

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний технічний університет
імені Івана Пулюя

**Кафедра автоматизації технологічних
процесів та виробництв**

Методичні вказівки
до лабораторної роботи № 10
Керування кроковим двигуном з використанням програмного
симулятора AVR Simulator IDE
з курсу
”Мікропроцесорні та програмні засоби автоматизації”

Тернопіль 2020

Методичні вказівки до лабораторної роботи №10 «Керування кроковим двигуном з використанням програмного симулятора AVR Simulator IDE» з курсу «Мікропроцесорні та програмні засоби автоматизації».

Методичні вказівки розглянуті і схвалені кафедрою «Автоматизація технологічних процесів та виробництв», протокол № 8 від 18.02.2020 р.

Відповідальні за випуск

доцент, к.т.н. Медвідь В.Р.,
асистент Пісьціо В.П.

Лабораторна робота №10

Керування кроковим двигуном з використанням програмного симулятора AVR Simulator IDE

1. Послідовність роботи з програмним симулятором AVR Simulator IDE

Основне вікно програми AVR Simulator IDE має вигляд, показаний на (рис. 1).

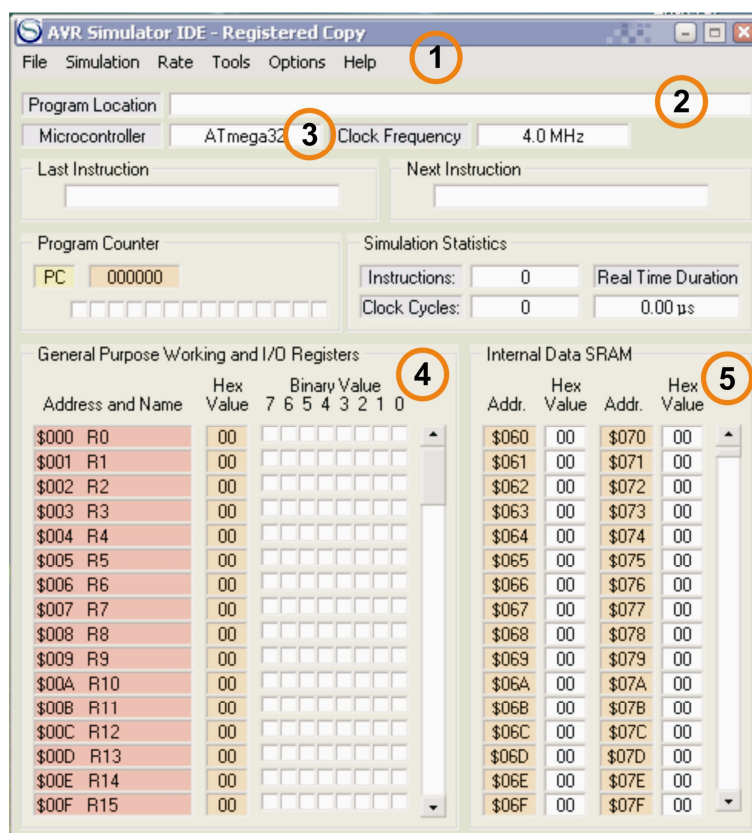


Рис. 1. Основне вікно програми AVR Simulator IDE

У верхній частині знаходяться меню, через які можна отримати доступ до основних і додаткових модулів програми (поз. 1)(рис. 1).

В рядку Program Location вказано шлях до обраної програми і її ім'я (поз. 2).

В рядку Microcontrollers, відображається тип обраного мікроконтролера (поз. 3).

У нижній частині вікна є дві панелі (поз.4 і поз.5), де відображається стан внутрішніх регістрів мікроконтролерів AVR (регістрів загального користування та регістрів вводу/виводу), та SRAM внутрішніх даних відповідно.

Також у основному вікні відображені лічильник програм, мнемоніка останньої виконуваної інструкції, мнемоніка наступної інструкції, що буде виконуватися, цикли та інструкції лічильника і тривалість імітації в режимі реального часу.

2. Послідовність роботи з програмним симулятором наступний:

- запуск програми AVR Simulator IDE;
- вибір типу мікроконтролера, для якого написана програма;
- вибір частоти кварцового генератора (впливає тільки на відображувані програмою дані про час виконання програми або команди, але не на швидкість роботи програми, що налагоджуються в AVR Simulator IDE);
- завантаження програми у вигляді HEX-файлу або запуск вбудованого компілятора мови асемблера і написання в ньому потрібної програми;
- вибір потрібних модулів віртуальних пристроїв;
- вибір швидкості і режиму роботи програми симулятора;
- запуск процесу симуляції роботи програми на обраному МК.

Якщо потрібно скористатися для роботи з симулятором власною програмою або внести зміни у вже розроблену, необхідно створити або завантажити для цього файл асемблера, з якого після компіляції буде створений необхідний для роботи з симулятором hex-файл.

