |
| Настройка дисплея | 0 | 0 | 0 | 0 | 0 | 0 | 1 | D | C | B | Включення (D=1) чи виключення (D=0) дисплея. Відображання курсору (C=1) чи курсор скритий (C=0). При B=1 символ, на котрий вказує курсор, мигає |

*Закінчення табл. 3.1*

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Зсув дисплея чи курсору | 0 | 0 | 0 | 0 | 0 | 1 | S / C | R / L | * | * | S/C=0, R/L=0 – зсув курсору вліво (AC=AC-1); S/C=0, R/L=1 – курсор зсувається вправо (AC=AC+1); S/C=1, |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | R/L=0 – текст на табло зсувається вліво; S/C=1, R/L=1 – текст на табло зсувається вправо |
| Ініціалізація | 0 | 0 | 0 | 0 | 1 | DL | N | F | * | * | DL=1 – 8-розрядна шина даних; DL=0 – 4-розрядна шина даних; N=1 – дворядковий дисплей; N=0 – однорядковий; F=0 –матриця 5x8, F=1 матриця 5×10 |
| Встановлення адреси пам'яті CG | 0 | 0 | 0 | 1 | ACG | | | | | | Встановити значення в лічильник адреси CG, 6-бітна адреса ACG |
| Встановлення адреси пам'яті DD | 0 | 0 | 1 | ADD | | | | | | | Встановити значення в лічильник адреси DD, 7-бітну адресу ADD |
| Читання прапорця зайнятості | 0 | 1 | BF | AC | | | | | | | Зчитати прапорець зайнято та вміст лічильника пам'яті (CG чи DD) встановлюється попередньою командою |
| Запис байту в пам'ять DD чи CG | 1 | 0 | Байт даних | | | | | | | | Запис байту даних в пам'ять CG чи DD, адресу котрої було встановлено попередньою командою |
| Читання байту з пам'яті DD чи CG | 1 | 1 | Байт даних | | | | | | | | Читання байту даних з пам'яті, адреса була встановлена попередньою командою |

Приклад підпрограми запису команди:

```
SendCom:        ; Передача команди RS=0 формується апаратно
```

```
        sts   0x8004, prm1; під час запису за адресою 0×8004
        ret
```

Приклад підрограми запису даних:

```
SendDat:        ; Передача байту даних RS=1 формується апаратно
        sts   0x8005, prm1      ; під час запису за адресою 0×8005
        ret
```

Приклад підпрограми, що виводить на дисплей рядок символів:

```
OutText:        ; Вивід рядка символів  на табло
        lsl   ZL
        rol   ZH            ; Покажчик Z – на 1 позицію вліво
OT1:
        lpm                 ; Загружаємо код символу в r0
        mov   prm1,r0       ; Копіюємо код символу в prm1
        rcall WaitBusy      ; чекаємо готовності РК-модуля до прийому
        sts   0x8005, prm1      ; Передаємо символ в РК-модуль
        adiw  Z,1           ; Інкрементуємо покажчик Z
        dec   cnt           ; Лічильник – 1
        brne  OT1           ; Виводимо всі символи на РК-табло
        ret
```

Процедура створення власних символів:

```
DefChar:    ; Визначення символів користувача
        lsl   prm1          ; Помножити prm1 на 2
        lsl   prm1          ; Помножити prm1 на 4
        ldi   ZH,high(CharTab)  ; Адреса ст. байта шаблона символу
        lsl   prm1          ; prm1 x 8
        mov   ZL,prm1       ; Z -> адреса символу
        rol   ZH            ; Покажчик  Z – на 1 позицію вліво
        sbr   prm1,1<<6     ; Розряд 6 = 1 -> адреса в пам’яті CG
        rcall WaitBusy      ; Чекаємо готовності РК-модуля прийому
        rcall SendCom       ; Встановимо лічильник адреси пам’яті CG
        ldi   cnt,8
DC1:
        lpm                 ; Завантажуємо в r0 шаблон символу
        mov   prm1,r0       ; Копіюємо шаблон символу в prm1
        rcall WaitBusy      ; Чекаємо готовності РК-модуля до прийому
        rcall SendDat       ; Передаємо шаблон, автоінкремент адреси
        adiw  ZL,1          ; Ікремент покажчика Z
        dec   cnt           ; Лічильник  – 1
        brne  DC1           ; Копіюємо всі 8 символів в пам’ять CG
        ret
```

## *Лабораторна робота №4*

**Тема:** *реалізація АЦП з використанням ЦАП та компаратора.*

**Мета:** *закріпити знання з теоретичного курсу.*

## *Хід роботи*

1. В середовищі AVR Studio скласти та відлагодити програму, що включає в себе підпрограму реалізації АЦП методом послідовного наближення та підпрограму відображення результатів на семи-сегментному дисплеї.

2. Записати програму в пам’ять ЕОМ та перевірити її роботу, в разі потреби удосконалити програму.

3. Визначити похибку в роботі АЦП на всьому діапазоні вхідних сигналів.

4. Доповнити програму (п.1) підпрограмою реалізації АЦП методом половинного ділення (порозрядного зважування, як його також називають), відлагодити програму в середовищі AVR Studio.

5. Записати програму в пам’ять ЕОМ перевірити роботу.

6. Оцінити похибку АЦП під час обох методів реалізації, зробити висновки.

## Короткі теоретичні відомості

Реалізація АЦП з використанням ЦАП та компаратора.

Порівняно простий метод реалізації АЦП, для реалізації 8-бітного АЦП на макеті використано ЦАП з паралельним входом, доступ до нього як до зовнішньої пам’яті з адресою 0×E000H.

Суть методу: контролер почергово записує дані в ЦАП, котрий їх перетворює на аналоговий сигнал Uout, компаратор порівнює вхідну напругу з результатом перетворення та видає результат лог. 1 при Uout<Ux, та лог. 0 при Uout >Ux (рис. 4.1).

Рис. 4.1. Структурна схема реалізації АЦП з допомогою ЦАП та компаратора

Контролер знаходить межу перемикання компаратора (напруги однакові), а код, записаний в ЦАП, і буде результатом АЦ перетворення.

