Міністерство освіти і науки України
Одеська національна академія зв'язку ім. О.С. Попова
Кафедра комп'ютерно-інтегрованих технологічних процесів та виробництв

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робот №№ 1-5 з дисципліни «ОБЧИСЛЮВАЛЬНА ТЕХНІКА ТА МІКРОПРОЦЕСОРИ»

Одеса 2016

УДК 004.2+004.3+004.4 План НМВ 2016 р.
ББК З973.26-018я73

Рецензент – доц. каф. ІТ Ларін Д.Г.
Укладачі: С.Ю. Манаков, С.П. Главацький, І.С. Кушнір.

Методичні вказівки до лабораторних робот №№ 1-5
з дисципліни «Обчислювальна техніка та мікропроцесори»

Розглядаються основні принципи побудови, функціонування та програмування мікроконтролерів AVR ATmega16 фірми Atmel мовами асемблер та C/C+, робота з портами вводу/виводу та підсистемою переривань.

УХВАЛЕНО
на засіданні кафедри КІТ П і В
та рекомендовано до друку.
Протокол № 4 від 10.11.2016 р.

ЗАТВЕРДЖЕНО
методичною радою
академії зв’язку.
Протокол №6 від 23.02.2016 р.

2

ЗМІСТ

3

Лабораторна робота № 1

ВИВЧЕННЯ АРХІТЕКТУРИ МІКРОКОНТРОЛЕРА ATmega16

1 Мета роботи

Метою роботи є вивчення архітектури МК ATmega16. Набуття базових навичок зі створення проектів, компіляції та налагодження програм в інтегрованому середовищі розробки Atmel Studio.

2 Основні положення

2.1 Загальна характеристика МК сімейства Mega компанії Atmel

Представники сімейства Mega є 8-розрядними МК, призначеними для вбудованих додатків. Вони виготовляються за КМОН-технологією з низьким споживанням енергії, яка в поєднанні з удосконаленою RISC-архітектурою дозволяє досягти найкращого співвідношення швидкодія/енергоспоживання. МК даного сімейства є найбільш розвиненими представниками МК AVR компанії Atmel.

Особливості МК AVR сімейства Mega:

- FLASH-пам'ять програм обсягом 8 ... 128 Кбайт (число циклів стирання/запису не менше 1000);
- оперативна пам'ять (статичне ОЗП) об'ємом 1 ... 4 Кбайт;
- пам'ять даних на основі ЕСППЗП (EEPROM) обсягом 512 байт ... 4 Кбайт (число циклів стирання/запису не менше 100000);
- можливість захисту від читання і модифікації пам'яті програм і даних;
- можливість програмування безпосередньо у системі через послідовні інтерфейси SPI і JTAG;
- можливість самопрограмування;
- можливість внутрішньосхемного налагодження відповідно до стандарту IEEE 1149.1 (JTAG);
- різні способи синхронізації: вбудований RC-генератор з внутрішнім або зовнішнім RC-ланцюжком або з зовнішнім резонатором (п'єзокерамічним або кварцовим); зовнішній сигнал синхронізації;
- наявність декількох режимів зниженого енергоспоживання;
- наявність детектора зниження напруги живлення (brown-out detector, BOD);
- можливість програмного зниження частоти тактового генератора.

Основні характеристики процесора МК сімейства Mega є такі самі, що й МК інших сімейств – Classic і Tiny:

- повністю статична архітектура; мінімальна тактова частота може дорівнювати нулю;
- АЛП підключено безпосередньо до регістрів загального призначення;
- більшість команд виконуються за один машинний цикл;
- багаторівнева система переривань; підтримка черги переривань.

У той самий час процесор МК сімейства Mega має низку характеристик, притаманних саме цьому сімейству:
- найбільше число джерел переривань (до 27 джерел, з яких до 8 зовнішніх);
- наявність програмного стека в усіх моделях сімейства;
- наявність апаратного помножувача.