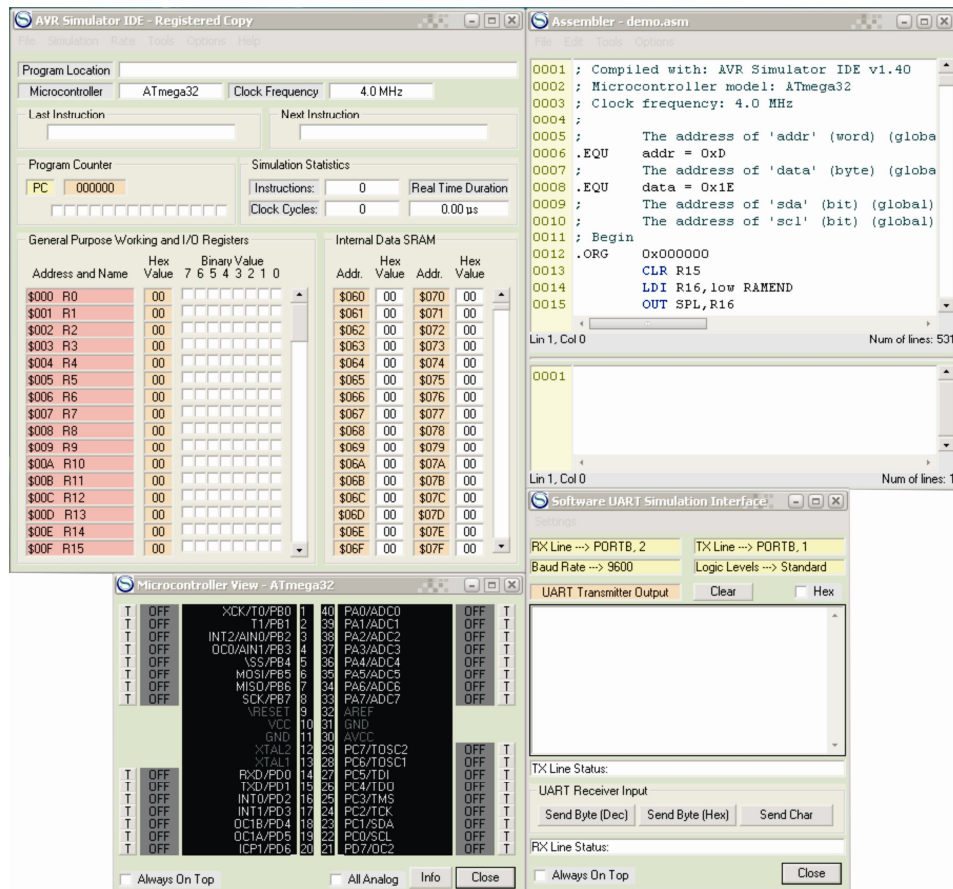


Рис. 2 Вікно симулятора з полем компілятора Assembler, апаратними виводами контролера, полем послідовного інтерфейсу

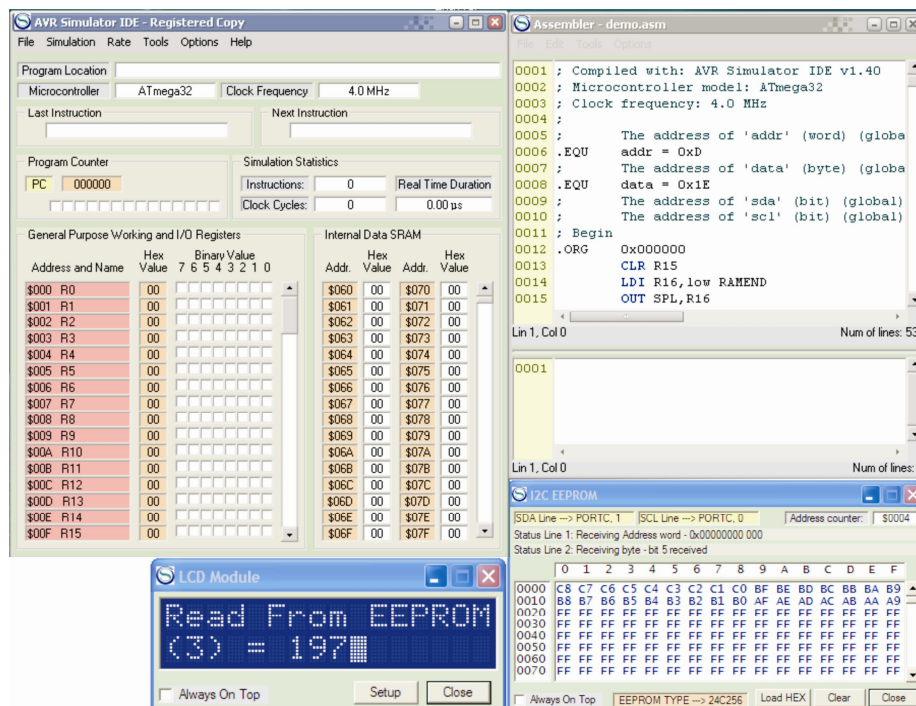


Рис. 3 Вигляд симулятора з полем компілятора Assembler, LED- модулем, I2C EEPROM

Для цього:

1. Натиснути Options | Assembler. Відкриється вікно компілятора Assembler – UNTITLED (рис. 2);

2. У вікні Assembler натиснути опцію File. Розкриється закладка, з якої для створення нового файлу потрібно натиснути New, а для завантаження вже створеного – OPEN.

3. Після вибору і завантаження файлу (з розширенням .asm), його текст з'явиться у вікні Assembler .

4. Для компіляції створеного або завантаженого і потім зміненого файлу, натисніть Tools і у вікні, що розкриється – Assemble. В нижній половині вікна Assembler з'явиться лістинг відкомпільованого файлу і, одночасно, при відсутності помилок, буде створений одноіменний hex-файл.

3. Для виконання лабораторної роботи по вивченню основних операцій МК:

1. Вивчити програмну модель AVR Simulator IDE.

2. Вивчити команди арифметичних, логічних операцій, операцій з портами та пам'яттю AVR мікроконтролера.

3. Дослідити роботу програм за вказівкою викладача та вміст регістрів контролера, які використовуються при виконанні цієї програми.

4. Записати для вибраних команд асемблера коментар щодо їх призначення.

4. Послідовність роботи з симулятором при виконанні програм

Виконати програму відповідно до вказаного викладачем завдання в AVR Simulator ID, для чого необхідно:

1. Запустити AVR Simulator IDE;

2. Натиснути Options | Select Microcontroller;

3. Вибрати ATmega32 і натиснути кнопку Select;

4. Натиснути Tools і у вікні, що розкриється, вибрати «Assembler». Відкриється вікно компілятора «Assembler – UNTITLED» (рис. 2);

5. Набрати текст заданої програми у вікні «Assembler»;

6. Натиснути Tools і у вікні, що розкриється – Assemble. В нижній половині вікна Assembler з'явиться лістинг відкомпільованого файлу;

7. Одночасно, при відсутності помилок, буде створений файл з розширенням «hex», для якого можна вибрати ім'я та шлях для запису. Записати його на «Робочий стіл» комп'ютера;

8. Вибрати File | Load Program і завантажити створений файл hex-файл;

9. В основному вікні симулятора натиснути Rate | Step By Step, а далі вибрати опцію Simulation і натиснути Start. Симулятор готовий до виконання програми в кроковому режимі;

10. Для виконання наступної команди програми потрібно натиснути на закладку STEP, яка з'явиться справа від закладки HELP вгорі основного вікна симулятора після вибору крокового режиму його роботи;

11. Для виконання програми в автоматичному режимі потрібно вибрати Rate | Extremely Fast simulation rate;

12. Щоб зупинити виконання програми, потрібно натиснути Simulation | Stop.

Вміст регістрів мікроконтролера, які використовуються при виконанні команд програми, знайти в області регістрів Address and Name, яка розташована в лівій нижній частині основного вікна симулятора (виділені рожевим кольором). Всі регістри восьмирозрядні.

В процесі виконання програми по зміні кольору комірок видно, вміст яких регістрів змінюється. Забарвлення комірки відповідного розряду регістру помаранчевим кольором означає наявність “1”, білим - “0”.

5. Завдання на лабораторну роботу

Завдання: вивід даних через порти AVR-контролера для керування кроковим двигуном.

