ФАКУЛЬТЕТ АВТОМАТИКИ, ТЕЛЕМЕХАНІКИ ТА ЗВ'ЯЗКУ

Кафедра „Спеціалізовані комп'ютерні системи"

ВИВЧЕННЯ АРХІТЕКТУРИ, СИСТЕМИ КОМАНД І ОСНОВНИХ ПРИНЦИПІВ ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт з дисциплін

«*МІКРОПРОЦЕСОРНІ ПРИСТРОЇ*»,

«*МІКРОПРОЦЕСОРНА ТЕХНІКА*»,

«*ПРИСТРОЇ ІНФОРМАЦІЙНО-УПРАВЛЯЮЧИХ СИСТЕМ*»

Харків – 2010

Методичні вказівки розглянуто та рекомендовано до друку на засіданні кафедри «Спеціалізовані комп'ютерні

системи» 5 березня 2009 р., протокол № 6.

Методичні вказівки до виконання лабораторних робіт з вивчення технічних засобів мікропроцесорних систем містять опис чотирьох лабораторних робіт, присвячених вивченню мікроконтролерів AVR на прикладі мікроконтролера ATmega 128 в інтегрованому середовищі розробки AVR Studio 4.13 з використанням ПЕОМ. Включають короткі теоретичні відомості, необхідні для успішного виконання завдання, опис програмно-апаратного забезпечення, перелік розрахункових і експериментальних завдань, методику виконання експериментальної частини роботи, контрольні питання.

Методичні вказівки призначені для студентів спеціальностей «Автоматика й автоматизація на транспорті», які вивчають курси «Мікропроцесорні пристрої», «Мікропроцесорна техніка», «Пристрої інформаційно-управляючих систем», денної й заочної форм навчання.

Укладачі:

проф. Б.Т. Ситнік,
доц. А.В. Мамонов,
асист. В.О. Бриксін

Рецензент

доц. В.С. Коновалов

ВИВЧЕННЯ АРХІТЕКТУРИ, СИСТЕМИ КОМАНД І ОСНОВНИХ ПРИНЦИПІВ ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ

Методичні вказівки
до лабораторних робіт з дисциплін

«Мікропроцесорні пристрої»,
«Мікропроцесорна техніка»,
«Пристрої інформаційно-управляючих систем»

Відповідальний за випуск Мамонов А.В.

Редактор Решетилова В.В.

Підписано до друку 17.04.09 р.
Формат паперу 60x84 1/16 . Папір писальний.
Умовн.-друк.арк. 2,5. Обл.-вид.арк. 2,75.
Замовлення № Тираж 100. Ціна

Видавництво УкрДАЗТу, свідоцтво ДК № 2874 від. 12.06.2007 р.
Друкарня УкрДАЗТу,
61050, Харків - 50, майд. Фейєрбаха, 7

УКРАЇНСЬКА ДЕРЖАВНА АКАДЕМІЯ ЗАЛІЗНИЧНОГО ТРАНСПОРТУ

ФАКУЛЬТЕТ АВТОМАТИКИ, ТЕЛЕМЕХАНІКИ ТА ЗВ'ЯЗКУ

Кафедра «Спеціалізовані комп'ютерні системи»

ВИВЧЕННЯ АРХІТЕКТУРИ, СИСТЕМИ КОМАНД І ОСНОВНИХ ПРИНЦИПІВ ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт з дисциплін

«Мікропроцесорні пристрої», «Мікропроцесорна техніка», «Пристрої інформаційно-управляючих систем»

Харків 2010

Методичні вказівки розглянуто та рекомендовано до друку на засіданні кафедри «Спеціалізовані комп'ютерні системи» 5 березня 2009 р., протокол № 6.

Методичні вказівки до виконання лабораторних робіт з вивчення технічних засобів мікропроцесорних систем містять опис чотирьох лабораторних робіт, присвячених вивченню мікроконтролерів AVR на прикладі мікроконтролера ATmega 128 в інтегрованому середовищі розробки AVR Studio 4.13 з використанням ПЕОМ. Включають короткі теоретичні відомості, необхідні для успішного виконання завдання, опис програмно-апаратного забезпечення, перелік розрахункових і експериментальних завдань, методику виконання експериментальної частини роботи, контрольні питання.

Методичні вказівки призначені для студентів спеціальностей «Автоматика й автоматизація на транспорті», які вивчають курси «Мікропроцесорні пристрої», «Мікропроцесорна техніка», «Пристрої інформаційно-управляючих систем», денної й заочної форм навчання.

Укладачі:

проф. Б.Т. Ситник,
доц. А.В. Мамонов,
асист. В.О. Бриксін

Рецензент

доц. В.С. Коновалов

ЗМІСТ

ВСТУП

Лабораторний практикум з дисциплін «Мікропроцесорні пристрої», «Мікропроцесорна техніка», «Пристрої інформаційно-управляючих систем» призначений для надання допомоги студентам у засвоєнні й закріпленні знань з відповідних дисциплін, набуття практичних навичок у процесі самостійного розв'язання ряду задач, у яких використовується мікропроцесорна техніка або проектуються компоненти систем керування.

Готуючись до лабораторних робіт, студент зобов'язаний:

- вивчити теоретичний матеріал відповідної лабораторної роботи;
- виконати домашнє завдання й продумати відповіді на контрольні питання;
- вивчити програму й методику лабораторних досліджень, підготувати необхідні розрахунки відповідно до завдання й попередні варіанти лістингів програм.

У процесі виконання лабораторної роботи студент провадить налагодження програми й досліджує її роботу.

Щоб захистити лабораторну роботу, студент повинен виконати такі три етапи:

- виконати домашнє завдання;
- виконати на лабораторному стенді практичну частину роботи, пояснюючи хід її виконання викладачеві й відповідаючи на питання;
- підготувати і захистити звіт про проведену роботу.

Курс лабораторних робіт проводиться з використанням ПЕОМ типу IBM PC/AT у середовищі програмування AVR Studio 4.13 з використанням симулятора мікроконтролерів AVR сімейства MEGA. Це універсальні 8-розрядні високопродуктивні мікроконтролери, засновані на RISC ядрі і призначені для розв'язання широкого спектра задач для вбудованих систем керування.

1 ВИВЧЕННЯ АРХІТЕКТУРИ, СИСТЕМИ КОМАНД І

ОСНОВНИХ ПРИНЦИПІВ ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ

Мета лабораторних робіт: вивчити архітектуру й принципи програмування мікроконтролера AVR ATmega128 на прикладі розроблення програм мовою асемблера; освоїти користувальницький інтерфейс середовища програмування AVR Studio.

1.1 Домашнє завдання

Перед роботою необхідно опрацювати теоретичний матеріал за літературою [1 – 4] і конспект лекцій, ознайомитися зі структурою й принципами функціонування мікроконтролера AVR ATmega128, системою команд і основами програмування мовою асемблера. При підготовці до лабораторної роботи необхідно скласти попередні варіанти лістингів програм, що вказуються у пунктах практичного виконання роботи, і зробити необхідні розрахунки відповідно до завдання.

1.2 Короткі теоретичні положення

1.2.1 Архітектура й програмна модель мікроконтролера AVR ATmega128. Механізм роботи з регістрами, пам'яттю й портами введення-виведення

У мікроконтролері AVR ATmega128 реалізована гарвардська архітектура, у відповідності до якої адресні простори пам'яті програм і даних фізично розділені (доступ до цих областей пам'яті здійснюється по роздільних шинах). Така організація дозволяє ядру процесора одночасно працювати з пам'яттю програм і даних, що підвищує швидкодію. Пам'ять програм являє собою перепрограмувальний постійний запам'ятовуючий пристрій, що стирається електри-

чно, обсягом 128 кбт, виконаний за технологією FLASH-пам'яті і призначений для зберігання команд, які керують функціонуванням мікроконтролера, а також для зберігання констант, що не змінюють своїх значень у ході виконання програми. Так як довжина команди становить 16 біт, то пам'ять програм має 16-розрядну організацію. Для адресації пам'яті програм використовується 16-розрядний регістр – програмний лічильник PC.

Пам'ять даних організована за принципом сполученої архітектури введення-виведення й розділена на три частини: регістрова пам'ять, пам'ять портів (регістрів) введення-виведення й статичний ОЗП (SRAM), розташовані в єдиному адресному просторі.

Регістрова пам'ять включає 32 8-розрядних регістри загального призначення (РЗП) R0 – R31, об'єднаних у регістровий файл. Кожний з регістрів загального призначення безпосередньо пов'язаний з арифметико-логічним пристроєм (АЛП). Шість 8-розрядних регістрів (R26 – R31) можуть використовуватися як три 16-розрядних регістри-покажчики для непрямої адресації (таблиця 1.1).

Таблиця 1.1 – 16-розрядні регістри X, Y, Z, що використовуються для непрямої адресації пам'яті

Регістр-покажчик	**X**		**Y**		**Z**	
Позначення байта	**XH**	**XL**	**YH**	**YL**	**ZH**	**ZL**
Регістр	R27	R26	R29	R28	R31	R30

Безпосередньо пам'ять даних являє собою статичний ОЗП (SRAM) обсягом 4 кбт, що займає діапазон адрес 0100h-10FFh. Доступ до статичного ОЗП даних може бути легко здійснений через п’ять різних режимів адресації.