Для пошуку межі перемикання використовують алгоритми послідовного наближення та половинного ділення (порозрядного зважування), один з них чи обидва у поєднанні.

**Послідовне наближення**

Контролер почергово зарисує в ЦАП числа від 0 до FF з кроком 1 та перевіряє біт порту, котрий з'єднаний з виходом компаратора. Межа перемикання виходу компаратора з 1 на 0 є ознакою, що перетворення закінчено, а код, що був записаний в компаратор, є результатом АЦП.

Даний метод простий в реалізації, однак досить тривалий у часі. Для реалізації навіть 8-бітного АЦП потрібно від 1 до 255 кроків перебору коду, кожен крок включає в себе час на передачу даних в ЦАП, час їх обробки в ЦАП та час для формування вихідної напруги з ЦАП.

На протязі часу перетворення мікроЕОМ задіяна повністю, а вхідна напруга Ux повинна бути незмінною, тому даний метод використовується лише у випадку керування «повільними» процесами (рис. 4.2).

Рис. 4.2 Алгоритми реалізації:
А – послідовного наближення; Б – половинного ділення

Підпрограма реалізації алгоритму послідовного наближення

```
go: clr Res
go1:inc Res
    ;in tmp, PinB;$16; PortB вхід з компаратора
    sts 0xf000,Res          ; видаємо чисто в ЦАП
    rcall Wait              ; почекаємо повільну ЦАП
    sbic PinB,7 ; результат досягнуто вих. ;компаратора =0
    ;sbrc tmp,7
    rjmp go1                ;1: продовжимо GO1
```

```
mov Resulr,Res   ; 0: досягли, обробка результату
ret
```

**Половинне ділення** (порозрядне зважування).

Інтервал перетворення ділиться навпіл (старший біт=1 решта =0), перевіряємо, чи переключився компаратор, якщо так, напруга Ux менша, біт=0, перевіряємо наступний. Якщо компаратор не переключився, отже результат менший; додаємо до старшого біту =1 сусідній молодший та перевіряємо і так кожний розряд.

Такий алгоритм дозволяє виконати 8-бітне перетворення за 8 ітерацій, що значно швидше від попереднього методу.

Підпограма реалізації:

```
do: ldi step,0x80 ; встановимо старший байт=1
    clr Res       ; очистимо регістр результату
    ldi cnt, 8    ; 8-біт, встановимо лічильник
do1:              ; робочий цикл
    add Res,Step      ; додамо крок обчислення до результату
    sts 0xf000,Res    ; в ЦАП
    rcall Wait_l      ; зачекаємо обробку в МС ЦАП
    sbis PinB,7       ; перевіримо вихід компаратора  =1?
    sub Res,Step      ; перескочили, отже обнулить поточний біт
    clc               ; очистить C для коректного зсуву
    lsr     Step      ; змістити біт вправо наступний молодший біт
    dec cnt           ; зменшить лічильник ітерацій
    brne do1          ; якщо не закінчено – продовжить
    mov Result,Res    ; зберегти результат
    ret               ; вихід
```

## *Лабораторна робота №5*

**Тема:** *реалізація перетворення сигналів типу «ширина імпульсу» та «частота» в цифровий код.*

**Мета:** *навчитись використовувати різні режими роботи таймерів/лічильників та закріпити знання з теоретичного курсу.*

Сигнали типу «частота імпульсу» умовно розділяють на «швидкі» та «повільні», для перетворення їх на цифровий код використовують різні методи. «Швидкі» – підраховують кількість вхідних імпульсів протягом заданого інтервалу часу, у якості задатчиків часу використовують таймери. Для перетворення «повільного» сигналу навпаки, протягом вхідного періоду (чи імпульсу) таймер/лічильник підраховує тактові імпульси, безпосередньо чи через внутрішній перед дільник (таймер працює в режимі лічильника). Дана робота складається з двох частин: перетворення «повільних» сигналів та «швидких».

## *Хід роботи*

1. В середовищі AVR Studio скласти та відлагодити програму, котра:

   1.1) налаштовує таймер 0 для роботи в режимі таймера-задатчика часових інтервалів;

   1.2) підраховує імпульси на вході мікро-ЕОМ протягом заданого інтервалу часу;

   1.3) обробляє отримані значення (усереднення результатів);

   1.4) відображає результат на дисплеї.

2. Завантажити програму в пам’ять контролера та первірити роботу програми на лабораторному макеті. Визначити швидкість обертання валу двигуна (об./хв) під час включення двигуна на максимальну напругу (лог. 1).

3. В середовищі AVR Studio скласти та відлагодити програму, котра:

3.1) налаштовує таймер 1 для роботи в режимі лічильника тактових імпульсів;

3.2) по фронту вхідного сигналу на вході мікро-ЕОМ запускає лічильник 1, котрий підраховує тактові імпульси протягом високого рівня вхідного сигналу;

3.3) по спалу вхідного сигналу зупиняє таймер/лічильник 1 та обробляє отримані значення (усереднення результатів);

3.4) відображає результат на дисплеї.

4. Завантажити програму в пам’ять контролера та перевірити роботу програми на лабораторному макеті. Визначити швидкість обертання валу двигуна (об./хв) під час включення двигуна на максимальну напругу (лог. 1).

5. Порівняти результати перетворень обома методами, зробити висновки.

## Опис лабораторного макету

Для виконання даної роботи необхідно доповнити макет платою розширення для систем автоматизованого керування. Встановити на платі перемички JP1 в положення «MOTOR» – керування двигуном; JP2 в положення «PWM» – широтно-імпульсне керування; JP3 в положення «REG» – регулювання частоти обертання двигуна. Керування обертами двигуна в даній роботі не проводиться, двигун просто вмикають біт 2 порту B встановлюють рівним 1(PortB.2=1). У якості датчика обертів використовуємо U1 на основі датчика Холла, він же джерело імпульсів. Сигнали з датчика подаються на PortD.4 (четвертий розряд порту D), а також відображаються світлодіодом VD1, що розміщений на платі розширення біля двигуна.