1. Вивчити програмну модель AVR Simulator IDE.
2. Вивчити команди арифметичних операцій AVR – мікроконтролера.
3. Дослідити роботу програми з Прикладу 1 та вміст регістрів, які використовуються при виконанні цієї програми.
4. Записати для вибраних команд асемблера коментар щодо їх призначення (див. Приклад 2).

6. Режим керування біполярним кроковим двигуном

6.1. Типи крокових двигунів

Кроковий двигун (КД) являє собою безколекторний двигун постійного струму з фіксованими положеннями валу.

КД призначено для точного позиціювання валу без застосування систем зворотного зв'язку. Обмотки КД є частиною статора. На роторі розташовано постійний магніт або, у випадках зі змінним магнітним опором, зубчастий блок з магнітом'якого матеріалу.

Кроковий двигун має не менш двох положень стійкої рівноваги ротора в межах одного оберту. Напруга живлення обмоток керування кроковим двигуном - це послідовність однополярних або двополярних прямокутних імпульсів, що надходять від електронного комутатора або контролера. Результуючий кут відповідає кількості перемикачів комутатора, а частота обертання двигуна – частоті перемикачів електронного комутатора.

На рис. 4 зображено положення ротора крокового двигуна залежно від комутації обмоток. Послідовно комутуючи *струм в обмотках відповідно до діаграм*, наведених на рис.5, можна змусити обертатися вектор магнітного поля, а за ним і ротор, у прямій або зворотній послідовності. Від'ємне значення струму через обмотку на діаграмі відповідає логічному «1-0» та зворотньому «0-1» значенням напруги, прикладеної до обмотки. При такому керуванні двигун має 4 стійких стани на одному оберті ротора.

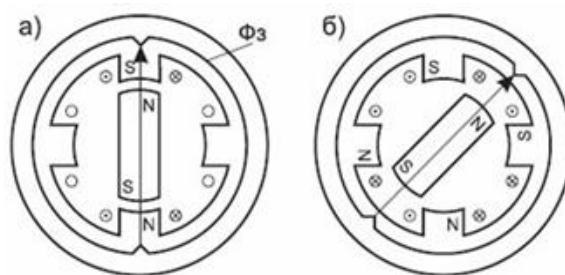


Рис. 4. Положення ротора при кроковому (а) і півкроковому керуванні (б)

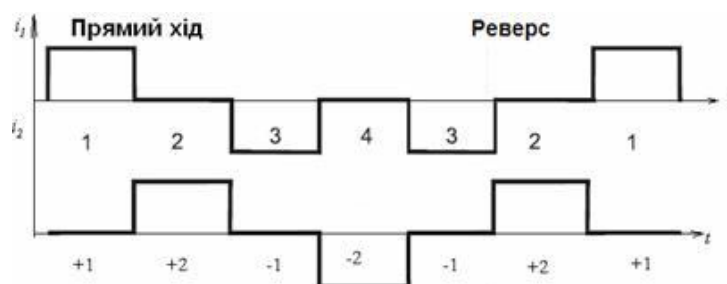


Рис. 5. Прямий і зворотний хід у кроковому двигуні

6.2. Біполярні і уніполярні крокові двигуни

В залежності від того, якою є форма обмоток крокового двигуна, двигуни діляться на **уніполярні і біполярні**.

У **біполярного** двигуна по 1 обмотці в кожній фазі, тобто всього дві обмотки і відповідно 4 виводи (рис. 6,а). Для забезпечення обертання валу на ці обмотки має подаватися напруга із змінною полярністю. Тому для біполярного двигуна необхідний півмостовий або мостовий драйвер, забезпечений двополярним живленням.

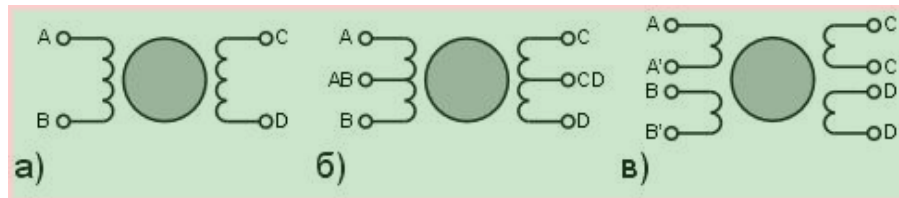


Рис. 6. Біполярні та уніполярні крокові двигуни

Уніполярний двигун також, як і біполярний, для кожної фази має по одній обмотці, але кожна обмотка містить відвід від середини. У зв'язку з цим, шляхом перемикання половинок обмоток крокового двигуна з'являється можливість міняти напрям магнітного поля (6,б).

Уніполярні двигуни можуть забезпечуватися чотирма обмотками, кожна з яких містить власні виводи - тобто їх всього вісім (рис. 6,в).

Уніполярний двигун, що має дві обмотки з відводами по середині, можна використовувати як біполярний. У цьому випадку проводи, що йдуть від середини обмоток, не використовуються.

Симулятор **AVR Simulator IDE** імітує роботу **уніполярного крокового двигуна**.

Він містить дві обмотки з середніми виводами (рис. 7), які заземляються (у вікні симулятора крокового двигуна заземлення середніх виводів не показане).

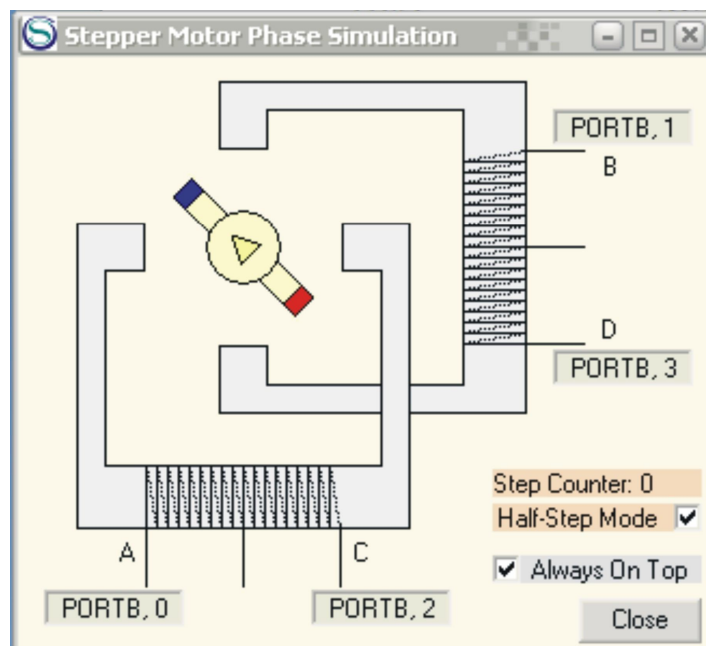


Рис. 7 Вікно «Stepper Motor Phase Simulation»

6.3. Основні способи управління кроковим двигуном

- Повнокроковий режим без перекриття фаз
- Повнокроковий режим з перекриттям фаз
- Півкроковий режим

Повнокроковий режим без перекриття фаз

У цьому режимі (рис. 8) в один момент часу живиться тільки одна фаза двигуна. Полюси ротора займають положення навпроти обмотки, на яку подається живлення, залежно від напрямку протікання струму в ній. Недоліком даного способу є менший момент двигуна.

