НАВЧАЛЬНО-МЕТОДИЧНА ЛІТЕРАТУРА

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Тернопільський національний технічний університет

імені Івана Пулюя

Кафедра автоматизації технологічних

процесів та виробництв

Методичні вказівки
до лабораторної роботи №9
Виконання арифметичних, логічних операцій, вводу/виводу та запису в пам'ять на програмному симуляторі AVR Simulator IDE
з курсу
"Мікропроцесорні та програмні засоби автоматизації"

Тернопіль 2020

Методичні вказівки до лабораторної роботи №9 **“Виконання арифметичних, логічних операцій, вводу/виводу та запису в пам’ять на програмному симуляторі 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 №9

Виконання арифметичних, логічних операцій, вводу/виводу та запису в пам'ять на програмному симуляторі AVR Simulator IDE

1. Інтерфейс програмного симулятора AVRSimulator IDE

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

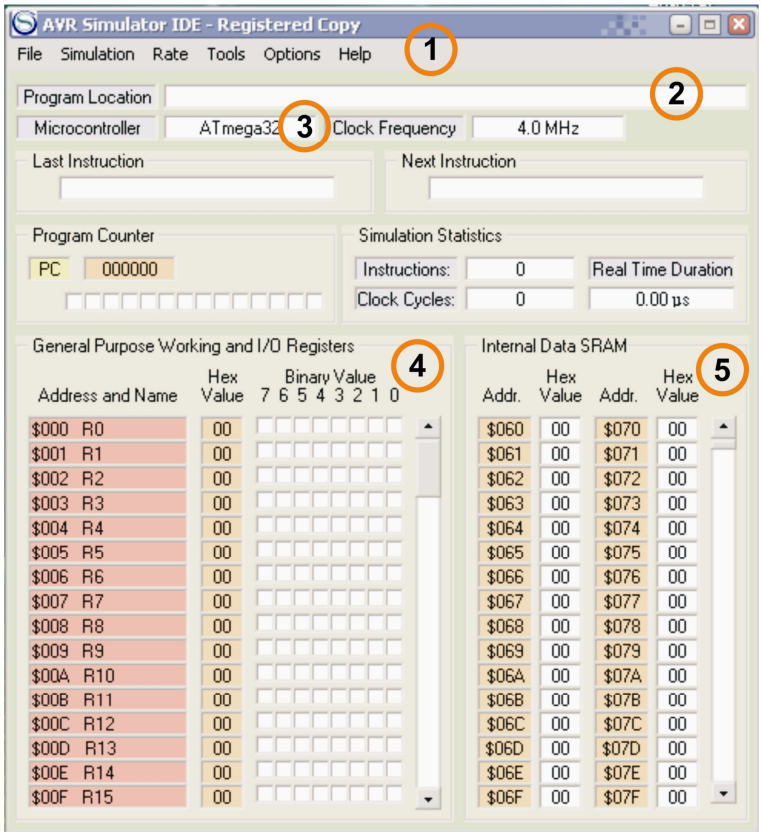

Рис. 1. Основне вікно програми AVR Simulator IDE

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.