## Ініціалізація таймера 0 для роботи в режимі таймера за датчика часових інтервалів

Для запуску таймерів в AVR необхідно вказати джерело вхідних імпульсів для підрахунку: це можуть бути зовнішні імпульси на відповідному вводу контролера чи імпульси власного тактового генератора безпосередньо тактової частоти чи через дільник. Для цього необхідно встановити відповідні біти в регістрі TCCR0 (табл. 5.1).

*Таблиця 5.1*

**Біти налаштування перед дільника таймерів**

| Біти регістра TCCR0 | | | Вхід таймера 0 |
|---|---|---|---|
| CS02, біт2 | CS01, біт1 | CS00, біт0 | |
| 0 | 0 | 0 | Сигналу немає, таймер не працює |
| 0 | 0 | 1 | Таймер підраховує тактові імпульси (Ф) |
| 0 | 1 | 0 | Підраховує ТІ, ділені на 8 (Ф/8) |
| 0 | 1 | 1 | Підраховує ТІ, ділені на 64 (Ф/64) |
| 1 | 0 | 0 | Підраховує ТІ, ділені на 256 (Ф/256) |
| 1 | 0 | 1 | Підраховує ТІ, ділені на 1024 (Ф/1024) |

Бувають випадки, коли для формування заданих інтервалів часу потрібні менші інтервали, ніж повний регістр таймера (від 00 до 255), тоді в рахунковий регістр таймера записують потрібне число, «перед установка»; рахунок, при цьому, почнеться не з 0, а з записаного числа.

У разі переповнення таймера (перехід результату підрахунків з 0FF до 00) апаратно генерується переривання, підрахунок продовжується з 00 в лічильнику; апаратна передустановка непередбачена, тому для наступних використань таймера необхідно в його рахунковий регістр записати потрібне число передустановки.

Якщо в програмі передбачено використання переривань по переповненню таймера, то слід встановити відповідну маску в регістрі масок TIMSK (біт 1) та дозволити глобальні переривання командою SEI.

```
ldi Time, 0          ; значення константи
in tmp,tccr0         ; завантажимо tccr0 в tmp
ori  tmp, 05         ; виберемо дільник для вх. імпульсів 1/1024
```

```
        out tccr0, tmp          ; значення в tccr0
        ldi tmp, 02             ; дозволимо переривання від таймера 0
        out TIMSK, tmp          ; запишемо в регістр масок
        out TCNT0, Time         ; передустановка таймера 0
        sei                     ; дозвіл всих переривань
```

**Робоча підпрограма**

Проводить підрахунок імпульсів, що подаються на вхід PinD,4; підпрограма спочатку чекає низького рівня сигналу на вході, потім чекає фронту імпульсу, повторно перевіряє наявність високого рівня, і якщо фронт не був помилковим спрацюванням, збільшує значення лічильника імпульсів на 1. Далі – знову чекає низького рівня вхідного сигналу і т.д. протягом часу вимірювання, котрий задається таймером 0. По перериванню від таймера 0 обробляється результат.

```
go0:
        sbis PinD,4         ; лог. 1 на вході PinD,4?
        rjmp go0            ; ні на go0
        sbis PinD,4         ; перевіримо ще раз
        rjmp go0
        inc Count           ;значить 1, імпульс почався
go1:sbic PinD,4             ;лог. 0 на вході PinD,4?
        rjmp go1            ;якщо 1 на go1
        rjmp go0            ;0 на go0
```

**Обробка переривання від таймера 0**

```
Timer0_Int:     ;використовуємо 2 переривання
        inc Flag        ;збільшити на 1
        cpi flag,2      ;вже два?
        brlo l1         ;ні – продовжимо
        mov Res,Count   ;так – зберегти результат
        clr Count       ;очистить лічильник импульсів
        clr flag        ;очистити лічильник переривань
```

```
        sts 0xa000,Res     ;результат підрахунків на дисплей
l1:     out TCNT0, Time    ;передустановка таймера
        reti               ;повернення з переривання
```

Замість таймера 0 можна використати таймер 1, цей таймер 16-бітний, і, відповідно, дозволяє задавати інтервали в межах $1\text{-}2^{15}$.

Налаштування таймера 1 аналогічно до таймера 0: встановити переддільники CS10-CS12 та записати значення в регістр керування TCCR1B, змінити вектор переривання на 06 – переривання від таймера 1. Крім того, слід обрати потрібний інтервал переривання – записати в рахункові регістри таймера передустановочні значення; обов'язково і в старший, і в молодший (TCNT1H та TCNT1L). В підпрограмі обробки переривань слід також записувати передустановочні значення.

Друга частина роботи

```
do:                    ; робочий цикл
do0:  sbis PinD,4      ; лог. 1 на вході PinD,4?
      rjmp do0         ; ні – чекаємо
      ldi tmp, 0x2     ; так – команда вкл. T1
      out TCCR1B,tmp   ; в регістр управління таймером 1
D1: sbic PinD,4        ; імпульс закінчено?
      rjmp D1          ; ні –чекаємо
      sbic PinD,4      ; перевіримо ще раз
      rjmp D1
      clr tmp          ; очистить регістр tmp
      out TCCR1B,tmp   ; зупинити  таймер 1
      in ResL, TCNT1L  ; читаємо результат
      in ResH, TCNT1H
      out TCNT1H, tmp      ; очистити регістри лічильника
      out TCNT1L, tmp
      sts 0xb000,ResL      ; результат – на дисплей
      sts 0xa000,ResH
```

rjmp do ; повторити

## *Лабораторна робота №6*

**Тема:** *формування аналогових сигналів керування виконавчими механізмами.*

**Мета:** *навчитись керувати аналоговими ВМ з використанням ЦАП та широтно імпульсної модуляції.*

## *Хід роботи*

1. Доповнити програму Л.Р. №5 можливістю змінювати напругу живлення двигуна вентилятора за допомогою ЦАП.

2. Записати програму в пам’ять ЕОМ та перевірити коректність вимірювання швидкості обертання валу двигуна за різних значень напруги живлення, в разі потреби удосконалити програму.

3. Дослідити залежність швидкості обертання валу двигуна від напруги живлення, для цього передбачити в програмі можливість одночасного відображення швидкості обертання валу та цифрового значення напруги керування (код, що записується в ЦАП). Для зручності досліду доцільно передбачити послідовне збільшення напруги (від мінімальної до максимальної), крок напруги та час вимірювань/перемикань обрати самостійно.