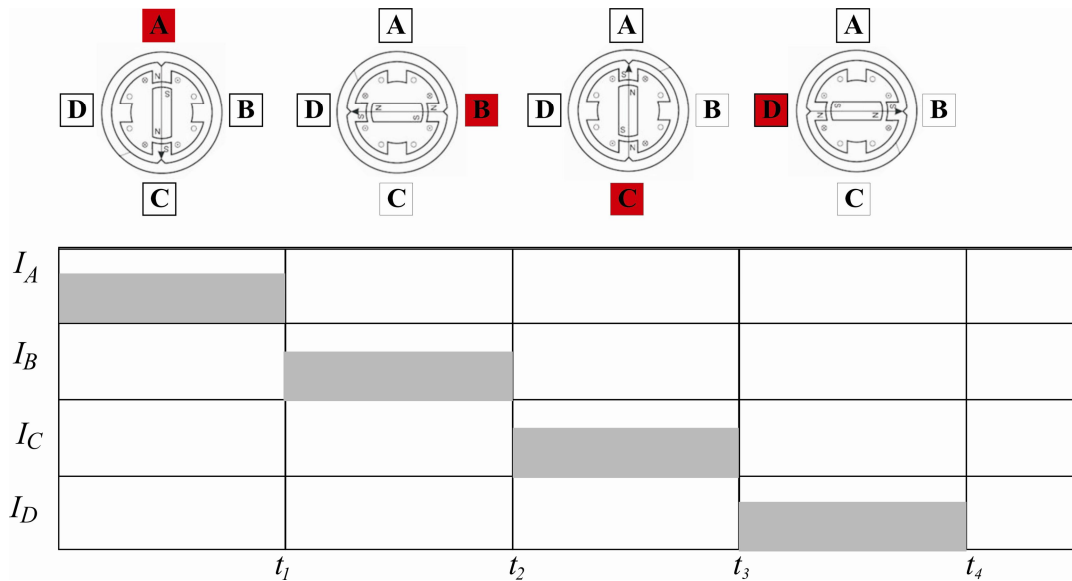


Рис. 8. Діаграми роботи в повнокроковому режимі без перекриття фаз для уніполярного двигуна

Повнокроковий режим з перекриттям фаз

Характерною рисою даного режиму є те, що одночасно подається живлення на дві суміжні фази, і ротор зупиняється не навпроти полюсів, а в проміжному положенні між ними.

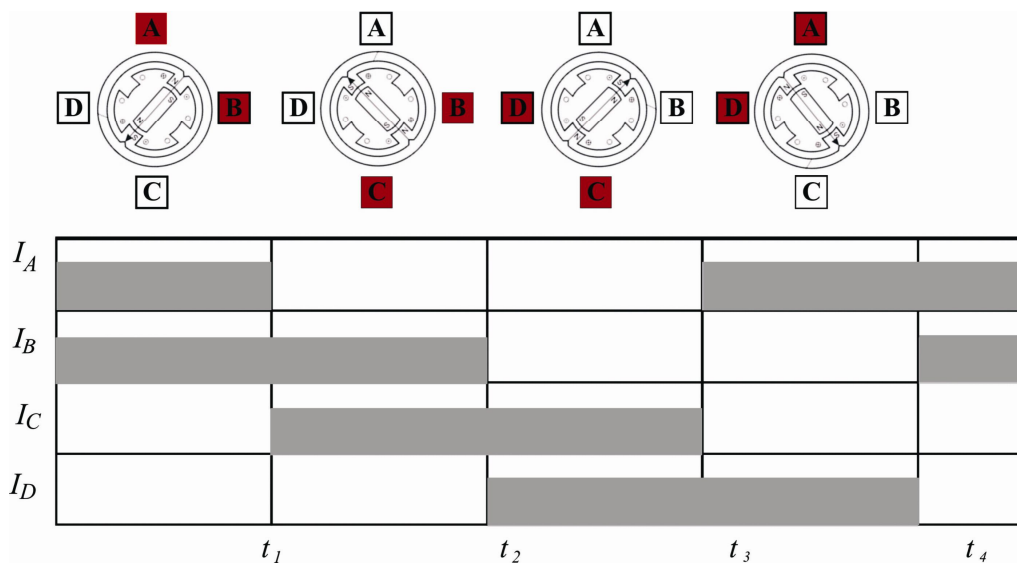


Рис. 9. Діаграми роботи в повнокроковому режимі з перекриттям фаз для уніполярного двигуна

Даний спосіб комутації фаз (рис. 9) забезпечує в $2^{\frac{1}{2}}$ разів більший момент. При зупинці двигуна важливо не знеструмлювати його обмотки, щоб двигун забезпечував повний момент, бо може статися зсув ротора на половину кроку і відповідно втрата положення.

Півкроковий режим

Є комбінацією двох вищеназваних, тобто ротор зупиняється як навпроти полюсів, так і в проміжному положенні між ними. З одного боку, це дозволяє зменшити крок у два рази, з іншого - будуть коливання моменту, тому що коли на дві фази подається живлення, то момент буде в $2^{\frac{1}{2}}$ разів більшим.

Щоб уникнути цього явища, в момент, коли включені дві фази, необхідно знижувати на них струм теж у $2^{\frac{1}{2}}$ рази, але це призведе до загального зменшення моменту.

Проблемою півкрокового режиму є перехід в стан з однією включеною фазою. В цьому випадку потрібно якомога швидше звести в фазі, яка вимикається, струм до нуля.

Коли використовується мостова схема, це здійснюється вимиканням всіх ключів, що призводить до того, що великий розрядний струм протікає через діоди і джерело живлення.

Якщо ж залишити один ключ замкненим, то коло розряду буде включати в себе діод і ключ, тому струм буде спадати повільніше.