Регістр стану SREG розташований в області введення-виведення й містить інформацію про поточний стан мікроконтролера.

При генерації переривання й виклику підпрограм

адреса повернення із програмного лічильника записується в стек. Стек також використовується для зберігання тимчасових даних і локальних змінних. Стек розташовують у статичному ОЗП пам'яті даних. У програмі до виклику будь-якої процедури або дозволу переривань повинна бути виконана ініціалізація покажчика стека (SP), тобто область стека в статичному ОЗП пам'яті даних повинна бути визначена програмно. Покажчик стека SP доступний для читання й запису в *просторі введення-виведення.* Регістр покажчика стека вказує на вершину стека. Зверніть увагу на організацію стека, що спрямовується від старших у молодші позиції статичного ОЗП. Це означає, що команда поміщення в стек **push** зменшує значення покажчика стека.

Покажчик стека однократно декрементується при поміщенні даних у стек інструкцією **push** і двічі декрементується при поміщенні в стек адреси повернення. Покажчик стека однократно інкрементується при витяганні даних зі стека інструкцією **pop** і двічі інкрементується при витяганні адреси повернення при виконанні інструкції виходу з підпрограми **ret** або виходу із процедури обробки переривань.

1.2.2 Система команд мікроконтролера AVR ATmega128

Базовий набір команд мови асемблера для мікроконтролерів AVR містить 120 інструкцій, які можна розділити на 4 групи: команди пересилання даних; арифметичні й логічні команди; інструкції для роботи з бітами; команди керування ходом виконання програми.

Розглянемо тільки ті команди, які використовуються в даних лабораторних роботах.

Команди пересилання даних. Група команд пересилання даних містить у собі інструкції із завантаження значень констант, пересилання даних типу регістр-регістр, регістр-пам'ять, регістр-порт введення-виведення. Команди даної групи є двохоперандними, причому першим

операндом є приймач даних, а другим – джерело даних.

Команда завантаження констант **ldi R_d, K_8** застосовується для запису безпосереднього значення (байта) K_8 у регістр-приймач R_d. Як регістр-приймач можуть використовуватися регістри загального призначення R16 – R31. Якщо константа представлена у двійковій або шістнадцятковій системах числення, то перед значенням константи K_8 необхідно вказати специфікатор системи числення 0b – для двійкової, 0x (або $) – для шістнадцяткової відповідно.

Приклади:

```
ldi R16, 125          ; Завантаження в R16 десяткового
                      ; числа 125
ldi R20, 0xFF         ; Завантаження в R20 шістнадцятко-
                      ; вої константи FFh
ldi R23, 0b11011001   ; Завантаження в R23 двійкової конс-
                      ; танти 11011001
```

Команда пересилання даних між регістрами **mov R_d, R_s** використовується для пересилання значення з регістра-джерела R_s у регістр-приймач R_d. Операнди в команді є винятково регістрами загального призначення R0 – R31.

Приклад:

```
mov R16, R0           ; Завантаження в R16 значення з регістра
                      ; R0
```

Команди пересилання даних між регістром і коміркою пам'яті з використанням непрямої адресації, коли адреса комірки пам'яті заноситься в один з 16-розрядних регістрів X,Y,Z (див. таблицю 1.1), мають такий формат:

st (R_{16}), R_s – завантаження даних з регістра загального призначення R_s у комірку пам'яті, адреса якої знаходиться в 16-розрядному регістрі R_{16};

ld R_d, (R_{16}) – завантаження в регістр загального призначення Rd даних з комірки пам'яті, адреса якої знаходиться в 16-розрядному регістрі R_{16}.

Приклади:

```
ld R2, X        ; Завантаження в R2 значення з комірки
                ; пам'яті за адресою, указаною в X
st Y, R5        ; Завантаження значення з регістра R5 в
                ; пам'ять за адресою, указаною в Y
```

Команди пересилання даних між регістром і коміркою пам'яті з використанням прямої адресації, коли адреса комірки пам'яті прямо вказується в команді, мають такий формат:

lds R_d, K_{16} – завантаження в регістр загального призначення R_d даних з комірки пам'яті, 16-розрядна адреса якої K_{16}.

sts K_{16}, R_s – завантаження даних з регістра загального призначення Rs у комірку пам'яті, 16-розрядна адреса якої K_{16}.

Приклади:

```
lds R2, $10FF   ; Завантаження в R2 значення з комірки
                ; пам'яті ; за адресою 10FFh
sts $10FF, R5   ; Завантаження значення з регістра R5 в
                ; комірку пам'яті за адресою 10FFh
```

Для звертання до портів введення-виведення в мікроконтролері передбачені спеціальні команди **in** і **out**:

in R_d, P – уведення даних з порту з адресою P у регістр загального призначення R_d;

out P, R_s – виведення даних з регістра загального призначення R_s у порт із адресою P.

Приклади:

```
in R10, 0x15    ; Уведення даних з порту з адресою 15h в
                ; регістр загального призначення R10
out 0x2F, R8    ; Виведення даних з регістра загального
                ; призначення R8 у порт із адресою 2Fh
```

Для роботи зі стеком передбачені команди **push** і **pop**:

push R_s – збереження вмісту регістра загального призначення R у стеці;

pop R_d – завантаження 1 байта зі стека в регістр загального призначення R.

Приклади:

```
push R14        ; Збереження вмісту R14 у стеці
pop R14         ; Відновлення вмісту R14 з стека
```

Арифметичні команди. Для роботи із цілими двійковими числами цілочисельний АЛП мікроконтролера AVR ATmega128 підтримує більше десятка арифметичних і логічних команд.

Основними арифметичними командами є інструкції додавання, вирахування й множення. Операндами в командах даної групи можуть бути тільки регістри загального призначення. Результат операції (крім множення) записується за адресою першого операнда.

Команди для виконання операцій додавання й вирахування:

add Rd, Rs – команда додавання; дія: Rd←Rd+Rs;

adc R_d, R_s – команда додавання з урахуванням переносу; дія: $R_d \leftarrow R_d+R_s+C$.

sbiw R_{16}, K_6 – команда вирахування константи з регістрової пари, Y або Z; дія: Rdh:Rdl ← Rdh:Rdl – K_6, де Rdh і Rdl – старший і молодший байти регістра відповідно; K_6 = 0

– 63.

Команди змінюють прапори переносу **C**, переповнення **V**, знака **N**, **S,** і нуля **Z**.

Приклади:

```
add R10, R15        ; Дія: R10 ← R10+R15
adc R10, R15        ; Дія: R10←R10+R15+C
sbiw YL, 9          ; Дія: R29:R28 ← R29:R28 – 9
```

Команда негативного збільшення (декремента):

dec R – декремент; дія R←R–1.

У ролі операнда в цих командах допускається використовувати тільки регістр загального призначення.

Приклад:

```
dec R16             ; Дія: R16 ← R16 – 1
```

Команди переходу. Команди умовного переходу викликаються відразу після інструкцій, що викликають зміни бітів регістра стану SREG і на основі аналізу прапорів здійснюють перехід за указаною адресою (міткою) у пам'яті команд.

Одна з команд цієї групи:

brne M – перехід на мітку M, якщо нерівно;

Приклад:

```
brne label1         ; Виконати перехід на мітку label1, якщо
```

; прапор **Z**=0

Команди виклику підпрограм здійснюють перехід на виконання підпрограми й відрізняються способами адресації.

Команда відносного виклику підпрограми:

rcall k – виконує перехід до підпрограми, адреса якої утворюється додаванням умісту лічильника команд із константою **k**. Адреса наступної за **rcall** команди (2 байти) зберігається в стеці. На практиці замість числових значень зсуву адреси вказуються мітки підпрограм. Дія: STACK ← PC + 1; PC ← PC + k + 1; SP ← SP – 2.

Команда повернення з підпрограми:

ret – виконує повернення з підпрограми в те місце, звідки підпрограма була викликана; дія: SP ← SP + 2; PC ← STACK.

Приклади:

```
rcall routine    ; Викликати підпрограму з ім'ям
                 (міткою) ; routine
Ret              ; Повернення з підпрограми
```

2 ЛАБОРАТОРНА РОБОТА 1 ВИВЧЕННЯ ОСНОВНИХ ПРИНЦИПІВ ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРА

Мета роботи: вивчення системи команд мікроконтро-

лера AVR ATmega 128 і принципів його програмування мовою асемблера в середовищі AVR Studio, способів виконання й налагодження програм з використанням програмного симулятора.

Завдання: набрати в середовищі програмування AVR Studio наведену в завданні вихідну програму мовою асемблера, зробити асемблерування й виконання в різних режимах.

2.1 Програма й методика досліджень

2.1.1 Створіть папку для файлів проекту

Примітка - У назвах папок і файлів можна використовувати тільки букви латинського алфавіту, цифри й знак підкреслення. У назвах повинно бути не більше вісьмох символів і повинні бути відсутні пробіли.