4. Доповнити програму модулем ШІМ у якості формувача аналогових сигналів керування. Рекомендується використання апаратного ШІМ, що входить до складу AVR контролера.

5. Повторити дослід п.3 з використанням ШІМ. Побудувати два графіки залежності швидкості обертання від напруги на одних координатних осях. Порівняти результати, зробити висновки.

**Підпрограма ініціалізації внутрішнього ШІМ**

**З використанням таймера 0**

```
Init_PWM:
    sbi DDRB,(0<<PB0) ;ножну на вихід
    ldi   tmp,(1<<CS00)|(1<<WGM00)|(1<<WGM01)|(1<<COM01)   ;без переддільника, FAST, прямий
    out TCCR0,tmp
    ldi tmp,128        ;значення для ШІМ,
    out OCR0,tmp       ;в регістр порівняння.
```

**З використанням таймера 1**

```
    ldi tmp,0x81       ;TCCR1A (bit1,0) pwm11 =0, pwm10=1
                       ;8bit, TOP=255 період 0,52мс дельник 1:1
                       ;TCCR1A bit7 COM1A1=1 COM1A0=0
                       ; COM1B1=0 COM1B0=0 тільки А
                       ;pwm11  =0,pwm10=1  10bit,TOP=1023  ;f=1/2043 T/C1 період 2мс
    ldi tmp,0xA1       ;TCCR1A (bit1,0) pwm11 =0, pwm10=1
                       ;8bit, TOP=255 період 0,52мс дільник 1:1
                       ;TCCR1A ст.тетрада COM1A1=1 COM1A0=0
                       ;COM1B1=1 COM1B0=0 ШІМ А і В одночасно
                       ;pwm11 =1, pwm10=1 10bit,TOP=1023 ;f=1/2043 T/C1 період 2мс
    out TCCR1A, tmp
    ldi tmp,01         ; дільник 1:1 мл. 3 біти CS02,CS01=0 CS0=1
```

```
;       ldi tmp,02          ; дільник 1:8 3 біти CS02=0 CS01=1 CS0=0
                            ;(період 4,5мс при 8біт ШІМ)
;       ldi tmp,03          ; дільник 1:64 3 біти CS02=0 CS01=1 CS0=1
                            ;(період 35 мс при 8біт ШІМ)
        out TCCR1b, tmp ;значення в регістр
        ldi tmp, 0          ;задаємо ст.байт ШІМ
        out OCR1BH,tmp ;в регістр порівняння  ШІМ В
        ldi tmp, 0x0f               ;молодший байт ШІМ
        out OCR1Bl,tmp  ;в регістр порівняння В
        ldi tmp, 0                  ;ст. байт ШІМ А
        out OCR1aH,tmp ;в регістр порівняння А
        ldi tmp, 0x0f               ;молодший байт
        out OCR1Al,tmp              ;в молодший регістр порівняння А
        ret
```

Після закінчення ініціалізації ШІМ починає працювати, однак для отримання імпульсів на виході мікро ЕОМ необхідно налаштувати відповідні біти портів для роботи «на вихід»:

```
        ldi tmp,0x04        ;біт порту ОС1В ШИМ В
        out DDRE,tmp        ;на вихід
        out PortE,tmp       ;
        ldi tmp,0x20        ;біт 5 порту ОС1А ШИМ А
        out DDRD,tmp        ;на вихід
```

```
out PortD,tmp     ;
```

Змінювати параметри ширини імпульсу можна в будь-якому місці програми шляхом вводу нових значень в регістри порівняння: OCR1BH : OCR1BL – старший та молодші байти для каналу B, та OCR1AH : OCR1AL – для каналу A відповідно. Після закінчення поточного періоду імпульсу нові параметри вступлять в силу. Для зміни ширини імпульсу під час реалізації ШІМ з використання таймера 0 значення слід записувати в регістр порівняння таймера 0 OCR0.

## *Лабораторна робота №7*

**Тема:** *використання інтерфейсу I2C.*

**Мета:** *освоїти програмні методи реалізації інтерфейсу $I^2C$ та взаємодію контролера з інтегральним датчиком температури та годинником реального часу.*

## *Хід роботи*

1. В середовищі AVR Studio скласти та відлагодити підпрограми, необхідні для програмної реалізації інтерфейсу $I^2C$:

   1.1) генератора синхроімпульсів та запису байту;

   1.2) генератора синхроімпульсів та читання байту;

   1.3) формування умови «старт» та «рестарт»;

   1.4) формування умови прийому та передачі імпульсу підтвердження;

   1.5) формування умови «стоп».

2. Скласти програму читання коду температури з цифрового датчика DS1621, апаратний адрес 0x90. Записати програму в пам’ять ЕОМ та перевірити її роботу на лабораторному макеті. Цифровий код

перетворити на °С та відобразити на екрані, в разі потреби удосконалити програму.

3. Скласти програму, котра зчитує поточну дату та час з годинника реального часу DS 1302 та відображає результат на дисплеї. Записати програму в пам’ять ЕОМ та перевірити роботу на макеті.

4. Доповнити програму п.3 можливістю корекції дати та часу; вводу нових значень дати та часу. Для наглядності роботи використати РК-дисплей, текстові пояснення та мигаючий курсор.

## Короткі теоретичні відомості

Мереж $I^2C$ передбачає використання пристроями ліній як на вхід, так і на вихід; всі пристрої, що об’єднані в мережу $I^2C$, мають вихід відкритий колектор. Вони можуть видавати лише сигнал лог. 0 (підкороти на нульову шину), а сигнал лог. 1 формується за допомогою додаткових резисторів, котрі «підтягують» мережу до шини живлення. Сигнал лог. 1 присутній лише тоді, коли виводи всіх абонентів мережі знаходяться в стані «на вхід», а видача хоча б одним абонентом лог. 0 встановить нуль на шині.

Будь-яка трансакція по інтерфейсу $I^2C$ починається з умови «Старт»: обидві лінії мережі знаходяться в стані лог. 1, це передумова старту, пристрій Майстер переводить шину SDA з 1 в 0 при одиниці на шині SLC. При програмній реалізації $I^2C$ необхідно, як мінімум, два рази перевірити наявність лог. 1 на шині SDA (передумова старту).