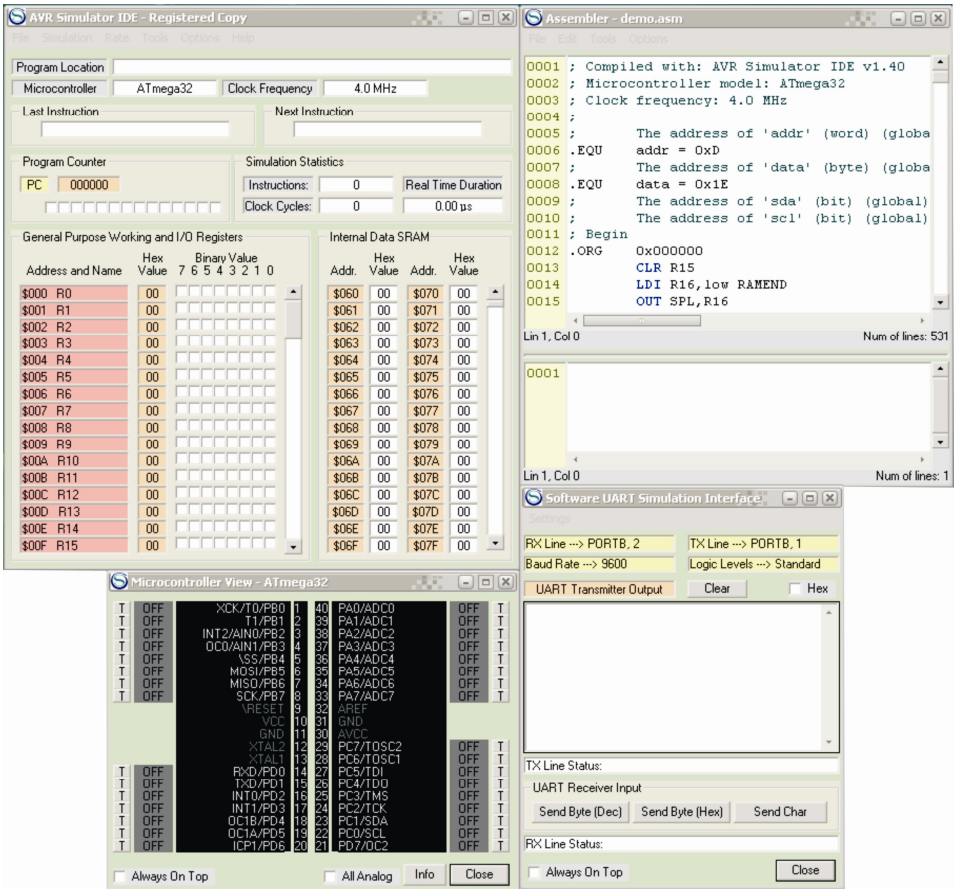

Рис. 2 Вікно симулятора з полем компілятора Assembler, апаратними виводами контролера, полем послідовного інтерфейсу

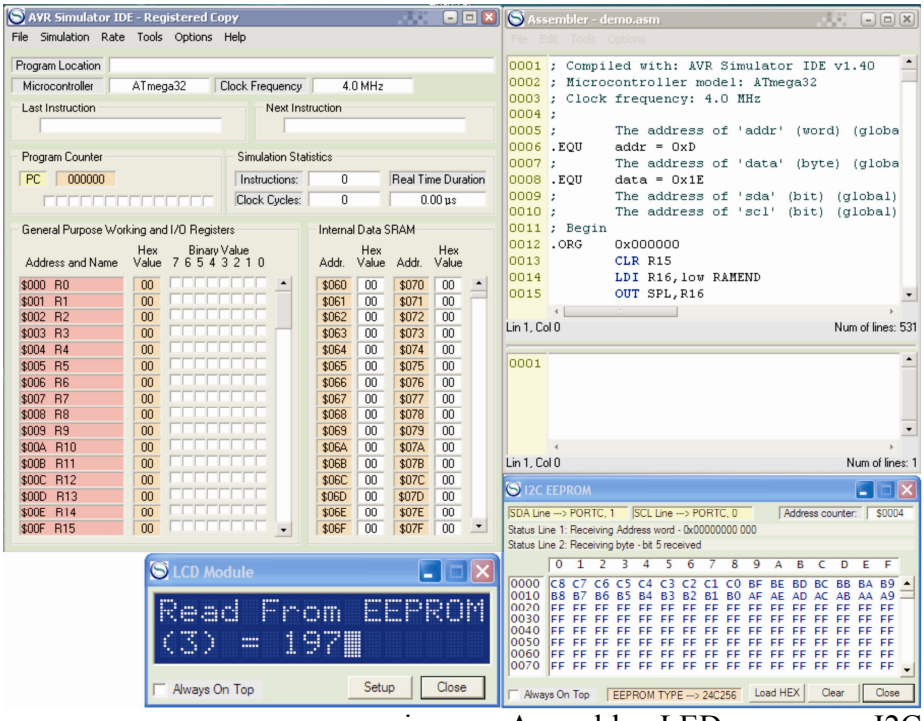

Рис. 3 Вигляд симулятора з полем компілятора Assembler, LED- модулем, I2C EEPROM

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 вивченю 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 контролера, які використовуються при виконанні команд програми, знайти в області регістрів Adress 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

5.1 **Додати два числа**, використовуючи наступну програму, за варіантами: 1)665 і 390, 2) 350 і 290. 3) 690 і 590, 4) 1030 і 780.

1. Додаються молодші байти.
2. Додаються старші байти з перенесенням, яке могло виникнути (переповнення регістра) при додаванні молодших.

```
        ; позначення
.equ            data1=665
.equ            data2=390
              ; основна програма
start:
; вираз 1
              ldi  R16,low(data1)      ; молодший байт
              ldi  R17,high(data1)     ; старший байт
; вираз 2
              ldi  R18,low(data2)      ; молодший байт
              ldi  R19,high(data2)     ; старший байт
; після додавання результат зберігається в R18, R19
              add  R18,R16             ; додаємо молодші байти
              adc  R19,R17             ; додаємо старші байти і прапор перенесення
              nop
```

З**авдання** (виконується в лабораторії)

1. Виконати программу додавання двох слів на програмному стимуляторі відповідно до вказаного викладачем варіанту.
2) За прикладом попередньої програми виконати віднімання двох операндів за варіантами: 610 і 485, 2) 350 і 290. 3) 690 і 590, 4) 1030 і 780.
3) Записати вміст регістрів PC, R17,R19, SREG в кінці виконання кожної з програм

5.2 **Перемножити два числа**, використовуючи наступну програму, за варіантами: 1)120 і 24, 2) 90 і 10. 3) 124 і 12, 4) 64 і 16.