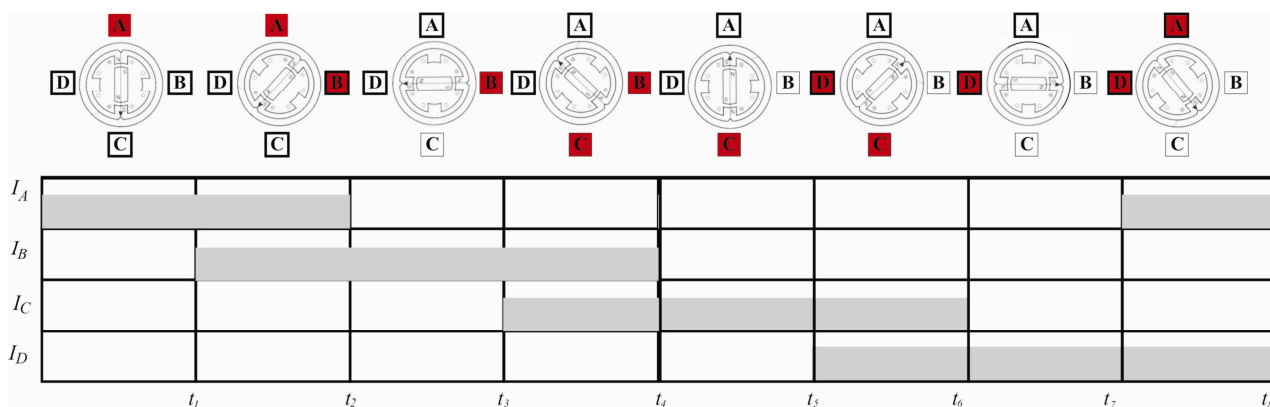


Рис. 10. Діаграми роботи в півкроковому режимі для уніполярного двигуна

Діаграму, зображену на рис. 8 у вигляді **покрокової залежності струмів через обмотки** в повнокроковому режимі без перекриття фаз, зобразимо у вигляді логічних рівнів напруг на виводах порту В на кожному з кроків роботи двигуна протягом двох обертів ротора (табл. 1).

Таблиця 1

	PORTB.3	PORTB.2	PORTB.1	PORTB.0
Крок 1	«0»	«0»	«0»	«1»
Крок 2	«0»	«0»	«1»	«0»
Крок 3	«0»	«1»	«0»	«0»
Крок 4	«1»	«0»	«0»	«0»
Крок 5	«0»	«0»	«0»	«1»
Крок 6	«0»	«0»	«1»	«0»
Крок 7	«0»	«1»	«0»	«0»
Крок 8	«1»	«0»	«0»	«0»
.....				

Робота крокового двигуна **в повнокроковому режимі без перекриття фаз** відповідно до логічних рівнів напруг з табл. 1, реалізується програмою в **Прикладі 1**.

Приклад 1

Програма забезпечує керування кроковим двигуном, який під'єднаний до чотирьох молодших розрядів порту В (рис. 7).

Текст програми з робочого файлу має наступний вигляд:

; основна програма

start:

ldi R16, 0x00

ldi R17, 0xFF

; Порт В на вихід з низьким початковим рівнем

out DDRB, R17

out PORTB, R16

M1: ldi R20, 0x01

; завантаження «0001» для виводу в молодший розряд

; порту В

out PORTB, R20

; вивід вмісту R20 в порт В

M2: lsl r20

; зсув вмісту R20 вліво

out PORTB, R20

; вивід вмісту R20 в порт В

brhc M1

; перехід на мітку, якщо прапорець додаткового

; перенесення рівний «1» (з 4-го в 5-ий розряд)

jmp M2

por

7. Завдання для виконання лабораторної роботи

Завдання 1

1. Виконати програму (Приклад1) в режимі «Normal». Переконалися в реалізації обертання ротора крокового двигуна в процесі виконання програми.

2. Виконати програму в повнокроковому режимі виконання програми. Вміст тих регістрів, значення яких змінюється в процесі виконання команд програми, записати в шістнадцятковому коді в табл. 2.

Таблиця 2

Регістр	PC	R16	R17	R20	R21	DDRB	PORTB	SREG	DDRA	PORT A
Команда 1										
Команда 2										
.....										
Команда n										

Завдання 2

1. Скласти програму, яка реалізує **реверсне обертання крокового двигуна в повнокроковому режимі роботи двигуна** (режим ілюструється діаграмою на рис. 10).

2. Виконати програму в режимі «Normal». Переконалися в реалізації реверсного обертання ротора крокового двигуна в процесі виконання програми.

3. Виконати програму в кроковому режимі виконання програми.

4. Вміст тих регістрів, значення яких змінюється в процесі виконання команд програми, записати в шістнадцятковому коді в табл. 2.

5. З виконуваної програми вибрати десять команд і за таблицею команд асемблера для AVR-контролера (Додаток 1) записати коментар щодо призначення цих команд (див. Приклад 2, де наведено такий запис для однієї команди).

Завдання 3

1. Скласти програму, яка реалізує обертання **крокового двигуна в напівкроковому режимі** його роботи, що ілюструється діаграмою на рис. 10.

2. Скласти таблицю за прикладом табл. 1, яка ілюструє зміну рівнів напруги на виводах порту В в процесі керування двигуном, користуючись діаграмою зміни струмів через обмотки двигуна.

3. Виконати програму в режимі «Normal». Переконалися в реалізації обертання ротора крокового двигуна з кроком в 45^0 в процесі виконання програми (двигун має 8 стійких станів на одному оберті ротора).

4. Виконати програму в кроковому режимі виконання програми.

5. Вміст тих регістрів, значення яких змінюється в процесі виконання команд програми, записати в шістнадцятковому коді в табл. 2.

6. З виконуваної програми вибрати десять команд і за таблицею команд асемблера для AVR-мікроконтролера (в Додатку 1) записати коментар щодо призначення цих команд (див. Приклад 2, де наведено такий запис для однієї команди).

8. Контрольні запитання

1. Використання AVR-мікроконтролерів.

2. Програмування портів мікроконтролера.

3. Організація циклів в роботі мікроконтролера.

4. Формат та використання регістрів загального призначення.

5. Призначення та позначення основних елементів програмної моделі мікроконтролера.

9. Література

1. Програмування мікроконтролерів систем автоматики: конспект лекцій для студентів базового напрямку 050201 “Системна інженерія” / Укл.: А.Г. Павельчак, В.В. Самотий, Ю.В. Яцук – Львів: Львівська політехніка. – 2012. – 143 с.

2. Евстифеев А. В. Микроконтроллеры AVR семейств Тiny и Mega фирмы ATMEL, – [5-е изд., стер.] / Евстифеев А. В. – М.: Издательский дом «Додэка-XXI», 2008. 560 с.