Усі характеристики підсистеми вводу/виводу МК сімейства Mega такі самі, як і у МК інших сімейств:
- програмне конфігурування і вибір портів вводу/виводу;
- виводи можуть бути запрограмовані як вхідні або як вихідні незалежно один від одного;
- вхідні буфери з тригером Шмітта на всіх висновках;
- можливість підключення до всіх входів внутрішніх резисторів підтягування (опір резисторів становить 35 ... 120 кОм).

МК сімейства Mega мають найбільш багатий набір **периферійних пристроїв** (ПП). При цьому в більшості моделей є всі ПП, які взагалі зустрічаються у складі МК AVR. Цими пристроями є:
- 8-розрядні таймери/лічильники (таймери T0 і T2). У деяких моделей ці таймери/лічильники можуть працювати як годиники реального часу (в асинхронному режимі);
- 16-розрядні таймери/лічильники (таймери T1 і T3);
- сторожовий таймер WDT;
- генератори сигналу з ШІМ розрядністю 8 біт (один із режимів роботи 8-розрядних таймерів/лічильників T0 і T2);
- одно-, дво- і триканальні генератори сигналу з ШІМ регульованою розрядністю (один із режимів роботи 16-розрядних таймерів T1 і T3). Дозвіл ШІМ-сигналу для різних моделей становить 8 ... 10 біт або 1 ... 16 біт;
- аналоговий компаратор;
- багатоканальний 10-розрядний АЦП як з несиметричними, так і з диференціальними входами;
- повнодуплексний універсальний асинхронний приймач (UART);
- повнодуплексний універсальний синхронний/асинхронний приймач (USART);
- послідовний синхронний інтерфейс SPI;
- послідовний двопровідний інтерфейс TWI (аналог інтерфейсу I^2C).

Ядро МК AVR сімейства Mega, як і ядро МК сімейств Classic і Tiny, виконане за удосконаленою RISC-архітектурою (enhanced RISC). Арифметико-логічний пристрій (АЛП), що виконує всі обчислення, є підключений безпосередньо до 32-х робочих регістрів, які об'єднані у регістровий файл. Завдяки цьому АЛП виконує одну операцію (читання вмісту регістрів, виконання операції і запис результату назад в регістровий файл) за один машинний цикл. Практично кожна з команд (за винятком команд, у яких одним із операндів є 16-розрядна адреса) займає одну комірку пам'яті програм.

У МК AVR реалізована Гарвардська архітектура, яка характеризується роздільною пам'яттю програм і даних, кожна з яких має власні шини доступу до них. Така організація дозволяє одночасно працювати як з пам'яттю програм, так

і з пам'яттю даних. Поділ шин доступу дозволяє використовувати для кожного типу пам'яті шини різної розрядності, причому способи адресації і доступу до кожного типу пам'яті також різні.

Ще одним рішенням, спрямованим на підвищення швидкодії, є використання технології конвейєризації. Конвейєризація полягає в тому, що під час виконання поточної команди проводиться вибірка з пам'яті і дешифрування коду наступної команди. Причому, оскільки тривалість машинного циклу МК AVR складає всього один період тактового генератора, вони можуть забезпечувати таку ж продуктивність, що і RISC-МК інших фірм, але за більш низької тактової частоти.

Далі будемо розглядати тільки МК ATmega16, структурна схема якого показана на рис. 2.1. Це типовий представник сімейства Mega.

2.2 Архітектура МК ATmega16

2.2.1 Опис виводів мікросхеми

Мікросхема виготовляється в корпусах формату PDIP і TFQP. Розташування та позначення виводів мікросхеми ATmega16 у PDIP-корпусі показане на рис. 2.2. Опис наведено в табл. 2.1.