2.1.2 Запустіть AVR Studio і створіть новий проект. Для цього у вікні майстра настроювання проектів **Welcome to AVR Studio 4** натисніть кнопку **New Project**.

Якщо необхідно відкрити існуючий проект, натисніть кнопку **Open** і виберіть його у вікні, що відкрилося, **Open Project File or Object File.** Нещодавно створені проекти можна вибрати з вікна **Recent projects** майстра настроювання проектів. Файл проекту має розширення *.aps. Після відкриття файлу проекту можна відкрити у вікні редактора програм необхідний файл, наприклад, файл вихідної програми з розширенням *.asm.

Прапорець **Show dialog at startup** у вікні майстра настроювання проектів повинен бути встановлений, щоб це вікно з'являлося при кожному запуску програми. Якщо це вікно закрите, то створити новий або відкрити існуючий проект можна й з головного вікна **AVR Studio** (меню **Project**).

2.1.3 У новому вікні майстра настроювання проектів,

що з'явилося (при створенні нового проекту), у полі **Project type** виділіть пункт **Atmel AVR Assembler**.

2.1.4 Установіть прапорці **Create initial file** і **Create folder** для створення файлу вихідної програми й папки, де будуть перебувати всі файли проекту. Файлу вихідної програми буде автоматично присвоєне розширення *.asm.

2.1.5 Уведіть ім'я проекту в поле **Project name.** У полі I**nitial file** з'явиться те саме ім'я, але з розширенням *.asm.

2.1.6 Натисніть кнопку «**...**» праворуч від поля **Location**. У вікні **Select folder**, що з'явилося**,** знайдіть і відкрийте папку, у яку необхідно помістити папку з файлами нового проекту (див. 2.1.1), і натисніть кнопку **Select**. Вікно закриється. У вікні майстра настроювання проектів у полі **Location** з'явиться шлях в обрану папку.

2.1.7 Натисніть кнопку **Next**. У вікні майстра настроювання проекту в полі **Debug platform** виділіть пункт **AVR Simulator,** щоб вибрати в ролі платформи для налагодження програмний симулятор. У полі **Device** виділіть тип використовуваного мікроконтролера – **ATmega128** і натисніть кнопку **Finish**. Вікно майстра настроювання проекту закриється, і відкриється вікно редактора асемблерної програми (вікно програмного коду), у заголовку якого зазначене ім'я файлу вихідної програми й шлях до нього.

2.1.8 У вікні редактора наберіть програму, у якій два однобайтних числа заносяться в регістри загального призначення й підсумовуються. Сума пересилається в РЗП і комірку пам'яті програм.

; Програма підсумовування двох однобайтних чисел

```
.org 0              ; Директива установки початкової
                    ; адреси програми
ldi   r16, $1f      ; Завантаження в регістр R16 першого
```

```
                        ; доданка
ldi   r17, $24          ; Завантаження в регістр R17 другого
                        ; доданка
add   r16, r17          ; Додавання першого й другого
                        ; доданків
mov   r21, r16          ; Пересилання суми в R21
ldi   r26, $02          ; Завантаження молодшого байта адреси
                        ; в регістр-покажчик X
ldi   r27, $01          ; Завантаження старшого байта адреси
в
                        ; регістр-покажчик X
st    x, r21            ; Пересилання суми в пам'ять даних за
                        ; адресою, зазначеною в X
nop                     ; Немає операцій
nop
```

Команди набирайте з відступом, рівним інтервалу табуляції (клавіша **Tab**), щоб залишити місце для поля мітки. Між мнемокодом операції й операндами також робіть інтервал табуляції.

Для зручності сприйняття програми редактор може автоматично виділяти кольорами структурні елементи команди (мітки, мнемокод, операнди й т.п.). Наприклад, мнемокод – синім, коментарі – зеленими кольорами й т.п. Для цього в меню редактора **Edit** передбачені настроювання кольорів (пункт **Font and Color**).

2.1.9 Якщо потрібно створити лістинг програми, то необхідно встановити прапорець **Create List file** у вікні **Assembler Options.** Це вікно відкривається з меню редактора **Project**, пункт **Assembler Options**.

2.1.10 Виконайте асемблерування (трансляцію в машинні коди) вихідної програми, вибравши з меню **Build** редактори пункт **Build**. Зручно користуватися «гарячою» клавішею **F7** клавіатури комп'ютера.

Асемблерування можна виконати й, використовуючи відповідну кнопку на панелі інструментів вікна редактора.

У вікні **Build** з'явиться повідомлення про результати асемблерування. Якщо з'явиться повідомлення про помилку асемблерування, відзначене червоною міткою, клацніть двічі на повідомленні мишкою. У вікні редактора, ліворуч від команди, у якій виявлена помилка, з'явиться маркер, що вказує на цю команду.

Виправте помилку й повторіть асемблерування.

Примітка - Всі вікна, які необхідно використати при асемблеруванні й налагодженні програми, можна відкрити з меню **View** головного вікна, а також кнопками на його панелі інструментів.

У результаті асемблерування редактор автоматично створює в папці проекту ряд файлів (вихідна програма, об'єктна, лістинг і ін.), обновляючи їх після кожного асемблерування.

2.1.11 Проведіть покрокове виконання програми.

Для цього необхідно виконати такі дії:

1) перейдіть у режим налагодження програм, вибравши в меню **Debug** команду **Start Debugging.** Можна також скористатися відповідною кнопкою на панелі інструментів.

У вікні редактора з'явиться жовта стрілка напроти першої команди, і, можливо, відкриються кілька вікон середовища, використовуваних при налагодженні програм;

2) відкрийте, якщо це необхідно, вікно **Processor**. У цьому вікні представлені: стан регістрів спеціального призначення процесора (покажчика стека, регістра прапорів SREG, лічильника команд), регістрів загального призначення, регістрів-покажчиків X, Y і Z, тактова частота, лічильник машинних циклів виконуваної програми й час її виконання;

3) відкрийте вікно регістрів (у меню **View** пункт **Register**, або кнопкою на панелі інструментів). У ньому зазначені регістри загального призначення R0 – R31 і їхній

вміст;

4) відкрийте вікно **Memory.** У списку, що розкривається, цього вікна виберіть пам'ять даних **(Data**). Кнопкою **8/16** установіть однобайтний формат подання даних у вікні пам'яті;

5) розташуйте вікна в такий спосіб (зрушуючи їх і змінюючи їхні розміри), щоб вони не заважали одне одному й були видні дані, що містяться в них;

6) виконайте першу команду вихідної програми, вибравши в меню **Debug** команду **Step Into**. Зручно користуватися клавішею **F11** клавіатури.

Після натискання **F11** виконається перша команда (**ldi r16, $1f**) і жовта стрілка встановиться напроти наступної команди.

Перегляньте результат виконання команди у вікнах регістрів, процесора й пам'яті. У відповідних місцях повинні змінитися дані. Нові дані офарблюються в червоні кольори;

7) виконайте другу команду, натиснувши **F11**. Перегляньте зміну даних у вікнах;

8) аналогічно виконайте послідовно всі команди. Після виконання кожної команди переглядайте отримані результати;

9) якщо необхідно повторити виконання програми, необхідно попередньо виконати команду скидання, вибравши в меню **Debug** команду **Reset**. При цьому скидаються в нуль регістри мікроконтролера. Пам'ять даних не обнуляється. Щоб змінити значення числа, записане в пам’ять або регістр, необхідно двічі клацнути мишкою на необхідній комірці пам'яті або вмісті регістра й у віконці, що з'явилося, внести нове значення. У цьому ж віконці можна зробити зміну формату представлення даних (шістнадцятковий, двійковий, десятковий). Формат представлення даних усього вікна можна встановити, клацнувши у вікні правою кнопкою мишки й вибравши в меню, що з'явилося, відповідний пункт.

2.1.12 Вийдіть із режиму налагодження, вибравши в меню **Debug** команду **Stop Debugging**. Програма повернеться в режим редагування. Тепер можна робити зміни у вихідній програмі, знову асемблерувати її й потім знову переходити в режим виконання програми (налагодження).

2.1.13 Внесіть помилку в яку-небудь команду вихідної програми.

2.1.14 Виконайте асемблерування (натиснувши **F7**).

2.1.15 У вікні **Duild** прочитайте повідомлення про помилку (ліворуч від нього буде знаходитися червоний маркер).

2.1.16 Двічі клацніть мишкою на цьому повідомленні. У вікні редактора ліворуч від команди, у яку внесена помилка, з'явиться маркер, що вказує на цю команду.

2.1.17 Виправте помилку й повторіть асемблерування. Повинне з'явитися позначене зеленим маркером повідомлення про завершення асемблерування й відсутність помилок.

2.1.18 Змініть у вихідній програмі значення доданків. Повторіть асемблерування й виконання програми з новими даними. Перегляньте отримані результати.

2.1.19 Змініть у вихідній програмі доданки таким чином, щоб при підсумовуванні в кожному шістнадцятковому розряді виникав перенос.