Система команд AVR мікроконтролерів включає команди арифметичних і логічних операцій, команди передачі даних, команди, що керують послідовністю виконання програми і команди операцій з бітами.

Для зручності написання й аналізу програм всім операціям із системи команд крім двійкового коду зіставлені мнемокоди Ассемблера (символічні позначення операцій), що використовуються при створенні вихідного тексту програми.

Спеціальні програми-транслятори переводять потім символічні позначення в двійкові коди.

Спеціальна директива ассемблера **.device** забезпечує контроль відповідності команд, використовуваних у тексті програми, типу зазначеного процесора.

Під час виконання арифметичних, логічних чи операцій роботи з бітами ALU формує ознаки результату операції, тобто встановлює чи скидає біти в регістрі стану **SREG** (Status Register).

Регістр статусу - SREG - розміщений у просторі I/O за адресою \$3F (\$5F).

Таблиця 1 – Регістр статусу - SREG

Біти	7	6	5	4	3	2	1	0	
\$3F (\$5F)	I	T	H	S	V	N	Z	C	REG
Читання/Запис	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Початковий стан	0	0	0	0	0	0	0	0	

Bit 7 - I: Global Interrupt Enable - Дозвіл глобального переривання. Біт дозволу глобального переривання для дозволу переривання повинний бути встановлений у стан 1. Керування дозволом конкретного переривання виконується регістрами маски переривання GIMSK і TIMSK. Якщо біт глобального переривання очищений (у стані 0), то жодне з дозволів конкретних переривань, встановлених у регістрах GIMSK і TIMSK, не діє.

Біт I апаратно очищається після переривання і встановлюється для наступного дозволу глобального переривання командою RETI.

Bit 6 - T: Bit Copy Storage - Біт збереження копії. Команди копіювання біта BLD (Bit Load) і BST (Bit STore) використовують біт T, як біт джерело і біт призначення при операціях з бітами. Командою BST біт регістра копіюється до біту T, командою BLD біт T копіюється до регістру.

Bit 5 - H: Half Carry Flag - Прапор напівпереносу. Прапор напівпереносу вказує на напівперенос у ряді арифметичних операцій.

Bit 4 - S: Sign Bit, S = N V - Біт знаку. Біт S завжди знаходиться в стані, обумовленому логічною функцією АБО (OR) між прапором негативного значення N і доповненням до двох прапора переповнення V.

Bit 3 - V: Two's Complement Overflow Flag. Доповнення до двох прапора переповнення. Доповнення до двох прапора V підтримує арифметику доповнення до двох.

Bit 2 - N: Negative Flag – Прапор негативного значення. Прапор негативного значення N вказує на негативний результат ряду арифметичних і логічних операцій.

Bit 1 - Z: Zero Flag – Прапор нульового значення. Прапор нульового значення Z вказує на нульовий результат ряду арифметичних і логічних операцій.

Bit 0 - C: Carry Flag – Прапор переносу. Ознаки результату операції можуть бути використані в програмі для виконання подальших арифметично-логічних операцій чи команд умовних переходів.

Арифметичні і логічні конструкції

Мнемоніка	Операнди	Опис	Операція	Прапори	Цикли
ADD	Rd,Rr	Підсумовування без переносу	$Rd = Rd + Rr$	Z,C,N,V,H,S	1
ADC	Rd,Rr	Підсумовування з переносом	$Rd = Rd + Rr + C$	Z,C,N,V,H,S	1
SUB	Rd,Rr	Вирахування без переносу	$Rd = Rd - Rr$	Z,C,N,V,H,S	1
SUBI	Rd,K8	Вирахування константи	$Rd = Rd - K8$	Z,C,N,V,H,S	1
SBC	Rd,Rr	Вирахування з переносом	$Rd = Rd - Rr - C$	Z,C,N,V,H,S	1
SBCI	Rd,K8	Вирахування константи з переносом	$Rd = Rd - K8 - C$	Z,C,N,V,H,S	1
AND	Rd,Rr	Логічне І	$Rd = Rd \cdot Rr$	Z,N,V,S	1
ANDI	Rd,K8	Логічне І з константою	$Rd = Rd \cdot K8$	Z,N,V,S	1
OR	Rd,Rr	Логічне АБО	$Rd = Rd \vee Rr$	Z,N,V,S	1
ORI	Rd,K8	Логічне АБО з константою	$Rd = Rd \vee K8$	Z,N,V,S	1
EOR	Rd,Rr	Логічне що виключає АБО	$Rd = Rd \oplus Rr$	Z,N,V,S	1
COM	Rd	Побітна Іверсія	$Rd = \$FF - Rd$	Z,C,N,V,S	1
NEG	Rd	Зміна знака (Доп. код)	$Rd = \$00 - Rd$	Z,C,N,V,H,S	1
SBR	Rd,K8	Установити біт (біти) у регістрі	$Rd = Rd \vee K8$	Z,C,N,V,S	1
CBR	Rd,K8	Скинути біт (біти) у регістрі	$Rd = Rd \cdot (\$FF - K8)$	Z,C,N,V,S	1
INC	Rd	Інкрементувати значення регістра	$Rd = Rd + 1$	Z,N,V,S	1
DEC	Rd	Декрементувати значення регістра	$Rd = Rd - 1$	Z,N,V,S	1
TST	Rd	Перевірка на нуль або заперечність	$Rd = Rd \cdot Rd$	Z,C,N,V,S	1
CLR	Rd	Очистити регістр	$Rd = 0$	Z,C,N,V,S	1
SER	Rd	Установити регістр	$Rd = \$FF$	None	1
ADIW	Rd1,K6	Скласти константу і слово	$Rdh:Rdl = Rdh:Rdl + K6$	Z,C,N,V,S	2
SBIW	Rd1,K6	Вичитати константу зі слова	$Rdh:Rdl = Rdh:Rdl - K6$	Z,C,N,V,S	2