Всі передачі по шині виконуються 8-розрядними байтами, починаючи зі старшого біту. Кількість байт в посилці – необмежена, однак кожен байт повинен супроводжуватись бітом підтвердження (ASC). Сигнал свідчить про успішне отримання приймачем байту та про готовність приймати наступний.

Якщо приймач не готовий отримувати наступний байт даних, він не видає сигналу підтвердження, це може свідчити також про несправність приймача, його відсутність чи помилку в передачі.

Посилка складається з байту адреси відомого пристрою, молодший біт вказує на операцію читання, якщо він рівний 1, чи запису (молодший біт 0); потім пристрій Майстер передає адресу регістра в пристрої слейв, з котрим буде запис-читання, далі рестарт для читання чи байт для запису в вибраний регістр.

Приклад підпрограми підготовки до видачі команди «Старт» (підпрограма Restart), видачі команди «Старт»:

```
ReStart:    ; Перевірка можливості  старту
        SCL_Dwn   ;перевести шину SCL в лог.0
        nop
        SDA_Hi    ; Шину SDA в 1
        nop
        rcall wait5us      ; Цикл затримкики
        nop
        SCL_Up    ; Наростаючий фронт SCL->1
RS1:  sbis  PinB,SCL     ; Стан очікування відомого пристрою?
        rjmp  RS1
        rcall wait5us      ; Цикл затримки
Start:
        SDA_Lo    ; SDA = 0: умова старту
        rcall wait4us      ; Цикл затримки
```

Підпрограма запису байта в відомий пристрій:

```
WriteByte:
        sec   ; Прапорець переносу C = 1
        rol   dbyt   ; Зсув вліво C->LSB, MSB->C
        rjmp  bit1   ; Особлива обробка для розряду 1
WriteBit:
        lsl   dbyt   ; Якщо dbyt пустий, ...
```

```
bit1:  breq  GetAck       ; ... то передача завершена
       SCL_Dwn    ; Спадаючий фронт SCL
       brcc  WriteLow    ; Перехід, якщо прапорець C = 0
       nop   ; Щоб урівняти час виконання
       SDA_Hi      ; Встановити  SDA в 1
       rjmp  SCL_High
WriteLow:
       SDA_Lo      ; Встановити SDA в 0
       rjmp  SCL_High    ; Щоб зрівняти тривалість такту
SCL_High:
       rcall wait4us       ; Цикл затримки
       SCL_Up      ; Зростаючий фронт SCL
WB1:sbis  PinB,SCL       ; Стан очікування відомого пристрою?
       rjmp  WB1
       rcall wait5us       ; Цикл затримки
       rjmp  WriteBit
```

Підпрограма читання біта підтвердження:

```
GetAck:
       SCL_Dwn    ; Спадаючий фронт SCL
       SDA_Hi      ; SDA – високоомний стан
       rcall wait5us       ; Цикл затримки
       SCL_Up      ; Наростаючий фронт SCL
       GA1:sbis  PinB,SCL       ; Стан очікування відомого пристрою
       rjmp  GA1
       cbr   Flg,1<<Ack   ; Прапорець Ack = 0
       sbic  PinB,SDA     ; якщо SDA = 1,
       sbr   Flg,1<<Ack   ;   прапорець Ack = 1
       rcall wait4us       ; Цикл затримки
       ret
```

Підпрограма читання байту з відомого пристрою:

```
ReadByte:
        ldi   dbyt,1  ; Назначаємо в якості лічильника бітів
RB1:
        SCL_Dwn       ; спадаючий фронт SCL
        rcall wait5us
        SCL_Up        ; Наростаючий фронт SCL
RB2:
        sbis  PinB,SCL    ; Стан очікування відомого пристрою
        rjmp  RB2
        rcall wait5us
        clc               ; Прапорець C = 0
        sbic  PinB,SDA    ; Якщо SDA = 1
        sec               ;то    Carry = 1
        rol   dbyt    ; Зсуваємо прочитаний біт в байті
        brcc  RB1     ; Перехід, якщо читання не завершено

SetAck:     ; Видача біту підтвердження
        SCL_Dwn   ; Спадаючий фронт SCL
        sbrs  Flg,Ack       ;Пропускаємо наступну команду, якщо
        ;Ack=1
        rjmp  SA1   ; Перехід, якщо Ack = 0
        SDA_Hi      ; Встановить SDA в 1
        rjmp  SA2
SA1: SDA_Lo     ; Встановити SDA в 0
SA2: rcall wait5us
        SCL_Up      ; Наростаючий фронт SCL
SA3: sbis  PinB,SCL    ; Стан очікування відомого пристрою
        rjmp  SA3
        rcall wait5us
        ret

        Видача умови завершення трансакції:
Stopp:
```

```
SCL_Dwn    ; спадаючий фронт SCL
SDA_Lo     ; SDA в 0
rcall wait5us
SCL_Up     ; SCL в 1
rcall wait5us
SDA_Hi     ; Наростаючий фронт SCL
rcall wait5us
ret
```

## *Лабораторна робота №8*

**Тема:** *використання UART.*

**Мета:** *освоїти роботу апаратного UART для реалізації взаємодії контролер – PC, та контролер – контролер.*

1. В середовищі AVR Studio скласти та відлагодити програму, котра налаштовує UART на потрібний режим роботи для обміну даними з PC – ЕОМ. Програма повинна видавати байт даних та приймати байт даних, а результат роботи відображати на дисплеї. В якості дисплею доцільно використати РК-модуль

2. Завантажити програму в пам’ять контролера та переконатись в коректній роботі програми; зі сторони PC – ЕОМ використати програму TERMINAL.

3. Доповнити програму процедурою передачі кількох байт, наприклад, рядок символів та процедурою «прийом – модифікація – передача» байту. По закінченні прийому, переривання від UART, прийнятий байт збільшується на 1 та передається в зворотному напрямку.

4. Завантажити програму в пам’ять контролера та переконатись в коректній роботі програми.

5. Об’єднати два контролери між собою та переконатись в коректній роботі програми.