2.1.20 Повторіть асемблерування й виконання програми з новими даними. Перегляньте новий стан регістра AVR Studio прапорів SREG і поясніть його.

2.1.21 Внесіть у вихідну програму нові дані в десятковому форматі. Повторіть асемблерування й виконання про-

грами з новими даними. Перегляньте отримані результати.

2.1.22 Закрийте **AVR Studio**.

2.1.23 Відкрийте папку зі створеним проектом і подивіться, які файли створив асемблер середовища AVR Studio. Поясніть призначення цих файлів.

Склад проекту відображається також у вікні **Project**.
Результати роботи покажіть викладачеві.

2.2 Зміст звіту

Звіт повинен містити таке:

- назву роботи;
- мету роботи;
- текст програми на асемблері з докладними коментарями;
- аналіз отриманих результатів;
- короткі висновки з роботи.

2.3 Контрольні питання

1 Як створити новий проект у середовищі AVR Studio?

2 Які дії виконуються в наведеній у завданні програмі?

3 Для чого призначені директиви асемблера?

4 Де розміщається в контролері об'єктний код програми?

5 Для чого використовуються регістри загального призначення?

6 Куди розміщається сума після виконання команди **add**?

7 Які способи адресації використовуються в командах програми?

8 Як зробити асемблерування вихідної програми?

9 Як провадиться налагодження програми?

10 Як провести покрокове виконання програми в симуляторі мікроконтролера середовища AVR Studio?

11 Для чого використовується регістр прапорів SREG?

12 Яким чином можна змінити значення даних у регістрах і комірках пам'яті в середовищі AVR Studio?

13 Як одержати лістинг програми?

14 Які файли входять до складу проекту AVR Studio?

3 ЛАБОРАТОРНА РОБОТА 2 РОЗРОБЛЕННЯ Й НАЛАГОДЖЕННЯ ПРОГРАМИ ДОДАВАННЯ ДВОБАЙТНИХ ЧИСЕЛ

Мета роботи: вивчення команд пересилання даних і команд арифметичних операцій мікроконтролера AVR

ATmega 128, принципів програмування мовою асемблера в середовищі AVR Studio, способів виконання й налагодження програм з використанням програмного симулятора.

Завдання: розробити алгоритм і програму мовою асемблера додавання двобайтних чисел **6C9Eh** і **8F79h**, налагодити її й виконати в режимі трасування (покроковому).

Вказівки до складання програми: спочатку необхідно відповідно до варіанта завдання скласти вихідні дані для розроблюваної програми.

Варіанти завдання відрізняються розташуванням операндів: у регістрах загального призначення (РЗП) і пам'яті даних (ОЗП). Варіанти завдання залежно від місця розташування операндів наведені в таблиці 3.1.

Таблиця 3.1 – Варіанти завдання

Варіант	Перший доданок		Другий доданок		Сума	
	РЗП	ОЗП	РЗП	ОЗП	РЗП	ОЗП
1	●			●	●	
2	●			●		●
3	●		●		●	
4	●		●			●
5		●		●	●	
6		●		●		●
7		●	●		●	
8		●	●			●

У вихідних даних необхідно навести варіант завдання і його розшифровку, а також указати обрані місця розташування для кожного байта першого й другого доданків і суми. Приклад подання вихідних даних наведений у таблицях 3.2 і 3.3.

Вихідні дані для варіанта 1

Таблиця 3.2 – Розташування операндів

Варіант	Перший доданок	Другий доданок	Сума
1	РЗП	ОЗП	РЗП

Таблиця 3.3 – Розташування байтів операндів

Перший доданок		**Другий доданок**		**Сума**		**Проміжні регістри для підсумовування**	
Ст. байт	**Мол. байт**	**Ст. байт**	**Мол. байт**	**Ст. байт**	**Мол. байт**	**1-й до-данок**	**2-й До-данок**
R17	R16	0101 h	0100 h	R19	R18	R21	R20

Зауваження. Молодший і старший байти двобайтних чисел необхідно розміщати таким чином, щоб вони знаходилися в сусідніх регістрах або комірках пам'яті, причому молодший байт повинен розташовуватися в регістрі або комірці пам'яті з меншою адресою (номером).

Програма додавання двобайтних чисел повинна бути побудована в такий спосіб. Спочатку розміщають кожний байт доданків у пам'яті або регістрах загального призначення (РЗП) відповідно до варіанта завдання. Потім складають спочатку молодші байти доданків, а потім старші. Причому додавання старших байтів повинне здійснюватися з урахуванням переносу, що може виникнути при додаванні молодших байтів. Після кожного додавання отриману суму пересилають в пам’ять або РЗП відповідно до заданого варіанта. Команди підсумовування можуть складати тільки вміст двох регістрів, при цьому після додавання сума поміщається командою в один з них (у регістр-приймач), і один з доданків буде загублено. Тому для підсумовування використовуються допоміжні (проміжні) регістри.

Програма повинна складатися з таких етапів:

1) завантаження молодшого байта першого доданка в регістр загального призначення (РЗП) або в комірку пам'яті

даних (залежно від варіанта завдання; для наведеного прикладу – у регістр R16);

2) завантаження старшого байта першого доданка в РЗП або в комірку пам'яті даних (для наведеного прикладу – в R17);

3) завантаження молодшого байта другого доданка в РЗП або в комірку пам'яті даних (для наведеного прикладу – у комірку пам'яті даних з адресою 0100h);

4) завантаження старшого байта другого доданка в РЗП або в комірку пам'яті даних (для наведеного прикладу – у комірку пам'яті даних з адресою 0101h);

5) пересилання молодшого байта першого доданка в перший проміжний регістр (R20) для підсумовування;

6) пересилання молодшого байта другого доданка в другий проміжний регістр (R21) для підсумовування;

7) додавання молодших байтів доданків, розташованих у проміжних регістрах (R20, R21), командою додавання без урахування переносу;

8) пересилання суми молодших байтів у РЗП (для наведеного приклада – у регістр R18) або комірку пам'яті даних;

9) пересилання старшого байта першого доданка в перший проміжний регістр (R20) для підсумовування;

10) пересилання старшого байта другого доданка в другий проміжний регістр (R21) для підсумовування;

11) додавання старших байтів доданків, розташованих у проміжних регістрах (R20, R21), командою додавання з урахуванням переносу;

12) пересилання суми старших байтів у РЗП (R19) або комірку пам'яті даних.

3.1 Програма й методика досліджень

3.1.1 Запустіть середовище програмування AVR Studio.

3.1.2 Створіть новий проект.

3.1.3 У вікні редактора наберіть розроблену програму.

3.1.4 Виконайте асемблерування (трансляцію) завантаженої програми (натиснувши клавішу **F7**).

3.1.5 Усуньте указані транслятором помилки (якщо вони є) і повторіть асемблерування.

3.1.6 Перейдіть у режим налагодження (виконання) програм (команда **Start Debugging**).

3.1.7 Зручно розташуйте й при необхідності відкрийте вікна процесора (**Processor)**, регістрів (**Register**), пам'яті (**Memory**) і складу проекту (**Project**).

3.1.8 У вікні **Memory** виберіть пам'ять даних (**Data**).

3.1.9 Виконайте в режимі трасування (покроковому) команди програми (натискаючи клавішу **F11**), переглядаючи у вікнах налагоджувача результати виконання кожної команди (уміст РЗП, пам'яті даних, регістра прапорів, регістрів спеціального призначення процесора, регістрів-покажчиків, часові характеристики програми).

Результати роботи покажіть викладачеві.

Примітка - Порядок роботи в середовищі AVR Studio – створення нового проекту, набір програми, асемблерування, виправлення помилок, виконання команд за допомогою налагоджувача програм і перегляд отриманих результатів – наведений у методичних вказівках до лабораторної роботи 1.

3.2 Зміст звіту

Звіт повинен містити таке:

- назву роботи;
- мету роботи;
- варіант;
- вихідні дані, у яких необхідно вказати місце розташування кожного байта доданків і суми (указати конкретний регістр або адресу комірки пам'яті);
- текст програми на асемблері з докладними коментарями;
- аналіз отриманих результатів;
- короткі висновки з роботи.

3.3 Контрольні питання

1 Поясніть алгоритм роботи програми підсумовування двобайтних чисел.

2 У якому порядку розміщають байти двобайтних чисел у регістрах і комірках пам'яті?

3 Яким чином у програмі враховується перенос, що може виникнути при додаванні молодших байтів чисел?

4 Які способи адресації використовуються в командах програми?

5 Що відбувається із умістом регістра прапорів SREG при виникненні переносу при додаванні?

6 Чому для підсумовування використовуються допоміжні (проміжні) регістри?

7 Де в середовищі AVR Studio вказуються часові характеристики програми і чим вони обумовлюються?

4 ЛАБОРАТОРНА РОБОТА 3 ДОСЛІДЖЕННЯ ЦИКЛІЧНОЇ ПРОГРАМИ ДОДАВАННЯ

Мета роботи: дослідження принципів побудови циклічних програм мовою асемблера в середовищі AVR Studio, застосування директив асемблера, способів виконання й налагодження програм з використанням програмного симулятора.