Інструкції розгалуження

Мнемоніка	Операнди	Опис	Операція	Прапори	Цикли
RJMP	k	Відносний перехід	$PC = PC + k + 1$	None	2
IJMP	Немає	Непрямий перехід на (Z)	$PC = Z$	None	2
EIJMP	Немає	Розширений непрямий перехід на (Z)	$STACK = PC + 1, PC(15:0) = Z, PC(21:16) = EIND$	None	2
JMP	k	Перехід	$PC = k$	None	3
RCALL	k	Відносний виклик підпрограми	$STACK = PC + 1, PC = PC + k + 1$	None	3/4*
ICALL	Немає	Непрямий виклик (Z)	$STACK = PC + 1, PC = Z$	None	3/4*
EICALL	Немає	Розширений непрямий виклик (Z)	$STACK = PC + 1, PC(15:0) = Z, PC(21:16) = EIND$	None	4*
RET	Немає	Повернення з підпрограми	$PC = STACK$	None	4/5*
RETI	Немає	Повернення з переривання	$PC = STACK$	I	4/5*
CPSE	Rd,Rr	Порівняти, пропустити якщо рівні	$\text{if } (Rd == Rr) PC = PC + 2 \text{ or } 3$	None	1/2/3
CP	Rd,Rr	Порівняти	$Rd - Rr$	Z,C,N,V,H,S	1
CPC	Rd,Rr	Порівняти з переносом	$Rd - Rr - C$	Z,C,N,V,H,S	1
CPI	Rd,K8	Порівняти з константою	$Rd - K$	Z,C,N,V,H,S	1
SBRC	Rr,b	Пропустити якщо біт у регістрі очищений	$\text{if } (Rr(b) == 0) PC = PC + 2 \text{ or } 3$	None	1/2/3

SBRS	Rr,b	Пропустити якщо біт у регістрі встановлений	if(Rr(b)==1) PC = PC + 2 or 3	None	1/2/3
SBIC	P,b	Пропустити якщо біт у порту очищений	if(I/O(P,b)==0) PC=PC + 2 or 3	None	1/2/3
SBIS	P,b	Пропустити якщо біт у порту встановлений	if(I/O(P,b)==1) PC=PC + 2 or 3	None	1/2/3
BRBC	s,k	Перейти якщо прапор у SREG очищений	if(SREG(s)==0) PC=PC+ k + 1	None	1/2
BRBS	s,k	Перейти якщо прапор у SREG установлений	if(SREG(s)==1) PC = PC+k+ 1	None	1/2
BREQ	k	Перейти якщо дорівнює	if(Z==1) PC = PC + k + 1	None	1/2
BRNE	k	Перейти якщо не дорівнює	if(Z==0) PC = PC + k + 1	None	1/2
BRCS	k	Перейти якщо перенос установлений	if(C==1) PC = PC + k + 1	None	1/2
BRCC	k	Перейти якщо перенос очищений	if(C==0) PC = PC + k + 1	None	1/2
BRSH	k	Перейти якщо дорівнює чи більше	if(C==0) PC = PC + k + 1	None	1/2
BRLO	k	Перейти якщо менше	if(C==1) PC = PC + k + 1	None	1/2
BRMI	k	Перейти якщо мінус	if(N==1) PC = PC + k + 1	None	1/2
BRPL	k	Перейти якщо плюс	if(N==0) PC = PC + k + 1	None	1/2
BRGE	k	Перейти якщо більше чи дорівнює (зі знаком)	if(S==0) PC = PC + k + 1	None	1/2
BRLT	k	Перейти якщо менше (зі знаком)	if(S==1) PC = PC + k + 1	None	1/2
BRHS	k	Перейти якщо прапор внутрішнього переносу встановлений	if(H==1) PC = PC + k + 1	None	1/2
BRHC	k	Перейти якщо прапор внутрішнього переносу очищений	if(H==0) PC = PC + k + 1	None	1/2
BRTS	k	Перейти якщо прапор T встановлений	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Перейти якщо прапор T очищений	if(T==0) PC = PC + k + 1	None	1/2
BRVS	k	Перейти якщо прапор переповнення встановлений	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Перейти якщо прапор переповнення очищений	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Перейти якщо переривання дозволені	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Перейти якщо переривання заборонені	if(I==0) PC = PC + k + 1	None	1/2

Виконувати арифметико-логічні операції й операції читання безпосередньо над змістом комірок пам'яті не можна. Не можна також записати константу чи очистити вміст комірки пам'яті.

Система команд AVR дозволяє лише виконувати операції обміну даними між осередками SRAM і регістрами загального призначення.

Перевагами системи команд можна вважати різноманітні режими адресації комірок пам'яті.

Усі регістри введення/виведення можуть зчитуватися і записуватися через регістри загального призначення за допомогою команд IN, OUT.

Безпосередня установка і скидання окремих розрядів цих регістрів виконується командами SBI і CBI. Команди умовних переходів у якості своїх операндів можуть мати як біти-ознаки результату операції, так і окремі розряди регістрів введення/виведення, що побітно адресуються.

Інструкції передачі даних

Мнемоніка	Операнди	Опис	Операція	Прапори	Цикли
MOV	Rd,Rr	Скопіювати регістр	$Rd = Rr$	None	1
LDI	Rd,K8	Завантажити константу	$Rd = K$	None	1
LDS	Rd,k	Пряме завантаження	$Rd = (k)$	None	2*
LD	Rd,X	Непряме завантаження	$Rd = (X)$	None	2*
LD	Rd,X+	Непряме завантаження з пост-інкрементом	$Rd=(X),$ $X=X+1$	None	2*
LD	Rd,-X	Непряме завантаження з пре-декрементом	$X=X-1, Rd=(X)$	None	2*
LD	Rd,Y	Непряме завантаження	$Rd = (Y)$	None	2*
LD	Rd,Y+	Непряме завантаження з пост-інкрементом	$Rd=(Y), Y=Y+1$	None	2*
LD	Rd,-Y	Непряме завантаження з пре-декрементом	$Y=Y-1, Rd= (Y)$	None	2*
LDD	Rd,Y+q	Непряме завантаження з заміщенням	$Rd = (Y+q)$	None	2*
LD	Rd,Z	Непряме завантаження	$Rd = (Z)$	None	2*
LD	Rd,Z+	Непряме завантаження з пост-інкрементом	$Rd= (Z), Z=Z+1$	None	2*
LD	Rd,-Z	Непряме завантаження з пре-декрементом	$Z=Z-1, Rd = (Z)$	None	2*
LDD	Rd,Z+q	Непряме завантаження з заміщенням	$Rd = (Z+q)$	None	2*
STS	k,Rr	Пряме збереження	$(k) = Rr$	None	2*
ST	X,Rr	Непряме збереження	$(X) = Rr$	None	2*
ST	X+,Rr	Непряме збереження з пост-інкрементом	$(X)=Rr, X=X+1$	None	2*
ST	-X,Rr	Непряме збереження з пре-декрементом	$X=X-1, (X)=Rr$	None	2*
ST	Y,Rr	Непряме збереження	$(Y) = Rr$	None	2*
ST	Y+,Rr	Непряме збереження з пост-інкрементом	$(Y)=Rr, Y=Y+1$	None	2
ST	-Y,Rr	Непряме збереження з пре-декрементом	$Y=Y-1, (Y)= Rr$	None	2
ST	Y+q,Rr	Непряме збереження з заміщенням	$(Y+q) = Rr$	None	2
ST	Z,Rr	Непряме збереження	$(Z) = Rr$	None	2
ST	Z+,Rr	Непряме збереження з пост-інкрементом	$(Z)= Rr, Z=Z+1$	None	2
ST	-Z,Rr	Непряме збереження з пре-декрементом	$Z=Z-1, (Z) = Rr$	None	2
ST	Z+q,Rr	Непряме збереження з заміщенням	$(Z+q) = Rr$	None	2
LPM	Нет	Завантаження з програмної пам'яті	$R0 = (Z)$	None	3
LPM	Rd,Z	Завантаження з програмної пам'яті	$Rd = (Z)$	None	3
LPM	Rd,Z+	Завантаження з програмної пам'яті з пост-інкрементом	$Rd=(Z), Z=Z+1$	None	3
SPM	Нет	Збереження в програмній пам'яті	$(Z) = R1:R0$	None	-
IN	Rd,P	Читання порту	$Rd = P$	None	1
OUT	P,Rr	Запис у порт	$P = Rr$	None	1
PUSH	Rr	Занесення регістра в стек	$STACK = Rr$	None	2
POP	Rd	Витяг регістра зі стека	$Rd = STACK$	None	2