Таблиця 2.1 – Опис виводів мікросхеми ATmega16

Позначення	Опис
XTAL1	Вхід тактового генератора
XTAL2	Вихід тактового генератора
RESET RESET	Вхід скидання
AREF	Вхід опорної напруги для АЦП
AGND	Загальний вивід (аналоговий)
AVCC	Вивід джерела живлення АЦП
GND	Загальний вивід
VCC	Вивід джерела струму
PA0 (ADC0) – PA7 (ADC7)	A0 – A7 (Вхід каналу 0–7 АЦП)
PB0 (T0/XCK)	B0 (Вхід зовнішнього тактового сигналу таймера/ лічильника T0 / Вхід/вихід тактового сигналу USART)
PB1 (T1)	B1 (Вхід зовнішнього тактового сигналу таймера/лічильника T1)
PB2 (AIN0/INT2)	B2 (Позитивний вхід компаратора /Зовнішнє переривання)
PB3 (AIN1/OC0)	B3 (Інверсний вхід компаратора / Вихід таймера/лічильника T0 (режими Compare, PWM))
PB4 (SS SS)	B4 (Вибір Slave-пристрою на шині SPI)
PB5 (MOSI)	B5 (Вихід (Master) або вхід (Slave) даних модуля SPI)
PB6 (MISO)	B6 (Вхід (Master) або вихід (Slave) даних модуля SPI)
PB7 (SCK)	B7 (Вихід (Master) або вхід (Slave) тактового сигналу модуля SPI)
PC0 (SCL)	C0 (Тактовий сигнал модуля TWI)

6

PC1 (SDA)	C1 (Лінія даних модуля TWI)
PC2 (TCK)	C2 (Тактовий сигнал JTAG)
PC3 (TMS)	C3 (Вибір режиму JTAG)
C4 (TDO)	C4 (Вихід даних JTAG)
PC5 (TDI)	C5 (Ввід даних JTAG)
PC6 (TOSC1)	C6 (Вихід для підключення резонатора до таймера/лічильника T2)
PC7 (TOSC2)	C7 (Вхід для підключення резонатора до таймера/лічильника T2)
PD0 (RXD)	D0 (Вхід USART)
PD1 (TXD)	D1 (Вихід USART)
PD2 (INT0)	D2 (Вхід зовнішнього переривання)
PD3 (INT1)	D3 (Вхід зовнішнього переривання)
PD4 (OC1B)	D4 (Вихід B таймера/лічильника T1 (режими Compare, PWM))
PD5 (OC1A)	D5 (Вихід A таймера/лічильника T1 (режими Compare, PWM))
PD6 (ICP)	D6 (Вхід захоплення таймера/лічильника T1 (режим Capture))
PD7 (OC2)	D7 (Вихід таймера/лічильника T2 (режими Compare, PWM))