Завдання: набрати в середовищі програмування AVR Studio наведену в завданні вихідну програму мовою асемблера, зробити асемблерування й виконання в різних режимах.

4.1 Програма й методика досліджень

4.1.1 Створіть папку для файлів проекту.

4.1.2 Запустіть AVR Studio.

4.1.3 Створіть новий проект.

4.1.4 У вікні редактора наберіть наведену нижче програму. Програма реалізує алгоритм n-кратного додавання числа із самим собою (множення числа на n) з використанням циклів.

;Програма n-кратного підсумовування числа із самим собою

```
; Призначення імен змінним (регістрам), константам і
; початкової адреси програми:
.def  Num=r16          ; Лічильник циклів
.def  Slag1=r17        ; Перший доданок
.def  Rezult=r17       ; Результат підсумовування
.def  Slag2=r18        ; Другий доданок
.equ  N=5              ; Число повторень (циклів)
.equ  AdrRez=$100      ; Адреса результату підсумовування
.org 0                 ; Початкова адреса програми
; Текст програми:
        ldi   Num, N       ; Завантаження числа повторень в
                           ; лічильник циклів
```

```
        clr   Slag1          ; Перший доданок установлюємо
                             ; рівним 0
        ldi   Slag2, 3       ; Другий доданок установлюємо
                             ; рівним 3
Sum:    add   Slag1, Slag2   ; Підсумуємо доданки
        dec   Num            ; Зменшуємо вміст лічильника
                             ; циклів на 1
        brne  Sum            ; Повторити підсумовування
                             ; (перехід на початок циклу – мітку
                             ; Sum), якщо вміст лічильника
                             ; циклів не дорівнює нулю
        sts   AdrRez, Rezult ; Точка виходу із циклу, якщо
                             ; вміст лічильника циклів
                             ; дорівнює нулю. Завантаження
                             ; результату N-кратного
                             ; підсумовування в комірку
                             пам'яті
        nop                  ; Немає операції
        nop
```

4.1.5 Пояснення до програми

На початку програми змінним і константам, з якими працює програма, призначаються імена. Це дозволяє замість номерів регістрів, констант і адрес пам'яті використати в командах програми імена, що відображають їхнє призначення в даній програмі. Це забезпечує зручність роботи із програмою й полегшує її розуміння. Для призначення імен регістрам призначена директива асемблера **.def,** константам – директива **.equ**. Початкова адреса пам'яті програм, де повинна буде розташовуватися програма, вказується директивою **.org**.

Алгоритм роботи програми такий:

1) один з регістрів загального призначення (R16) використовується як лічильник циклів програми (Num). У нього записується кількість циклів (N), які необхідно виконати програмі;

2) тому що необхідно підсумовувати число із самим собою N разів, а команда додавання використовує два доданки, один з них (Slag1) приймаємо рівним нулю (очищаємо регістр, де цей доданок розташований). Цей доданок розташований у регістрі-приймачі команди додавання, тому після додавання в ньому опиниться сума. Другий же доданок (Slag2), розташований в регістрі-джерелі, буде залишатися незмінним;

3) заносимо другий доданок у регістр;

4) складаємо доданки. Після першого додавання в регістрі-приймачі (Rezult) опиниться перша сума. Тому що перший доданок дорівнює нулю, перша сума буде дорівнювати першому доданку. Оскільки регістр-приймач (R17) спочатку містить перший доданок, а після додавання – суму, йому присвоєно два імені: Slag1 і Rezult, які використовуються в програмі відповідно до їх призначення;

5) декрементуємо вміст лічильника циклів. Тепер його вміст буде на одиницю менше, тому що одна операція додавання вже виконана;

6) повторюємо додавання другий раз, повернувшись на початок циклу (на виконання команди додавання, позначеної міткою **Sum**), після чого знову декрементуємо вміст лічильника циклів. До першої суми, розташованої на місці першого доданка, додається другий доданок, тому сума подвоїться, а вміст лічильника циклів знову зменшиться на одиницю;

7) повторюємо цей цикл доти, поки вміст лічильника циклів не стане рівним нулю, після чого відбувається вихід із циклу, і отримана сума зберігається в комірці пам'яті даних. Тому що підсумовування провадилося N раз, результат буде дорівнювати N Slag2.

4.1.6 Виконайте асемблерування (трансляцію) завантаженої програми (натиснувши клавішу **F7**).

4.1.7 Усуньте зазначені транслятором помилки (якщо вони є) і повторіть асемблерування.

4.1.8 Перейдіть у режим налагодження (виконання) програм (команда **Start Debugging**).

4.1.9 Зручно розташуйте й при необхідності відкрийте вікна процесора (**Processor)**, регістрів (**Register**) і пам'яті (**Memory**).

4.1.10 У вікні **Memory** виберіть пам'ять даних (**Data**).

4.1.11 Виконайте в режимі трасування (покроковому) команди програми (натискаючи клавішу **F11**), переглядаючи у вікнах налагоджувача результати виконання кожної команди (уміст РЗП, пам'яті даних, регістра прапорів, регістрів спеціального призначення процесора, регістрів-покажчиків, часових характеристик програми). Результати роботи покажіть викладачеві.

Зверніть увагу на значення прапора нуля **Z** регістра прапорів SREG під час виконання циклічної частини програми й при виході із циклу, коли вміст лічильника циклів Num стане рівним нулю.

4.1.12 Розрахуйте кількість машинних циклів, затрачуваних на виконання вихідної програми, і час виконання програми й порівняйте їх з показаннями лічильника циклів і часом виконання (**Cycle Counter** і **Stop Watch** у вікні **Processor**).

Кількість машинних циклів, затрачуваних на виконання вихідної програми, дорівнює сумі машинних циклів, затрачуваних кожною командою програми. Необхідно врахувати, що команди, які входять у циклічну частину програми, виконуються кілька разів. Кількість циклів, за яку виконується кожна команда, наведена в довідкових даних за системою команд мікроконтролера ATmega 128. Довідкові дані щодо

команд, які використовуються у програмі, наведені в додатку А.

Тривалість машинного циклу дорівнює періоду повторення імпульсів, вироблюваних тактовим генератором. Частота тактового генератора (**Frequency**) зазначена у вікні **Processor**.

Час виконання програми дорівнює добутку сумарної кількості машинних циклів, затрачуваних на виконання програми, на тривалість одного машинного циклу.

4.1.13 Повторіть виконання програми для інших значень другого доданка Slag2 і кількості циклів N (див. 4.1.11).

4.1.14 Відновіть початкове значення (FF) комірки пам'яті з адресою AdrRez і встановіть курсор на команду **nop.** Виконайте програму в режимі виконання до курсора, вибравши команду **Run To Cursor** у меню **Debug**, і перегляньте результати її виконання.

Примітка - Порядок роботи в середовищі AVR Studio – створення нового проекту, набір програми, асемблерування, виправлення помилок, виконання команд за допомогою налагоджувча програм і перегляд отриманих результатів – наведений у методичних вказівках до лабораторної роботи 1.

4.2 Зміст звіту

Звіт повинен містити таке:

- назву роботи;
- мету роботи;
- текст програми на асемблері з докладними коментарями;
- розрахунок кількості машинних циклів і часу, затрачуваного на виконання програми;
- аналіз отриманих результатів;
- короткі висновки з роботи.

4.3 Контрольні питання

1 Поясніть алгоритм роботи програми n-кратного підсумовування.

2 Для чого константам і змінним призначаються імена?

3 Яким чином у програмі призначаються імена константам і змінним?

4 Для чого призначені директиви асемблера, як вони впливають на машинний код програми?

5 Що являє собою мітка у вихідній програмі?

6 Що підставляє асемблер замість мітки в об'єктний код програми?

7 Яка умова виходу програми із циклу?

8 Як визначається час виконання програми?

9 Чому дорівнює час виконання команди?

10 Чому дорівнює тривалість одного машинного циклу?

11 Яким чином у середовищі AVR Studio виконати програму не покроково, а за один крок?

5 ЛАБОРАТОРНА РОБОТА 4 СПОСОБИ

ПРОГРАМНОГО ФОРМУВАННЯ ЧАСОВИХ ЗАТРИМОК

Мета роботи: дослідження способів побудови програм часових затримок мовою асемблера в середовищі AVR Studio, застосування директив асемблера, функцій у командах вихідного файлу, складання й використання під-програм, вивчення способів виконання й налагодження про-грам з використанням програмного симулятора.

5.1 Завдання 1 Дослідження циклічної програми формування часової затримки

Набрати в середовищі програмування AVR Studio наведену в завданні вихідну програму мовою асемблера, зробити її асемблерування й виконання. Пояснити отримані результати.

5.2 Програма й методика досліджень завдання 1

5.2.1 Створіть папку для файлів проекту.

5.2.2 Запустіть AVR Studio.

5.2.3 Створіть новий проект.

5.2.4 У вікні редактора наберіть наведену нижче про-граму. Програма реалізує циклічний алгоритм формування часової затримки тривалістю t = 5 мс.