Будь-яку операцію множення можна замінити операцією додавання.

Перемножити два числа:

```
        ; позначення
.equ            data1=120
.equ            data2=24
              ; основна програма
start:
              ldi  R16, data2          ; завантаження лічильника циклів
              ldi  R17, data1          ; константа, що додається
        ; після додавання результат зберігається в R18
M1:
              add  R18,R17             ; додаємо
              dec  R16                 ; декремент лічильника циклів
              brne  M1                 ; якщо R16 не нуль, то перехід на мітку M1
              nop
```

Операція додавання буде виконуватися доти, поки вміст регістра R16 не стане рівним нулю.

З**авдання** (виконується в лабораторії)

1. Виконати программу множення двох байтів на програмному стимуляторі відповідно до вказаного викладачем варіанту.

2) Записати вміст регістрів PC, R16, R17,R18, SREG в кінці виконання кожної з програм

5.3 **Виконати операції множення/ділення на 2^n**

Операція множення на 2^n може бути виконана за допомогою операції зсуву вмісту регістра вліво (←), а операція ділення – вправо (→), ***n*** разів.

З**авдання** (виконується в лабораторії)

1. Виконати программу ділення двох чисел на програмному стимуляторі відповідно до вказаного викладачем варіанту: 1) 296 і 8, 2) 360 і 16. 3) 680 і 4, 4) 480 і 8.

Для роботи з словами, які розбиті на байти, використовують інструкції зсуву через прапор перенесення.

```
        ; виконати операцію 256/8=32
        ; позначення
.equ            data1=256
 ; основна програма
start:
        ldi  R16, low(data1)      ; молодший байт
        ldi  R17, high(data1)     ; старший байт
        ; виконаємо операцію ділення
        clc                       ; прапор C скидаємо в 0
        ror  R17                  ; зсув вправо через C старший байт
        ror  R16                  ; зсув вправо через C молодший байт
        clc                       ; прапор C скидаємо в 0
        ror  R17                  ; зсув вправо через C старший байт
        ror  R16                  ; зсув вправо через C молодший байт
        clc                       ; прапор C скидаємо в 0
        ror  R17                  ; зсув вправо через C старший байт
        ror  R16                  ; зсув вправо через C молодший байт
        nop
```

2) За прикладом попередньої програми виконати множеня двох операндів за варіантами: 1) 356 і 8, 2) 560 і 16. 3) 160 і 4, 4) 230 і 8.

3) Записати вміст регістрів PC, R16,R17, SREG в кінці виконання кожної з програм

4) Переконатися в правильності виконаної операції, перевівши шістнадцятковий код вмісту регістрів r16, R17 в десяткові числа.

5.4. **Виконати логічне АБО** вмісту двох регістрів, в які попередньо записати константи відповідно до варіантів: 1)665 і 390, 2) 350 і 290. 3) 690 і 590, 4) 1030 і 780.

```
        ; позначення
.equ            data1=665
.equ            data2=390
                ; основна програма
start:
; вираз 1
                ldi  R16,low(data1)      ; молодший байт
                ldi  R17,high(data1)     ; старший байт
; вираз 2
```

```
                ldi  R18,low(data2)      ; молодший байт
                ldi  R19,high(data2)     ; старший байт
; після виконання операції результат зберігається в R18, R19
                ori  R18,R16             ; додаємо молодші байти
                ori  R19,R17             ; додаємо старші байти
                nop
```

5.5. **Виконати логічне І** вмісту двох регістрів, в які попередньо записати константи відповідно до варіантів: 1)120 і 24, 2) 90 і 10. 3) 124 і 12, 4) 64 і 16.

```
        ; позначення
.equ            data1=24
.equ            data2=120
                ; основна програма
start:
; вираз 1
                ldi  R16,low(data1)      ; молодший байт
                ldi  R17,high(data1)     ; старший байт
; вираз 2
                ldi  R18,low(data2)      ; молодший байт
                ldi  R19,high(data2)     ; старший байт
; після виконання операції результат зберігається в R18, R19
                ori  R18,R16             ; «І» молодших байтів
                ori  R19,R17             ; «І» старших байтів
                nop
```

Записати вміст регістрів PC, R18,R19, SREG в кінці виконання кожної з програм

5.6 **Записати константу в пам'ять SRAM**

1) Записати байт даних за вказаною адресою відповідно до варіанту: 1) 0x22–> 0x060, 2) 0x15–> 0x070, 3) 0x18–> 0x065, 4) 0x48–> 0x078.