7

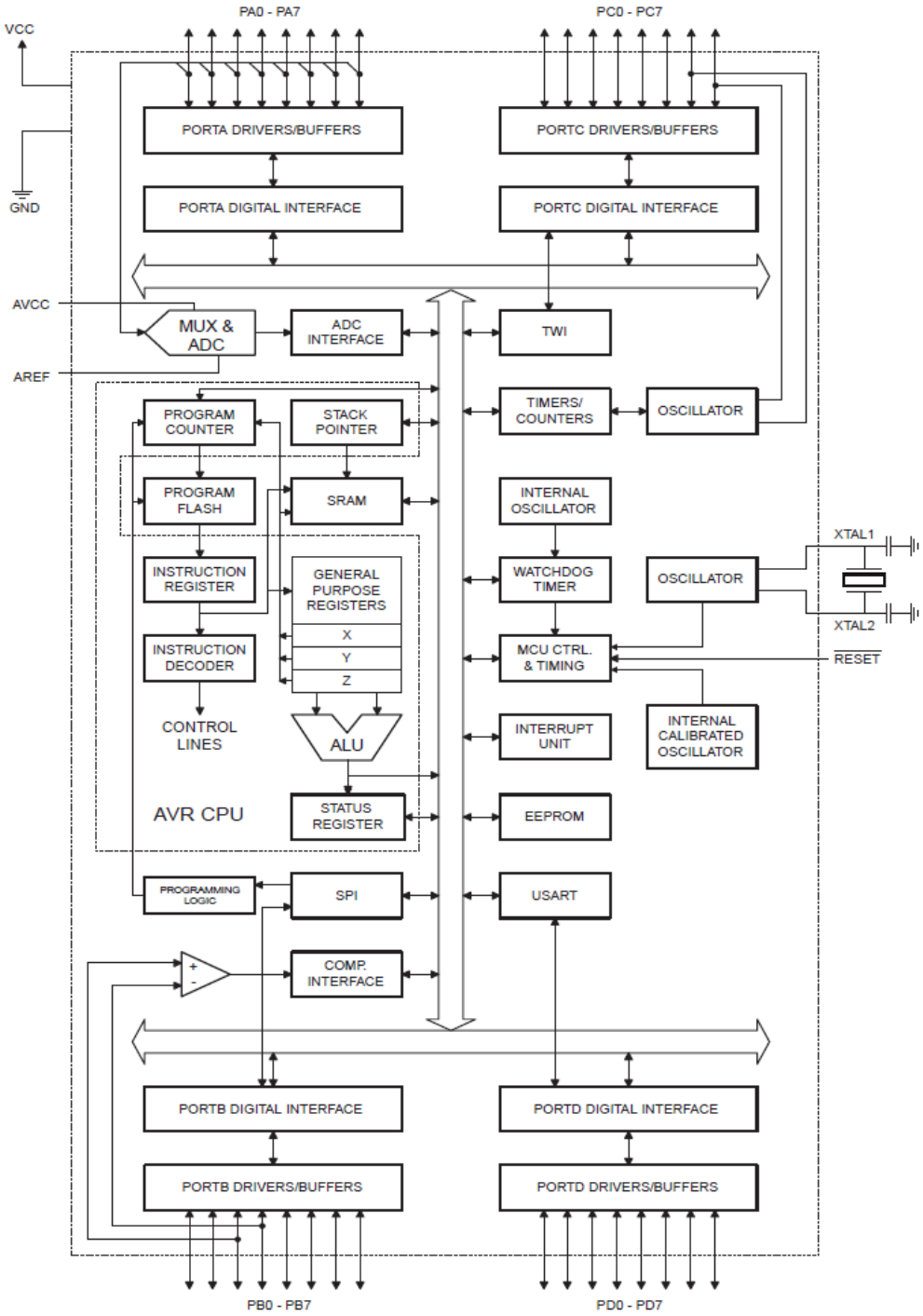


Рисунок 2.1 – Структурна схема МК ATmega16

8

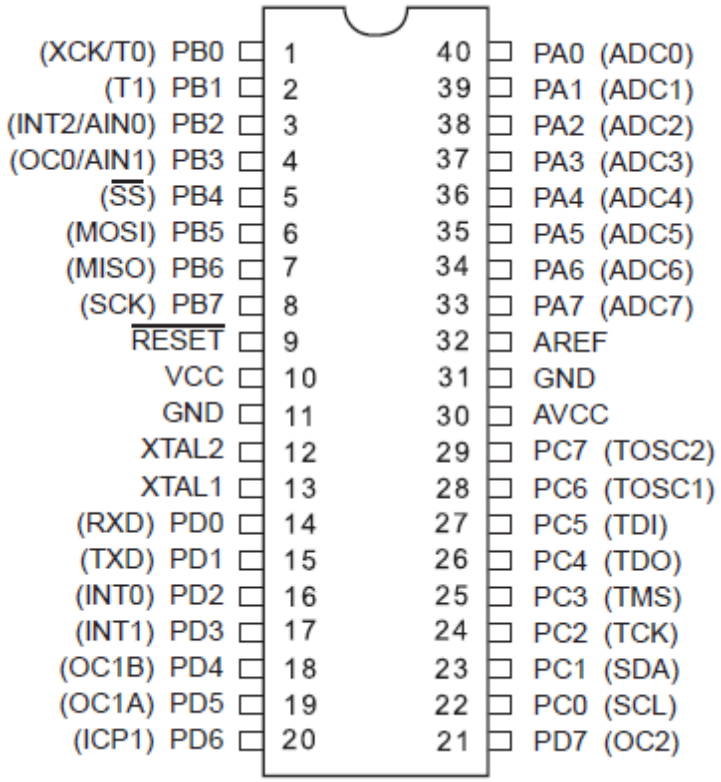


Рисунок 2.2 – Виводи мікросхеми ATmega16 в корпусі PDIP

2.2.2 Організація пам'яті

У МК AVR сімейства Mega реалізована Гарвардська архітектура, відповідно до якої є окремими не тільки адресні простори пам'яті програм і пам'яті даних, але також і шини доступу до них. Способи адресації і доступу до цих областей пам'яті є також різні. Така структура дозволяє центральному процесорові працювати одночасно як з пам'яттю програм, так і з пам'яттю даних, що істотно збільшує продуктивність. Кожна з областей пам'яті даних (ОЗП і EEPROM) також є розташованою у своєму адресному просторі. Карта пам'яті МК ATmega16 показана на рис. 2.3.

Пам'ять програм призначена для зберігання команд, що керують функціонуванням МК. Пам'ять програм також часто використовується для зберігання таблиць констант, які не змінюються під час роботи програми.

Пам'ять програм являє собою електрично стиране ППЗП (FLASH-ПЗП). У зв'язку з тим, що довжина всіх команд є кратною одному слову (16 біт), пам'ять програм має 16-розрядну організацію. Відповідно, обсяг пам'яті МК становить від 8К (8 × 1024) 16-розрядних слів.