; Програма формування часової затримки тривалістю 5 мс

```
; Призначення імен змінним (регістрам), константам і
; початкової адреси програми:
.include  "m128def.inc"
```

```
.org 0
; Програма:
        ldi   YL, low(5000) ; Завантаження молодшого байта
                            ; лічильника циклів
        ldi   YH, high(5000); Завантаження старшого байта
                            ; лічильника циклів
povtor: sbiw      YL, 1     ; Декремент умісту лічильника
                            ; циклів
        brnepovtor          ; Повернення на початок циклу
                            ; (мітку povtor), якщо вміст
                            ; лічильника циклів не дорівнює
                            ; нулю
        nop                 ; Немає операції
        nop
```

5.2.5 Пояснення програми до завдання 1

На початку програми змінним і константам, з якими працює програма, призначаються імена і вказується початкова адреса програми.

Програма починається з директиви **.include «ім'я файлу»**, що AVR Studio сприймає так, ніби замість неї в цьому місці був вставлений уміст цього файлу. Файл m128def.inc складається з директив призначення визначених імен константам і регістрам, використаних у даному мікроконтролері. Повний шлях до цього файлу указаний асемблера при установленні його параметрів. Асемблер транслює файл, що вставляється, до його закінчення й додає отриманий об'єктний код до вихідної програми. Звичайно, можна використати в програмі замість імені константи або регістра саму константу або адресу регістра, але вдало підібрані імена значно полегшують читання й розуміння програм.

Після асемблерування з'явиться можливість при необхідності переглянути вміст цього файлу. Файл m128def.inc з'явиться у вікні **Project**, у якому відображається склад

проекту, у папці **Included Files**. Подвійним клацанням миші відкрийте файл m128def.inc. На екрані поверх вікна програми з'явиться вікно із умістом файлу, що цікавить нас.

Затримка формується циклічною програмою, у якій багаторазово виконується деяка кількість команд, які не впливають на роботу основної програми, але забирають машинний час. У лічильник циклів завантажується число повторень команд у циклічній частині програми, і потім воно послідовно зменшується на одиницю доти, поки не стане рівним нулю. Чим більшу затримку хочемо одержати, тим більше число потрібно занести в цей лічильник. Для одержання затримки в 5 мс одного 8-розрядного регістра недостатньо, тому що він може забезпечити тільки $2^8 = 256$ повторень, і витримка часу вийде занадто малою. Тому як лічильник використаємо 16-розрядний регістр-покажчик Y. Молодший байт цього регістра має ім'я YL, а старший – YH. Ці імена присвоєні регістру директивою **.def** у приєднаному до програми файлі m128def.inc.

У ролі другого операнда в командах завантаження числа повторень використовуються *визначені в асемблері функції* **low(exp)** і **hight(exp)**, які повертають молодший і старший байт виразу **exp** відповідно (у нашому випадку – молодший і старший байти числа циклів).

Для декрементування вмісту лічильника використаємо команду **sbiw,** що вичитає безпосереднє значення (у нашому випадку 1) з 16-розрядного слова.

Довідкові дані щодо команд, використовуваних у програмі, наведені в додатку А.

Кількість числа повторень визначаємо з наступних міркувань. Тому що витримка часу досить велика й вимагає великої кількості повторень, то часом виконання команд, які не входять у цикл, можна знехтувати. Нехтуємо також і тим, що при виході із циклу команда **brne** займає не два, а один машинний цикли. Уважаємо, що ця команда завжди виконується за два машинних цикли.

Тоді вважаємо, що один цикл програми, який складається з команд **sbiw** і **brne,** виконується за чотири машинні цикли, а загальна кількість машинних циклів $n_{МЦ} \approx 4N$, де N – число повторень циклів програми. Звідси визначимо N.

$$N \approx \frac{n_{МЦ}}{4}. \qquad (5.1)$$

Час виконання одного машинного циклу дорівнює періоду тактової частоти

$$T_{МЦ} = \frac{1}{f}, \qquad (5.2)$$

де f – частота тактового генератора. f = 4 МГц = $4 \cdot 10^6$ Гц.

Тоді

$$n_{МЦ} = \frac{t}{T_{МЦ}} = t \cdot f, \qquad (5.3)$$

де t – заданий час затримки; $t = 5$ мс $= 5 \cdot 10^{-3}$ с.

Підставляючи значення $n_{МЦ}$ у формулу (5.1), одержимо

$$N \approx \frac{t \cdot f}{4} = \frac{5 \cdot 10^{-3} \times 4 \cdot 10^6}{4} = 5000.$$

Таким чином, для забезпечення витримки часу 5 мс необхідно виконати 5000 повторень.

5.2.6 Установіть в програмі невелике число циклів (3 – 5) і виконайте асемблерування (трансляцію) завантаженої програми (натиснувши клавішу **F7**).

5.2.7 Усуньте указані транслятором помилки (якщо вони є) і повторіть асемблерування.

5.2.8 Перейдіть у режим налагодження (виконання) програм (команда **Start Debugging**).

5.2.9 Зручно розташуйте й при необхідності відкрийте вікна **Proceccor, Project** і **Register**.

5.2.10 Виконайте програму в режимі трасування (натискаючи клавішу **F11**) і переконайтеся в правильності її роботи.

5.2.11 Відновіть у вихідній програмі вміст лічильника циклів і повторіть асемблерування.

5.2.12 Перейдіть у режим налагодження (виконання) програм. Відкрийте вікно **Processor.** Установіть одиниці вимірювання часу виконання програми – мілісекунди. Для цього клацніть правою кнопкою миші на рядку **Stop Watch** цього вікна й у списку, що з'явився, виберіть **пункт Stop Watch: show as milliseconds.**

5.2.13 Установіть курсор миші на команду **nop.** Виконайте програму в режимі виконання до курсора, вибравши команду **Run To Cursor** у меню **Debug**, потім перегляньте показання лічильника циклів і час виконання програми (**Cycle Counter** і **Stop Watch** у вікні **Processor**). Порівняйте ці показання з розрахунковими для витримки часу 5 мс.

5.2.14 Розрахуйте число повторень N відповідно до заданого варіанта, уведіть його у вихідну програму й повторіть асемблерування. Варіанти завдання наведені в таблиці 5.1.

Таблиця 5.1 – Варіанти завдання 1

Варіант	1	2	3	4	5	6	7	8	9	10	11	12
Затримка, мс	10	15	20	25	30	35	40	45	50	55	60	65

5.2.15 Виконайте програму в режимі виконання до курсора й перегляньте показання лічильника циклів і час виконання програми. Порівняйте ці показання з розрахунковими.

Результати роботи покажіть викладачеві.

Примітки

1 Тривалість одного машинного циклу дорівнює періоду повторення імпульсів, вироблюваних тактовим генератором (частота тактового генератора **Frequency** наведена у вікні **Processor**).

2 Порядок роботи в середовищі AVR Studio – створення нового проекту, набір програми, асемблерування, виправлення помилок, виконання команд за допомогою налагоджувача програм і перегляд отриманих результатів – наведений у методичних вказівках до лабораторної роботи 1.

5.3 Зміст звіту за завданням 1

Звіт повинен містити таке:

- завдання;
- мету роботи;
- текст програми на асемблері з докладними коментарями;
- розрахунок числа циклів для забезпечення часу затримки відповідно до заданого варіанта;
- аналіз отриманих результатів;
- короткі висновки з роботи.

5.4 Контрольні питання до завдання 1

1 Поясніть алгоритм роботи програми формування часової затримки.

2 Поясніть призначення директиви асемблера **.include.**

3 Поясніть призначення файлу m128def.inc. Як переглянути вміст цього файлу?

4 Який принцип програмного формування часової затримки?

5 Чому як лічильник використовуємо 16-розрядний регістр?

6 6 Де в програмі визначені імена байтів регістра-покажчика YL і YH?

7 Яке призначення функцій асемблера **low(exp)** і **hight(exp)**?

8 Як обчислюється час затримки програми?

5.5 Завдання 2 Розроблення й дослідження програми формування часової затримки із двома циклами

Розробити мовою асемблера, налагодити й дослідити програму часової затримки відповідно до заданого варіанта. Програма повинна складатися з основного й вкладеного циклів. Вкладений цикл являє собою програму часової затримки тривалістю 5 мс, наведену в завданні 1. За допомогою зовнішнього циклу забезпечується загальна часова затримка відповідно до варіанта завдання.

5.6 Вказівки до розроблення програми за завданням 2

У попередньому завданні для формування затримки часу як лічильник циклів (числа повторень) використовувався 16-розрядний регістр Y. Час затримки обмежений максимальним числом повторень, яке можна записати в лічильник, рівним $2^{16}–1 = 65535$. Ідея збільшення часу витримки полягає в тому, щоб повторити виконання циклічної програми затримки кілька разів. Простіше за все це можна зробити, організувавши повторення програми в іншому ци-

клі. Таким чином, для збільшення часу витримки використовуються програми з одним або декількома вкладеними циклами. Блок-схема алгоритму програми з одним вкладеним циклом подана на рисунку 5.1.