## Короткі теоретичні відомості

Ініціалізація UART.

```
InitUART:
    ldi tmp,0b11111000   ;біти RXCIE, TXCIE, UDRIE, RXEN,
                         ;TXEN=1, CHR9,RXB8,TXB8=0
    out $0a,tmp          ; в регістр керування UCR
    ldi tmp,12           ;встановити дільник для швидкості роботи
    out $09,tmp          ;UART 48-1200 24-2400 12-4800
    sei                  ;дозвіл переривань
```

Передача одного байту:

```
ldi tmp,0x31     ;один символ число 1 в ASCII
out $0c,tmp      ;в регістр передачі UART
```

Передача кількох байт, рядка символів. Байти в пам'яті, адреса байту в показнику Z, кількість символів (довжина рядка) в регістрі tmp. Переривання не використовуємо, перевіряємо прапорець «буфер передатчика пустий», якщо так – запишемо наступний байт для передачі. Всі байти передані – вихід з підпрограми.

```
T_Str:
    sbis $00b,5 ;USR UDRE (Uart Data Register Empty) регістр
                ;передавача пустий?;
    rjmp T_Str      ; ні –знову перевіряємо
    dec tmp         ;зменшити лічильник символів на 1
    breq Quit       ;0? Передали всі символи, на вихід
```

```
        lpm             ;читаємо наступний байт
        out $0c,r0          ;запишемо його в регістр передавача
        adiw Z,1        ;встановимо показчик на наступний символ
        rjmp T_Str      ;повторимо
Quit:   Ret             ;повернення з підпрограми.
```

Прийом одного байту з обробкою переривання від приймача:

```
U_Rx:   in t_buf, $0c           ;читати байт з приймача в регістр t_buf
        sts 0xa000, t_buf       ;видати на статичний дисплей
        reti                    ;повернення з переривання
```

### *Команди асемблера для AVR*

*Таблиця 8.1*

**Арифметичні та логічні команди**

| Мнемо-ніка | Операн-ди | Опис | Операція |
|---|---|---|---|
| **ADD** | Rd,Rr | Додавання без переносу | Rd = Rd + Rr |
| **ADC** | Rd,Rr | Додавання з переносом | Rd = Rd + Rr + C |
| **SUB** | Rd,Rr | Віднімання без переносу | Rd = Rd - Rr |
| **SUBI** | Rd,K8 | Віднімання константи | Rd = Rd - K8 |
| **SBC** | Rd,Rr | Віднімання з переносом | Rd = Rd - Rr - C |
| **SBCI** | Rd,K8 | Віднімання константи з переносом | Rd = Rd - K8 - C |
| **AND** | Rd,Rr | Логічне І | Rd = Rd · Rr |
| **ANDI** | Rd,K8 | Логічне І з константою | Rd = Rd · K8 |
| **OR** | Rd,Rr | Логічне АБО | Rd = Rd V Rr |
| **ORI** | Rd,K8 | Логічне АБО з константою | Rd = Rd V K8 |
| **EOR** | Rd,Rr | Логічне виключаюче АБО | Rd = Rd EOR Rr |
| **COM** | Rd | Побітна інверсія | Rd = $FF - Rd |
| **NEG** | Rd | Зміна знаку (Доп. код) | Rd = $00 - Rd |

| | | | |
|---|---|---|---|
| **SBR** | Rd,K8 | Встановити біт (біти) в регістрі | Rd = Rd V K8 |
| **CBR** | Rd,K8 | Очистить біт (біти) в регістрі | Rd = Rd · ($FF - K8) |
| **INC** | Rd | Інкрементувати значення регістра | Rd = Rd + 1 |

*Закінчення табл. 8.1*

| | | | |
|---|---|---|---|
| **DEC** | Rd | Декрементувати значення регістра | Rd = Rd -1 |
| **TST** | Rd | Перевірка на 0 чи від'ємне значення | Rd = Rd · Rd |
| **CLR** | Rd | Очистити регістр | Rd = 0 |
| **SER** | Rd | Встановити регістр | Rd = $FF |
| **ADIW** | Rdl,K6 | Додати константу до слова | Rdh:Rdl=Rdh:Rdl+ K6 |
| **SBIW** | Rdl,K6 | Відняти константу від слова | Rdh:Rdl = Rdh:Rdl - K 6 |
| **MUL** | Rd,Rr | Множення чисел без знаку | R1:R0 = Rd * Rr |
| **MULS** | Rd,Rr | Множення чисел зі знаком | R1:R0 = Rd * Rr |
| **MULSU** | Rd,Rr | Множення числа зі знаком з числом без знаку | R1:R0 = Rd * Rr |
| **FMUL** | Rd,Rr | Множ. дроб. чисел без знаку | R1:R0=(Rd*Rr)<<1 |
| **FMULS** | Rd,Rr | Множення дробових чисел зі знаком | R1:R0=(Rd*Rr)<<1 |
| **FMULSU** | Rd,Rr | Множення дробового числа зі знаком з числом без знаку | R1:R0=(Rd*Rr)<<1 |

*Таблиця 8.2*

**Команди розгалуження**

| Мне-моніка | Операн-ди | Опис | Операція |
|---|---|---|---|
| **RJMP** | k | Відносний перехід | PC = PC + k +1 |
| **IJMP** | Немає | Непрямий перехід на (Z) | PC = Z |
| **EIJMP** | Немає | Розширений непрямий перехід на (Z) | STACK = PC+1, PC(15:0) = Z, PC(21:16) = EIND |
| **JMP** | k | Перехід | PC = k |
| **RCALL** | k | Відносний виклик підпрограми | STACK = PC+1, PC = PC + k + 1 |
| **ICALL** | Немає | Непрямий виклик (Z) | STACK = PC+1, PC=Z |

| | | | |
|---|---|---|---|
| **EICALL** | Немає | Розширений непрямий виклик (Z) | STACK = PC+1, PC(15:0) = Z, PC(21:16) =EIND |
| **CALL** | k | Виклик підпрограми | STACK =PC+2, PC= k |
| **RET** | Немає | Повернення з підпрограми | PC = STACK |
| **RETI** | Немає | Повернення з підпрограми | PC = STACK |