9

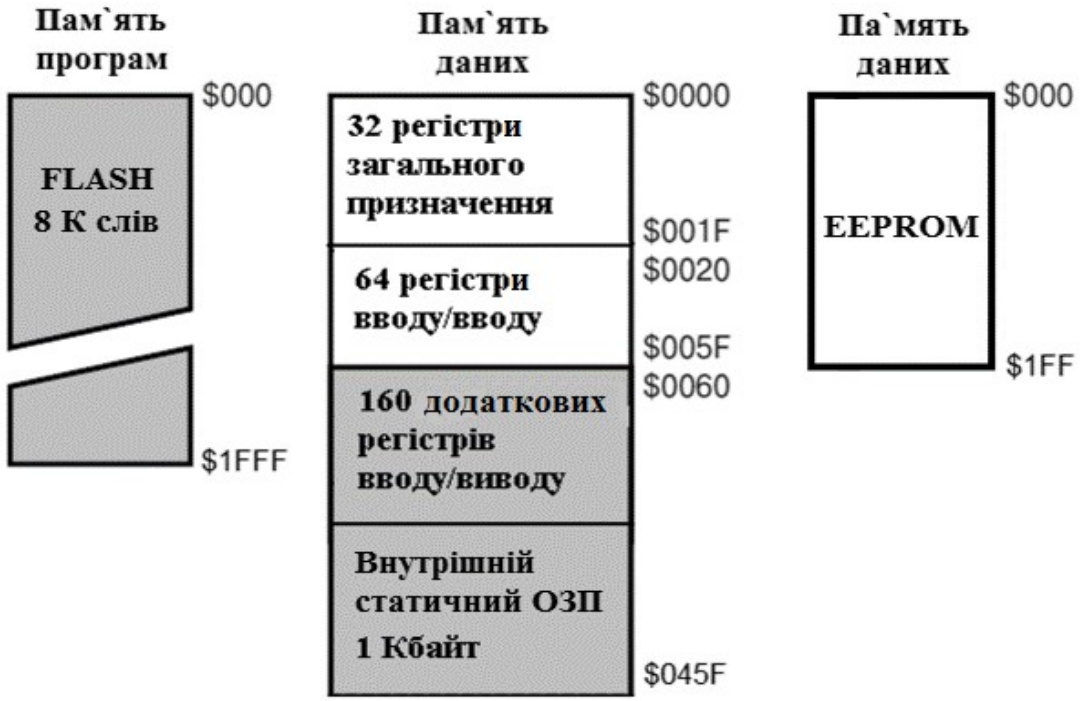


Рисунок 2.3 – Організація пам'яті МК ATmega16

Логічно пам`ять програм є розділеною на дві нерівні частини – область прикладних програм і область завантажувача. В останій може розташовуватися спеціальна програма (завантажувач), що дозволяє МК самостійно керувати завантаженням та вивантаженням прикладних програм. Якщо ж можливість самопрограмування мікроконтролера не використовується, прикладна програма може розташовуватися і в області завантажувача.

Для адресації пам'яті програм використовується регістр PC (Program Counter – лічильник команд). Розмір лічильника команд у ATmega16 становить 13 розрядів, відповідно до об'єму адресованої пам'яті.

За адресою $0000 пам'яті програм знаходиться вектор скидання. Починаючи з адреси $0002 пам'яті програм, розташовується таблиця векторів переривань. Якщо переривання в програмі не використовуються або таблиця векторів переривань розташовується в області завантажувача, то основна програма може починатися безпосередньо з адреси $0002.

FLASH-ПЗП, яке використовується в МК AVR, розраховане як мінімум на 1000 циклів стирання/записи.

Пам'ять даних мікроконтролерів сімейства Mega розподілена на три частини: регістрова пам'ять, оперативна пам'ять (статичне ОЗП) і енергонезалежний ЕСППЗП (EEPROM). Регістрова пам'ять включає 32 регістри загального призначення (РЗП) R0...R31, і службові регістри вводу/виводу (РВВ). Під РВВ у пам'яті мікроконтролера відводиться 64 байта.

Для зберігання змінних програм окрім регістрів загального призначення також може використовуватися статичне ОЗП об'ємом 1 Кбайт. Для довготривалого зберігання різної інформації, яка може змінюватися в процесі функціонування готової системи (калібровочні константи, серійні номери,

ключі і т.п.), в мікроконтролерах сімейства може використовуватися EEPROM-пам'ять. Її обсяг для ATmega16 становить 512 байта. Ця пам'ять розташована в окремому адресному просторі, а доступ до її здійснюється за допомогою певних РВВ.

2.2.3 Регістри загального призначення

Усі РЗП об'єднані в регістровий файл швидкого доступу, структура якого показана на рис. 2.4. Як було зазначено, в мікроконтролерах AVR усі 32 РЗП є безпосередньо доступними АЛП, завдяки чому будь-який РЗП може використовуватися практично в усіх командах і як операнд-джерело і як операнд-приймач. Таке рішення (у поєднанні з конвеєрною обробкою) дозволяє АЛП виконувати одну операцію (вилучення операндів з регістрового файла, виконання команди і запис результату назад до регістрового файла) за один машинний цикл.

R0	$00	
R1	$01	
R2	$02	
…		
R13	$0D	
R14	$0E	
R15	$0F	
R16	$10	
R17	$11	
…		
R26	$1A	регістр X, мол. байт
R27	$1B	регістр X, ст. байт
R28	$1C	регістр Y, мол. байт
R29	$1D	регістр Y, ст. байт
R30	$1E	регістр Z, мол. байт
R31	$1F	регістр Z, ст. байт

Рисунок 2.4 – Структура регистрового файла

Останні 6 регістрів файла (R26...R31) можуть також об'єднуватися в три 16-розрядних регістри X, Y і Z (рис. 2.5), що використовуються як вказівники за непрямої адресації пам'яті даних.