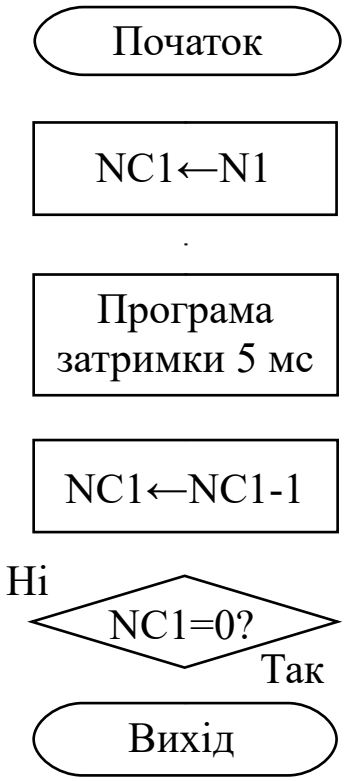

Рисунок 5.1 – Блок-схема алгоритму формування часової затримки з вкладеним циклом

Початок алгоритму включає:

- приєднання файлу m128def.inc;
- присвоєння регістру загального призначення (наприклад, регістру R16) імені лічильника команд зовнішнього циклу NC1;
- призначення імені числу повторень у зовнішньому циклі *N1*;
- установлення початкової адреси програми.

Вихід містить у собі кілька команд **nop**.

Програма функціонує в такий спосіб. Внутрішній (вкладений) цикл формує витримку часу t_2 , а зовнішній цикл повторює цю витримку часу *N1* раз. Тому загальний час затримки дорівнює

$$t = N1 \cdot t_2, \tag{5.4}$$

Загальне число циклів такої програми *N* дорівнює

$$N = N1 \cdot N2, \tag{5.5}$$

де *N2* – кількість повторень програми вкладеного (внутрішнього) циклу.

З огляду на те, що в даному завданні t_2=5 мс, з формули (5.4) визначимо число повторень у програмі зовнішнього циклу

$$N1 = \frac{t}{t_2} = \frac{t}{5}, \tag{5.6}$$

де *t* має розмірність – мілісекунди.

5.7 Програма й методика досліджень завдання 2

5.7.1 Створіть папку для файлів проекту.

5.7.2 Запустіть AVR Studio.

5.7.3 Створіть новий проект.

5.7.4 У вікні редактора наберіть розроблену програму.

5.7.5 Установіть в програмі для зовнішнього й вкладеного циклів невелике число повторень (3 – 5).

5.7.6 Зробіть налагодження програми і її виконання в

режимі трасування.

5.7.7 Розрахуйте число повторень для зовнішнього циклу *N1* відповідно до заданого варіанта, введіть його у вихідну програму, відновіть число повторень для вкладеного циклу. Повторіть асемблерування. Варіанти завдання наведені в таблиці 5.2.

5.7.8 Виконайте програму в режимі виконання до курсора й перегляньте показання лічильника циклів і час виконання програми. Порівняйте ці показання з розрахунковими.

Результати роботи покажіть викладачеві.

Таблиця 5.2 – Варіанти завдання 2

Варіант	1	2	3	4	5	6	7	8	9	10	11	12
Затримка, мс	70	75	80	85	90	95	100	105	110	115	120	125

5.8 Зміст звіту за завданням 2

Звіт повинен містити таке:

- завдання;
- мету роботи;
- блок-схему алгоритму;
- текст програми на асемблері з докладними коментарями;

- розрахунок числа циклів для забезпечення часу затримки відповідно до заданого варіанта;
- аналіз отриманих результатів;
- короткі висновки з роботи.

5.9 Контрольні питання до завдання 2

1 Поясніть алгоритм роботи програми формування часової затримки із вкладеним циклом.

2 Для чого при формуванні часової затримки використовуються вкладені цикли?

3 Чим обмежений час затримки в програмі з одним циклом?

4 Яке співвідношення між кількістю повторень зовнішнього й внутрішнього (вкладеного) циклів?

5 Як визначити час затримки програми із вкладеним циклом?

6 Як визначається загальне число циклів у програмі з одним вкладеним циклом?

7 Як виконати програму в середовищі AVR Studio за один крок?

5.10 Завдання 3 Розроблення й дослідження програми часової затримки із вкладеним циклом, оформленою у вигляді підпрограми

Наберіть у середовищі програмування AVR Studio наведену в завданні вихідну програму мовою асемблера, зробіть її асемблерування, виконання й дослідження.

Програма складається з основного й вкладеного циклів. Вкладений цикл являє собою програму часової затримки тривалістю 5 мс, наведену в завданні 1, оформлену у вигляді *підпрограми*. За допомогою зовнішнього циклу забезпечується загальна часова затримка відповідно до варіанта завдання. Основна програма викликає на виконання підпрограму стільки разів, скільки необхідно для забезпечення заданої затримки часу.

Поясніть отримані результати.

5.11 Програма й методика досліджень завдання 3

5.11.1 Створіть папку для файлів проекту.

5.11.2 Запустіть AVR Studio.

5.11.3 Створіть новий проект.

5.11.4 У вікні редактора наберіть наведену нижче програму формування часової затримки тривалістю *t* = 50 мс.

; Програма формування часової затримки тривалістю 50 мс

```
; Призначення імен змінним (регістрам), константам і початковій адресі
; програми:

.include  "m128def.inc"          ; Включення в програму
                                 ; файлу призначення
                                 ; стандартних імен
                                 ; регістрам і константам
                                 ; мікроконтролера
.def  NC1=r16                    ; Лічильник зовнішніх циклів
.def  tmp=r17                    ; Допоміжний регістр
.equ N1=19                       ; Число повторень
                                 ; зовнішнього циклу
.org 0                           ; Початкова адреса програми

; Текст програми:

        ldi   YL, $bb            ; Завантаження в регістр Y
                                 ; константи AABBh
        ldi   YH, $aa
        ldi   tmp, low(RamEnd)   ; Установлення початкової
                                 ; адреси стека
        out   SPL, tmp
        ldi   temp, high(RamEnd)
```

```
        out  SPH, temp
        ldi  NC1, N1            ; Завантаження числа
                                ; повторень в лічильник
                                ; зовнішніх циклів
povtor1: rcall    zd5ms         ; Виклик підпрограми
                                ; формування часової
                                ; затримки 5 мс (її ім'я -
                                ; мітка zd5ms)
        dec  NC1                ; Декремент умісту
                                ; лічильника зовнішніх циклів
        brne povtor1            ; Повернення на початок
                                ; зовнішнього циклу (мітку
                                ; povtor1), якщо вміст
                                ; лічильника NC1 не
                                ; дорівнює нулю
        nop                     ; Немає операції
        nop
```

; Підпрограма формування часової затримки
; тривалістю 5 мс

```
; Призначення імені константі й початковій адресі
; підпрограми:

.equ N2=5000            ; Число повторень внутрішнього циклу
.org $1E                ; Початкова адреса підпрограми

; Текст підпрограми:

zd5ms:                  ; Ім'я підпрограми (мітка її початку)
        push YL         ; Збереження в стеці вмісту регістра Y
        push YH
        ldi YL, low(N2); Завантаження молодшого байта
                        ; лічильника внутрішніх циклів Y
        ldi YH, high(N2); Завантаження старшого байта
                        ; лічильника внутрішніх циклів Y
povtor2: sbiw YL, 1     ; Декремент умісту лічильника
```

```
                    ; внутрішніх циклів Y
brne povtor2        ; Повернення на початок циклу (мітку
                    ; povtor), якщо вміст лічильника
                    ; внутрішніх циклів Y не дорівнює нулю
pop YH              ; Відновлення вмісту регістра Y,
                    ; збереженого в стеці
pop YL
ret                 ; Вихід з підпрограми й повернення на
                    ; продовження виконання основної
                    ; програми (перехід на команду
                    ; dec NC1)
```

5.11.5 Пояснення до програми за завданням 3

На початку програми змінним і константам, з якими працює програма, призначаються імена і вказується початкова адреса програми.

Константа AABBh поміщається в регістр Y з навчальною метою, щоб простежити її збереження в стеці після звернення до підпрограми й повернення в регістр при виході з неї.

Допоміжний регістр tmp використовується в програмі для завантаження молодшого й старшого байтів адреси початку області стека в регістри молодшого SPL і старшого SPH байтів покажчика стека SP.

Другим операндом у командах завантаження адреси в регістр tmp є визначені в асемблері функції **low(exp)** і **hight(exp)**, які повертають молодший і старший байт виразу **exp** відповідно (у цьому випадку – молодший і старший байти початкової адреси стека). Як початкову адресу стека приймаємо кінець області пам'яті даних, остання адреса якої має стандартне ім'я RamEnd. Це ім'я визначене в приєднаному до програми файлі m128def.inc. У міру заповнення стека адреса його вершини буде зменшуватися. Покажчик стека розташований *в області введення-виведення* пам'яті даних, тому для запису в нього даних використана

команда **out**. Тому що операндами команди **out** можуть бути тільки регістри, то для завантаження адреси в покажчик стека використовується допоміжний регістр tmp.