*Продовження табл. 8.2*

| | | | |
|---|---|---|---|
| **CPSE** | Rd,Rr | Порівняти, пропустить якщо рівні | if (Rd ==Rr) PC = PC 2 or 3 |
| **CP** | Rd,Rr | Порівняти | Rd -Rr |
| **CPC** | Rd,Rr | Порівняти з переносом | Rd - Rr - C |
| **CPI** | Rd,K8 | Порівняти з константою | Rd - K |
| **SBRC** | Rr,b | Пропустить, якщо біт в регістрі очищений | if(Rr(b)==0) PC = PC + 2 or 3 |
| **SBRS** | Rr,b | Пропустить, якщо біт в регістрі встановлений | if(Rr(b)==1) PC = PC + 2 or 3 |
| **SBIC** | P,b | Пропустить, якщо біт в порту очищений | if(I/O(P,b)==0) PC = PC + 2 or 3 |
| **SBIS** | P,b | Пропустить, якщо біт в порту встановлений | if(I/O(P,b)==1) PC = PC + 2 or 3 |
| **BRBC** | s,k | Перейти, якщо прапорець в SREG очищений | if(SREG(s)==0) PC = PC + k + 1 |
| **BRBS** | s,k | Перейти, якщо прапорець в SREG встановлений | if(SREG(s)==1) PC = PC + k + 1 |
| **BREQ** | k | Перейти, якщо рівні | if(Z==1) PC =PC+k+ 1 |
| **BRNE** | k | Перейти, якщо не рівні | if(Z==0)PC=PC+k+1 |
| **BRCS** | k | Перейти, якщо перенос встановлений | if(C==1) PC = PC + k + 1 |
| **BRCC** | k | Перейти, якщо перенос очищено | if(C==0) PC = PC + k + 1 |
| **BRSH** | k | Перейти, якщо рівні чи більше | if(C==0) PC = PC + k + 1 |
| **BRLO** | k | Перейти, якщо менше | if(C==1) PC = PC + k + 1 |
| **BRMI** | k | Перейти, якщо мінус | if(N==1)PC=PC +k+ 1 |
| **BRPL** | k | Перейти якщо плюс | if(N==0)PC = PC+k+1 |
| **BRGE** | k | Перейти, якщо рівні чи більше (зі знаком) | if(S==0) PC =PC+k+1 |
| **BRLT** | k | Перейти, якщо менше (зі знаком) | if(S==1) PC =PC+k+ 1 |
| **BRHS** | k | Перейти, якщо прапорець | if(H==1) PC =PC+k+1 |

| | | | |
|---|---|---|---|
| | | внутрішнього переносу встановлений | |
| **BRHC** | k | Перейти, якщо прапорець внутрішнього переносу очищений | if(H==0) PC =PC+k+1 |
| **BRTS** | k | Перейти, якщо прапорець T встановлений | if(T==1) PC = PC + k + 1 |

*Закінчення табл. 8.2*

| | | | |
|---|---|---|---|
| **BRTC** | k | Перейти, якщо прапорець T очищений | if(T==0) PC = PC + k + 1 |
| **BRVS** | k | Перейти, якщо прапорець переповнення встановлений | if(V==1) PC = PC + k + 1 |
| **BRVC** | k | Перейти, якщо прапорець переповнення очищений | if(V==0) PC = PC + k + 1 |
| **BRIE** | k | Перейти, якщо переривання дозволені | if(I==1) PC = PC + k + 1 |
| **BRID** | k | Перейти, якщо перер. забор. | if(I==0) PC = PC +k+1 |

*Таблиця 8.3*

**Команди передачі даних**

| Мнемо-ніка | Операнди | Опис | Операція |
|---|---|---|---|
| **MOV** | Rd,Rr | Копіювати регістр | Rd = Rr |
| **MOVW** | Rd,Rr | Копіювати пару регістрів | Rd+1:Rd = Rr+1:Rr,r,d even |
| **LDI** | Rd,K8 | Загрузить константу | Rd = K |
| **LDS** | Rd,k | Пряма загрузка | Rd = (k) |
| **LD** | Rd,X | Непряма загрузка | Rd = (X) |
| **LD** | Rd,X+ | Непряма загрузка з пост-інкрементом | Rd = (X), X=X+1 |
| **LD** | Rd,-X | Непряма загрузка з пре-декрементом | X=X-1, Rd = (X) |
| **LD** | Rd,Y | Непряма загрузка | Rd = (Y) |
| **LD** | Rd,Y+ | Непряма загрузка з пост-інкрементом | Rd = (Y), Y=Y+1 |
| **LD** | Rd,-Y | Непряма загрузка з пре-декрементом | Y=Y-1, Rd = (Y) |
| **LDD** | Rd,Y+q | Непряма загрузка з заміщенням | Rd = (Y+q) |
| **LD** | Rd,Z | Непряма загрузка | Rd = (Z) |
| **LD** | Rd,Z+ | Непряма загрузка з пост- | Rd = (Z), Z=Z+1 |

| | | | |
|---|---|---|---|
| | | інкрементом | |
| **LD** | Rd,-Z | Непряма загрузка з пре-декрементом | Z=Z-1, Rd = (Z) |
| **LDD** | Rd,Z+q | Непрямая загрузка з замещенням | Rd = (Z+q) |
| **STS** | k,Rr | Пряме зберігання | (k) = Rr |
| **ST** | X,Rr | Непряме зберігання | (X) = Rr |