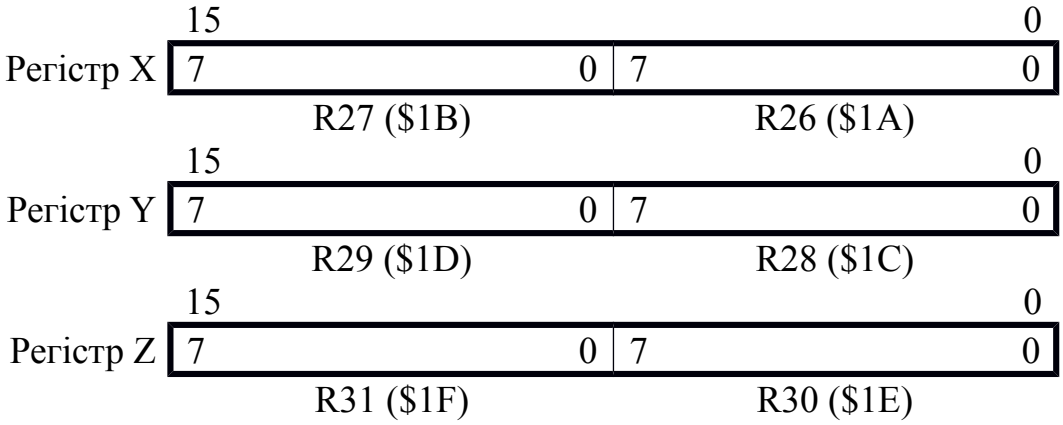


Рисунок 2.5 – Регістри-вказівники X, Y і Z

2.2.4 Регістри вводу/виводу

Усі РВВ умовно можна поділити на дві групи – службові регістри мікроконтролера і регістри, які відносяться до конкретних периферійних пристроїв (у т. ч. регістри портів вводу/виводу). В усіх мікроконтролерах AVR РВВ розташовуються в так званому просторі вводу/виводу розміром 64 байта (табл. 2.2). При назві адрес РВВ у дужках зазначаються відповідні до них адреси комірок ОЗП

Таблиця 2.2 – Регістри вводу/виводу ATmega16

Назва	Адреса	Функція
SREG	$3F ($5F)	Регістр стану (статуса)
SPH	$3E ($5E)	Вказівник стека, старший байт
SPL	$3D ($5D)	Вказівник стека, молодший байт
OCR0	$3C ($5C)	Регістр збіганнятаймера/лічильника T0
GICR	$3B ($5B)	Загальний регістр керування перериваннями
GIFR	$3A ($5A)	Загальний регістр прапорців переривань
TIMSK	$39 ($59)	Регістр маски переривань від таймерів/ лічильників
TIFR	$38 ($58)	Регістр прапорців переривань від таймерів/ лічильників
SPMCR	$37 ($57)	Регістр керування пам'яттю програм
TWCR	$36 ($56)	Регістр керування TWI
MCUCR	$35 ($55)	Регістр керування мікроконтролером
MCUCSR	$34 ($54)	Регістр керування та стану мікроконтролера
TCCR0	$33 ($53)	Регістр керування таймером/лічильником T0
TCNT0	$32 ($52)	Лічильний регістр таймера/лічильника T0
OSCCAL	$31 ($51)	Регістр калібрування тактового генератора
OCDR		Регістр внутрішньосхемного налагодження
SFIOR	$30 ($50)	Регістр спеціальних функцій