Вкладений цикл оформлений у вигляді підпрограми. Підпрограма здійснює затримку часу на 5 мс, і її початок указує мітка **zd5ms.** Ця мітка і є ім'ям підпрограми. Для виклику підпрограми використовується команда **rcall,** у якій у ролі операнда вказується ім'я викликуваної підпрограми (мітка). Команда зберігає в стеці адресу повернення в основну програму (у цьому випадку адресу наступної команди – **dec NC1**) і переходить на виконання підпрограми.

Помістити підпрограму можна в будь-яку вільну область пам'яті програм. У цьому випадку обрана початкова адреса підпрограми 1Eh.

Регістр Y використовується в підпрограмі як лічильник циклів, і записана в ньому в основній програмі константа AABBh буде загублена. Тому на початку підпрограми ця константа (вміст регістра Y**)** зберігається в стеці за допомогою команд **push**, що завантажують у стек спочатку молодший байт константи (BB), а потім – старший (AA). Перед виходом з підпрограми вміст регістра Y відновлюється. Константа витягається зі стека й поміщається в регістр Y командами **pop**. Витягати зі стека байти константи необхідно в порядку, зворотному порядку їхнього завантаження, тому що у вершині стека буде перебувати останнє завантажене в нього число – старший байт константи.

Після завершення підпрограми здійснюється вихід в основну програму командою **ret**. Команда **ret** витягає зі стека адресу повернення й завантажує її в лічильник команд. У результаті мікроконтролер переходить на продовження виконання основної програми: на виконання команди **dec NC1**, що іде після команди виклику підпрограми **rcall**.

Час виконання підпрограми (внутрішнього циклу) становить 5 мс. Підпрограма за допомогою програми зовнішнього циклу викликається й виконується ***N1*** раз. Отже, загальний час затримки складе

$$t \approx 5 \cdot N1 \text{ [мс]}. \quad (5.7)$$

Звідси одержимо

$$N1 \approx t/5 = 50/5 = 10. \quad (5.8)$$

Відзначимо, що формула (5.8) є наближеною, тому що в ній не врахований час, затрачуваний командами **rcall** і **ret** на виклик підпрограми й вихід з неї, а також час виконання команд нециклічної частини основної програми (команди **ldi** і **out**). Крім того, уважаємо, що затримка, формована підпрограмою, в точності дорівнює 5 мс, хоча вона також має погрішність (не враховані, наприклад, додані в програму затримки завдання 1 команди **push** і **pop)**. Однак неврахований час є значно меншим заданого значення затримки, тому погрішність буде невелика.

5.11.6 Установіть в програмі для зовнішнього й вкладеного циклів невелике число повторень (3 – 5).

5.11.7 Зробіть налагодження програми і її виконання в режимі трасування. Простежте результат виконання кожної команди програми. Зверніть увагу на виконання команд виклику підпрограми (**rcall**), повернення з підпрограми (**ret**) і на зміну вмісту лічильника команд, покажчика стека (**Stack Pointer** у вікні **Processor**), регістра Y і стека (пам'ять даних **Data** у вікні **Memory**).

5.11.8 За формулою (5.8) розрахуйте число повторень для зовнішнього циклу *N1* відповідно до заданого варіанта, уведіть його у вихідну програму й відновіть число повторень для вкладеного циклу. Установіть одиниці вимірювання часу виконання програми – мілісекунди. Повторіть асемблерування.

Варіанти завдання наведені в таблиці 5.3.

Таблиця 5.3 - Варіанти завдання 3

Варіант	1	2	3	4	5	6
Затримка, мс	130	135	140	145	150	155

Продовження таблиці 5.3

Варіант	7	8	9	10	11	12
Затримка, мс	160	165	170	175	180	185

5.11.9 Виконайте програму в режимі виконання до курсора й перегляньте показання лічильника циклів і час виконання програми. Порівняйте ці показання з розрахунковими й визначте відносну погрішність формування затримки часу.

Погрішність дорівнює

$$\delta = \frac{t_u - t}{t} \cdot 100\%, \qquad (5.9)$$

де t_u – обмірюваний час виконання програми, що з'явився після її виконання в пункті **Stop Watch** вікна **Processor**.

Для одержання необхідної точності при обчисленнях розмірність часу виконання програми у вікні **Processor** повинна бути встановлена в мікросекундах.

Після виконання програми були отримані такі результати: час затримки t_u **=** *50049 мкс*, кількість машинних циклів $n_{мци}$ *= 200196.*

Відносна погрішність завдання витримки часу дорівнює

$$\delta = \frac{50049 - 50000}{50000} \cdot 100 = 0{,}098\%.$$

Результати роботи покажіть викладачеві.

5.12 Зміст звіту за завданням 3

Звіт повинен містити таке:

- завдання;
- мету роботи;
- текст програми на асемблері з докладними коментарями;
- розрахунок числа циклів для забезпечення часу затримки відповідно до заданого варіанта;
- визначення погрішностей формування часу затримки;
- аналіз отриманих результатів;
- короткі висновки з роботи.

5.13 Контрольні питання до завдання 3

1 Поясніть алгоритм роботи програми формування часової затримки із вкладеним циклом у вигляді підпрограми.

2 З якою метою використовується в програмі допоміжний регістр tmp?

3 Навіщо виконується ініціалізація покажчика стека (SP)?

4 На яку комірку стека вказує покажчик стека?

5 Яка адреса пам'яті даних приймається як початкова адреса стека?

6 Як змінюються адреси вершини стека в міру його заповнення?

7 Які дії виконує програма при виклику підпрограми?

8 У якій області пам'яті поміщають підпрограму?

9 З якою метою в програмі використовується константа AABBh і для чого вона зберігається в стеці на початку підпрограми?

10 Як і в якому порядку дані витягають зі стека?

11 Які дії виконуються при виході з підпрограми?
12 Як визначається загальний час затримки?

СПИСОК ЛІТЕРАТУРИ

1 Баранов В.Н. Применение микроконтроллеров AVR: схемы, алгоритмы, программы. – М.: Издательский дом «Додэка-XXI», 2004. – 288 с.

2 Голубцов М.С., Кириченкова А.В. Микроконтроллеры AVR: от простого к сложному. – М.: СОЛОН-Пресс, 2006. – 304 с.

3 Евстифеев А.В. Микроконтроллеры AVR семейств Tiny и Mega фирмы «ATMEL». – М.: Издательский дом «Додэка-XXI», 2004. – 560 с.

4 Загарий Г.И., Мамонов А.В. Управляющие ЭВМ и микропроцессоры. Организация ввода-вывода данных и связи с оператором-технологом: Учеб. пособие. – Харьков: ХИИТ, 1989. – 108 с.

ДОДАТОК А

Команди мікроконтролера ATmega 128, використовувані в лабораторних роботах

Таблиця А.1

Мнемоніка	Операнди	Опис	Операція	Прапори	Цикли

ADD	R_d,R_s	Підсумовування без переносу	$R_d \leftarrow R_d + R_s$	Z,C,N,V,H,S	1
BRNE	k	Перейти, якщо не дорівнює	PC ← PC+k+1, якщо Z=0	–	1 (2) [1)]
CLR	R_d	Очистити регістр	$R_d \leftarrow 0$	Z,C,N,V,S	1
DEC	R_d	Декрементувати значення регістра	$R_d \leftarrow R_d - 1$	Z,N,V,S	1
LDI	R_d,K_8	Завантажити константу	$R_d \leftarrow K_8$	–	1
LDS	R_d,K_{16}	Пряме завантаження	$R_d \leftarrow (k_{16})$		
NOP	–	Ні операції	Немає	–	1
OUT	P,R_s	Запис у порт	$P \leftarrow R_s$	–	1
POP	R_d	Добування регістра зі стека	$R_d \leftarrow$ STACK	–	2
PUSH	R_s	Занесення регістра в стек	STACK ← R_s	–	2
RCALL	k	Відносний виклик підпрограми	STACK ← PC+1, PC ← PC+k+1	–	3
RET	Немає	Повернення з підпрограми	PC ← STACK	–	4
SBIW	Rdl,K_6	Відняти константу зі слова	Rdh:Rdl ← Rdh:Rdl – K_6	Z,C,N,V,S	2
STS	K_{16},R_s	Пряме збереження	$(k_{16}) \leftarrow R_s$	–	2

[1)] Команда займає два цикли, якщо перехід здійснюється (прапор Z = 0).

Примітки

1 У мові асемблера немає різниці в регістрі символів.
2 Операнди можуть бути таких видів:

- k – константа (позначається міткою адреси), або константний вираз;
- K_6 – константа (6 біт), або константний вираз;

- K_8 – константа (8 біт), або константний вираз;
- K_{16} – константа (16 біт), або константний вираз.
- P – константа (5-6 біт), або константний вираз;
- R_d – регістр-приймач у регістровому файлі;
- R_s – регістр-джерело в регістровому файлі;
- Rdh: R29 – для інструкції SBIW і регістра-покажчика Y;
- Rdl: R28 – для інструкції SBIW і регістра-покажчика Y.