*Закінчення табл. 8.3*

| | | | |
|---|---|---|---|
| **ST** | X+,Rr | Непряме зберігання з пост-інкрементом | (X) = Rr, X=X+1 |
| **ST** | -X,Rr | Непряме зберігання з пре-декрементом | X=X-1, (X)=Rr |
| **ST** | Y,Rr | Непряме зберігання | (Y) = Rr |
| **ST** | Y+,Rr | Непряме зберігання з пост-інкрементом | (Y) = Rr, Y=Y+1 |
| **ST** | -Y,Rr | Непряме зберігання з пре-декрементом | Y=Y-1, (Y) = Rr |
| **ST** | Y+q,Rr | Непряме зберігання з заміщенням | (Y+q) = Rr |
| **ST** | Z,Rr | Непряме зберігання | (Z) = Rr |
| **ST** | Z+,Rr | Непряме зберігання з пост-інкрементом | (Z) = Rr, Z=Z+1 |
| **ST** | -Z,Rr | Непряме зберігання з пре-декрементом | Z=Z-1, (Z) = Rr |
| **ST** | Z+q,Rr | Непряме зберігання з заміщенням | (Z+q) = Rr |
| **LPM** | Немає | Загрузка з програмної пам’яті | R0 = (Z) |
| **LPM** | Rd,Z | Загрузка з прогр. пам’яті | Rd = (Z) |
| **LPM** | Rd,Z+ | Загрузка и програмної пам’яті з пост-інкрементом | Rd = (Z), Z=Z+1 |
| **ELPM** | Немає | Розширена загрузка з програмної пам’яті | R0 = (RAMPZ:Z) |
| **ELPM** | Rd,Z | Розширена загрузка з програмної пам’яті | Rd = (RAMPZ:Z) |
| **ELPM** | Rd,Z+ | Розширена загрузка з програмної пам’яті з пост-інкрементом | Rd = (RAMPZ:Z), Z = Z+1 |
| **SPM** | Немає | Збереження в програмній пам’яті | (Z) = R1:R0 |
| **ESPM** | Немає | Розширене збереження в програмній пам’яті | (RAMPZ:Z) = R1:R0 |

| | | | |
|---|---|---|---|
| **IN** | Rd,P | Читання порта | Rd = P |
| **OUT** | P,Rr | Запис в порт | P = Rr |
| **PUSH** | Rr | Занесення регістра в стек | STACK = Rr |
| **POP** | Rd | Добування регістра з стека | Rd = STACK |

*Таблиця 8.4*

**Команди роботи з бітами**

| Мнемо-ніка | Операн-ди | Опис | Операція |
|---|---|---|---|
| **LSL** | Rd | Логічний зсув вліво | Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7) |
| **LSR** | Rd | Логічний зсув вправо | Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0) |
| **ROL** | Rd | Циклічний зсув вліво через C | Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7) |
| **ROR** | Rd | Циклічний зсув вправо через C | Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0) |
| **ASR** | Rd | Арифметичний зсув вправо | Rd(n)=Rd(n+1), n=0,...,6 |
| **SWAP** | Rd | Перестановка тетрад | Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0) |
| **BSET** | s | Встановлення прапорця | SREG(s) = 1 |
| **BCLR** | s | Очистка прапорця | SREG(s) = 0 |
| **SBI** | P,b | Встановити біт в порту | I/O(P,b) = 1 |
| **CBI** | P,b | Очистить біт в порту | I/O(P,b) = 0 |
| **BST** | Rr,b | Зберегти біт з регістру в T | T = Rr(b) |
| **BLD** | Rd,b | Загрузить біт з T в регістр | Rd(b) = T |
| **SEC** | Немає | Встановити прапорець переносу | C =1 |
| **CLC** | Немає | Очистить прапорець переносу | C = 0 |
| **SEN** | Немає | Встановити прапорець від'ємного числа | N = 1 |
| **CLN** | Немає | Очистить прапорець від'ємного числа | N = 0 |
| **SEZ** | Немає | Встановити прапорець 0 | Z = 1 |
| **CLZ** | Немає | Очистить прапорець нуля | Z = 0 |
| **SEI** | Немає | Встановити прапорець | I = 1 |

| | | | |
|---|---|---|---|
| | | переривання | |
| **CLI** | Немає | Очистить прапорець переривання | I = 0 |
| **SES** | Немає | Встановити прапорець числа зі знаком | S = 1 |

*Закінчення табл. 8.4*

| | | | |
|---|---|---|---|
| **CLN** | Немає | Очистить прапорець числа зі знаком | S = 0 |
| **SEV** | Немає | Встановити прапорець переповнення | V = 1 |
| **CLV** | Немає | Очистить прапорець переповнення | V = 0 |
| **SET** | Немає | Встановити прапорець T | T = 1 |
| **CLT** | Немає | Очистить прапорець T | T = 0 |
| **SEH** | Немає | Встановити прапорець внутрішнього переносу | H = 1 |
| **CLH** | Немає | Очистить прапорець внутрішнього переносу | H = 0 |
| **NOP** | Немає | Немає операції | Нет |

## СПИСОК ЛІТЕРАТУРИ


1. *Вольганг Трамперт.* AVR-RISC микроконтроллеры : підручник / В. Трамперт. – Київ : МК-Пресс, 2006. – 464 с.
2. *Баранов В.Н.* Применение микроконтроллеров AVR. Схемы, алгоритмы, программы : навч. посібник / В.Н. Баранов. – Київ : Додека, 2004. – 256 с.
3. *Вольганг Трамперт.* Измерение, управление и регулирование с помощью AVR мироконтроллеров / В. Трамперт. – Київ : МК-Пресс, 2006. – 208 с.
4. *Цирульник С. М.* Програмування мікроконтролерів AVR : навчальний посібник / С. М. Цирульник, О. Д. Азаров, Л. В. Крупельницький, Т. І. Трояновська. – Вінниця : ВНТУ, 2018. – 111 с.

**Навчально-методичне видання**

# ОСНОВИ КОМП'ЮТЕРНО-ІНТЕГРОВАНОГО УПРАВЛІННЯ

Методичні вказівки
до виконання лабораторних і практичних робіт
для здобувачів першого (бакалаврського) рівня вищої освіти
за спеціальністю 174 «Автоматизація, комп'ютерно-інтегровані технології та робототехніка»

Укладачі **Самойленко** Микола Іванович

Випусковий редактор *Л.С. Тавлуй*
Комп'ютерне верстання *Т.І. Кукарєвої*

Підписано до друку 24.06.2025. Формат 60 × 84 1/ 16
Ум. друк. арк. 2,56. Обл.-вид. арк. 2,75.
Електронний документ. Вид. № 54/III–25.

Видавець і виготовлювач
Київський національний університет будівництва і архітектури

Проспект Повітряних Сил, 31, Київ, Україна, 03037

Свідоцтво про внесення до Державного реєстру суб'єктів
видавничої справи ДК № 808 від 13.02.2002 р.