Інструкції роботи з бітами

Мнемоніка	Операнди	Опис	Операція	Прапори	Цикли
LSL	Rd	Логічний зсув вліво	$Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)$	Z,C,N,V,H,S	1
LSR	Rd	Логічне зрушення вправо	$Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)$	Z,C,N,V,S	1
ROL	Rd	Циклічне зрушення вліво через C	$Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)$	Z,C,N,V,H,S	1
ROR	Rd	Циклічне зрушення вправо через C	$Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)$	Z,C,N,V,S	1
ASR	Rd	Арифметичне зрушення вправо	$Rd(n)=Rd(n+1), n=0,...,6$	Z,C,N,V,S	1
SWAP	Rd	Перестановка тетрад	$Rd(3..0)=Rd(7..4), Rd(7..4)=Rd(3..0)$	None	1
BSET	s	Установка прапора	$SREG(s) = 1$	SREG(s)	1
BCLR	s	Очищення прапора	$SREG(s) = 0$	SREG(s)	1
SBI	P,b	Установити біт у порту	$I/O(P,b) = 1$	None	2
CBI	P,b	Очистити біт у порту	$I/O(P,b) = 0$	None	2
BST	Rr,b	Зберегти біт з регістра в T	$T = Rr(b)$	T	1
BLD	Rd,b	Завантажити біт з T у регістр	$Rd(b) = T$	None	1
SEC	Hi	Установити прапор переносу	$C = 1$	C	1
CLC	Hi	Очистити прапор переносу	$C = 0$	C	1
SEN	Hi	Установити прапор негативного числа	$N = 1$	N	1
CLN	Hi	Очистити прапор негативного числа	$N = 0$	N	1
SEZ	Hi	Встановити прапор нуля	$Z = 1$	Z	1
CLZ	Hi	Очистити прапор нуля	$Z = 0$	Z	1
SEI	Hi	Встановити прапор переривань	$I = 1$	I	1
CLI	Hi	Очистити прапор переривань	$I = 0$	I	1
SES	Hi	Установити прапор числа зі знаком	$S = 1$	S	1
CLN	Hi	Очистити прапор числа зі знаком	$S = 0$	S	1
SEV	Hi	Установити прапор переповнення	$V = 1$	V	1
CLV	Hi	Очистити прапор переповнення	$V = 0$	V	1
SET	Hi	Установити прапор T	$T = 1$	T	1
CLT	Hi	Очистити прапор T	$T = 0$	T	1
SEH	Hi	Установити прапор внутрішнього переносу	$H = 1$	H	1
CLH	Hi	Очистити прапор внутрішнього переносу	$H = 0$	H	1
NOP	Hi	Немає операції	Hi	None	1
SLEEP	Hi	Спати (зменшити енергоспоживання)	Дивитися опис інструкції	None	1
WDR	Hi	Скидання сторожового таймера	Дивитися опис інструкції	None	1

Асемблер не розрізняє регістр символів. Операнди можуть бути таких видів:

- Rd: результатуючий і вихідний регістр;
- Rr: вихідний регістр;

- **b**: константа (3 біти), може бути константний вираз;
- **s**: константа (3 біти), може бути константний вираз;
- **P**: константа (5-6 біт), може бути константний вираз;
- **K6**: константа (6 біт), може бути константний вираз;
- **K8**: константа (8 біт), може бути константний вираз;
- **k**: константа, може бути константний вираз;
- **q**: константа (6 біт), може бути константний вираз;
- **Rdl**: R24, R26, R28, R30 для інструкцій ADIW і SBIW;
- **X,Y,Z**: регістри непрямої адресації (X=R27:R26, Y=R29:R28, Z=R31:R30).

Додаток 2

Програма, що реалізує **півкроковий** режим обертання **уніполярного крокового двигуна** відповідно до діаграми на рис.10.

start:

```

    ldi R16, 0x00
    ldi R17, 0xFF
    ;Порт В на вихід з низьким початковим рівнем
    out DDRB, R17
    out PORTB, R16
    ldi R21, 0x03          ; завантаження лічильника циклів роботи програми
M1:  ldi R20, 0x01          ; завантаження «0001» для виводу в порт В
    out PORTB, R20        ; вивід вмісту R20 в порт В
    ldi R20, 0x03          ; завантаження «0011» для виводу в порт В
    out PORTB, R20        ; вивід вмісту R20 в порт В
    ldi R20, 0x02          ; завантаження «0010» для виводу в порт В
    out PORTB, R20        ; вивід вмісту R20 в порт В
    ldi R20, 0x06          ; завантаження «0110» для виводу в порт В
    out PORTB, R20        ; вивід вмісту R20 в порт В
    ldi R20, 0x04          ; завантаження «0100» для виводу порт В
    out PORTB, R20        ; вивід вмісту R20 в порт В
    ldi R20, 0x0C          ; завантаження «1100» для виводу порт В
    out PORTB, R20        ; вивід вмісту R20 в порт В
    ldi R20, 0x08          ; завантаження «1000» для виводу порт В
    out PORTB, R20        ; вивід вмісту R20 в порт В
    ldi R20, 0x09          ; завантаження «1001» для виводу порт В
    out PORTB, R20        ; вивід вмісту R20 в порт В
    dec R21                ; декремент лічильника циклів R21
    brne M1                ; перехід на мітку, якщо вміст R21 не нуль
    nop

```