Для запису в SRAM в асемблері AVR передбачено 12 команд.

Більшість з них є однотиповими та відрізняються лише використанням різних індексних регістрових пар X, Y, Z.

Найпростішою є команда sts, яка завантажує у вказану адресу комірки пам'яті значення з регістра загального призначення (**запис за прямою адресою**, що вказується в команді).

Необхідно пам'ятати, що пам'ять SRAM для мікроконтролера ATmega32 починається з адреси 0x060.

```
start:
        .equ            data=0x22
        .equ            adr=0x60
                        ; основна програма
        ldi  R17,0x04           ; завантажити число в лічильник циклів
        ldi  R16,data           ; завантажити число в регістр
M1:     sts  adr+1,R16          ; завантажити вміст регістра в SRAM за прямою адресою
        inc R16                 ; вміст регістра збільшити на 1
        dec  R17                ; зменшити вміст лічильника циклів
        brne  M1                ; перейти за міткою M1, якщо вміст лічильника не нуль
        nop
```

Інший варіант - запис в SRAM **за непрямою адресою** (міститься в парі регістрів):

```
start:
.equ        data=0x22
.equ        adr=0x60
        ; основна програма
ldi  R17,0x04           ; завантажити число в лічильник циклів
ldi  R16, data          ; завантажити число в регістр
ldi ZL, low(adr)        ; Z ← значення адреси
ldi ZH, high(adr)
M1:  st Z+, R16         ; SRAM ← r16; (Z+): запис в SRAM  за непрямою адресою і
                        ; збільшення адреси на 1
     inc R16            ; вміст регістра збільшити на 1
     dec  R17           ; зменшити вміст лічильника циклів
     brne  M1           ; перейти за міткою M1, якщо вміст лічильника не нуль
     nop
```

5.7 **Записати константи в пам'ять EEPROM**

1) Записати байт даних за вказаною адресою відповідно до варіанту: 1) 0x22–> 0x010, 2) 0x15–> 0x020, 3) 0x18–> 0x030, 4) 0x48–> 0x040.

Алгоритм запису одного байту в EEPROM-пам'ять:

1. Дочекатися готовності EEPROM (поки не скинеться біт EEWE).
2. Завантажити байт даних у регістр EEDR, а необхідну адресу - у регістр EEAR.
3. Встановити в «1» біт EEMWE регістра керування EECR.
4. Записати у розряд EEWE регістра керування логічну «1» протягом 4-х тактів.

EECR – регістр керування доступом до EEPROM-пам'яті. У ньому задіяні 4 біти для визначення поведінки МК при роботі з EEPROM.

3	EERIE	Дозвіл на переривання по завершенню запису в EEPROM
2	EEMWE	Попередній дозвіл на запис даних. Після нього повинен одразу бути встановлений біт EEWE, інакше він буде апаратно скинутий через 4 такти.
1	EEWE	Дозвіл на запис. При встановленні в 1 відбувається запис даних у EEPROM (при умові, що встановлений біт EEMWE).
0	EERE	Дозвіл на читання. При встановленні в 1 відбувається читання з EEPROM. По завершенню читання цей розряд скидається апаратно.

```
        ; позначення
.equ        data=0x22
.equ        adr=0x010
          ; основна програма
start:
        ldi R18,data1
        ldi R17,high(adr)       ; старший байт адреси
        ldi R16,low(adr)        ; молодший байт адреси
        out EEARH,R17           ; завантаження в адресний регістр старшого байту
        out EEARL,R16           ; завантаження в адресний регістр молодшого байту
        out EEDR,R18             ; завантаження даних
        cli
        sbi EECR, EEMWE         ; вст. попередній дозвіл на запис
```

```
sbi EECR, EEWE          ; вст. остаточний дозвіл на запис
sei
nop
```

5.8 **Записати константи в пам'ять FLASH**

1) Записати байт даних в пам'ять програм FLASH, використовуючи директиву **db**, та скопіювати вміст пам'яті в регістр R17 відповідно до вказаного варіанту:

Варіанти: 1)0x12,0x010,0x36,0x19; 2)0x15,0x020,0x10,0x44; 3)0x18,0x030; 4)0x48,0x90,0x29,0x64,0x33,0x72.

Для початку програми **необхідно обов'язково вибрати сегмент пам'яті FLASH** директивою **.cseg.**

Перед виконанням програми на симуляторі в опції «Tools» натиснути курсором на рядок «Program Memory Editor". З'явиться одноіменне **вікно з програмою**, в якому після запуску програми на виконання на початку сегменту запишуться дані, завантажені директивою db, а після них – виконувана програма.

Слід пам'ятати, що читання з пам'яті FLASH здійснюється з використанням **регістра непрямої адресації Z.**

```
.cseg                       ; вибір сегменту пам'яті FLASH
adr: .db 0x12,0x22,0x18,0x36  ; завантаження в пам'ять даних за адресою з міткою adr
     ;основна програма
start:
ldi ZL, adr                 ; завантаження байту адреси в регістр Z
ldi R16,0x04                ; завантаження лічильника циклів. Його вміст має
                            ; відповідати кількості байтів, завантажених в пам'ять
M1:
lpm  R17,Z+                 ; завантаження першого байту з FLASH пам'яті в регістр,
                            ; автоматично вміст регістра Z збільшиться на 1
dec R16                     ; організація циклу для завантаження наступних байтів
brne M1
nop
```

5.9 **Виконати підпрограму реалізації часової затримки** відповідно до вказаного варіанту: 1)на 10 машинних циклів, 2) 20 машинних циклів ів, 3) 300 машинних циклів.

Один цикл мікроконтролером виконується за 0,5 мкс.

```
     ; позначення
.equ          data=3
     ; основна програма
start:
          ldi  R16, data        ; завантаження лічильника циклів
M1:
          dec  R16              ; декремент лічильника циклів
          brne  M1              ; якщо R16 не нуль, то перехід на мітку M1
          nop
```

5.10 **Виконати підпрограму вводу/виводу через порти** відповідно до вказаного варіанту: 1) порт А на вивід, порт В – на ввід, 2) порт В на вивід, порт А – на ввід, 3) порт А на вивід, порт С – на ввід, 4) порт В на вивід, порт С– на ввід.

Виставити курсором активні значення на лініях порту, що працюють на ввід, і отримати введені значення на відповідних лініях порту, що працює на вивід (зафарбовуються зеленим кольором)..

```
     ; основна програма
```

```
start:
        ldi R16, 0x00
        ldi R17, 0xFF
                ;Порт A на вхід з підтягуючим резистором
        out DDRA, R16
        out PORTA, R17
                ;Порт B на вихід з низьким початковим рівнем
        out DDRB, R17
        out PORTB, R16
        in R16, PINA
        out PORTB, R16
        nop
```

Контрольні запитання

1. Використання AVR-мікроконтролерів.
2. Арифметичні та логічні команди мікроконтролера.
3. Програмування таймерів-лічильників.
4. Формат та використання регістрів загального призначення.
5. Призначення та позначення основних елементів програмної моделі мікроконтролера.

Література

1. Програмування мікроконтролерів систем автоматики: конспект лекцій для студентів базового напряму 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 Tiny и Mega фирмы ATMEL, – [5-е изд., стер.] / Евстифеев А. В. – М.: Издательский дом «Додэка-XXI», 2008. 560 с.

Додаток 1
Система команд мікроконтролерів AVR

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 функцію 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 И	Rd = Rd · Rr	Z,N,V,S	1
ANDI	Rd,K8	Логічне И с константою	Rd = Rd · K8	Z,N,V,S	1
OR	Rd,Rr	Логічне АБО	Rd = Rd V Rr	Z,N,V,S	1
ORI	Rd,K8	Логічне АБО з константою	Rd = Rd V K8	Z,N,V,S	1
EOR	Rd,Rr	Логічне що виключає АБО	Rd = Rd EOR Rr	Z,N,V,S	1
COM	Rd	Побітна Інверсія	Rd = $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Зміна знака (Доп. код)	Rd = $00 - Rd	Z,C,N,V,H,S	1
SBR	Rd,K8	Установити біт (біти) у регістрі	Rd = Rd V K8	Z,C,N,V,S	1
CBR	Rd,K8	Скинути біт (біти) у регістрі	Rd = Rd · ($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 · 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	Rdl,K6	Скласти константу і слово	Rdh:Rdl=Rdh:Rdl+ K6	Z,C,N,V,S	2
SBIW	Rdl,K6	Вичитати константу зі слова	Rdh:Rdl=Rdh:Rdl - K 6	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	if (Rd ==Rr) PC = PC 2 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	if(Rr(b)==0) PC = PC + 2 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: результуючий 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).