12

TCCR1A	$2F ($4F)	Регістр керування A таймера/лічильника T1
TCCR1B	$2E ($4E)	Регістр керування B таймера/лічильника T1
TCNT1H	$2D ($4D)	Лічильний регістр таймера/лічильника T1, ст. байт
TCNT1L	$2C ($4C)	Лічильний регістр таймера/лічильника T1, мол. байт
OCR1AH	$2B ($4B)	Регістр збігу A таймера/лічильника T1, ст. байт
OCR1AL	$2A ($4A)	Регістр збігу A таймера/лічильника T1, мол. байт
OCR1BH	$29 ($49)	Регістр збігу B таймера/лічильника T1, ст. байт
OCR1BL	$28 ($48)	Регістр збігу B таймера/лічильника T1, мол. байт
ICR1H	$27 ($47)	Регістр збігу таймера/лічильника T1, ст. байт
ICR1L	$26 ($46)	Регістр збігу таймера/лічильника T1, мол. байт
TCCR2	$25 ($45)	Лічильний регістр таймера/лічильника T2
TCNT2	$24 ($44)	Регістр збігу таймера/лічильника T2
OCR2	$23 ($43)	Регістр збігу таймера/лічильника T2
ASSR	$22 ($42)	Регістр стану асинхронного режиму
WDTCR	$21 ($41)	Регістр управління сторожовим таймером
UBRRH	$20 ($40)	Регістр швидкості передавання USART, ст. байт
UCSRC		Регістр керування та стану USART
EEARH	$1F ($3F)	Регістр адреси EEPROM, ст. байт
EEARL	$1E ($3E)	Регістр адреси EEPROM, мол. байт
EEDR	$1D ($3D)	Регістр даних EEPROM
EECR	$1C ($3C)	Регістр керування EEPROM
PORTA	$1B ($3B)	Регістр даних порта A
DDRA	$1A ($3A)	Регістр напрямку даних порта A
PINA	$19 ($39)	Виводи порту A
PORTB	$18 ($38)	Регістр даних порту B
DDRB	$17 ($37)	Регістр напрямку даних порту B
PINB	$16 ($36)	Виводи порту B
PORTC	$15 ($35)	Регістр даних порту C
DDRC	$14 ($34)	Регістр напрямку даних порту C
PINC	$13 ($33)	Виводи порту C
PORTD	$12 ($32)	Регістр даних порту D
DDRD	$11 ($31)	Регістр напрямку даних порту D
PIND	$10 ($30)	Виводи порту D
SPDR	$0F ($2F)	Регістр даних SPI
SPSR	$0E ($2E)	Регістр стану SPI
SPCR	$0D ($2D)	Регістр керування SPI
UDR	$0C ($2C)	Регістр даних USART
UCSRA	$0B ($2B)	Регістр керування та стану A USART
UCSRB	$0A ($2A)	Регістр керування та стану B USART
UBRRL	$09 ($29)	Регістр швидкості передавання USART, мол. байт
ACSR	$08 ($28)	Регістр керування та стану аналогового компаратора

ADMUX	$07 ($27)	Регістр керування мультиплексором АЦП
ADCSRA	$06 ($26)	Регістр керування та стану АЦП
ADCH	$05 ($25)	Регістр даних АЦП, ст. байт
ADCL	$04 ($24)	Регістр даних АЦП, мол. байт
TWDR	$03 ($23)	Регістр даних TWI
TWAR	$02 ($22)	Регістр адреси TWI
TWSR	$01 ($21)	Регістр керування TWI
TWBR	$00 ($20)	Регістр швидкості передавання TWI

Серед усіх РВВ є один регістр, що використовується найбільш часто в процесі виконання програм. Цим регістром є регістр стану (або статуса) **SREG**. Він розташовується за адресою $3F ($5F) і містить набір **прапорців** (ознак результату останньої операції АЛП), що показують поточний стан мікроконтролера. Більшість прапорців автоматично встановлюються в «1» або скидються в «0» при настанні певних подій (відповідно до результату виконання команд).

Всі розряди цього регістру доступні як для читання, так і для запису; після скидання мікроконтролера всі розряди регістру скидаються в «0». Формат цього регістру показано на рис. 2.6

	7	6	5	4	3	2	1	0
$3F	I	T	H	S	V	N	Z	C
Читання (R)/Запис (W)	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Початкове значення	0	0	0	0	0	0	0	0

Рисунок 2.6 – Регістр стану SREG

Опис байтів регістру SREG наведено нижче. Прапорець вважається встановленим, якщо він містить значення «1», і скинутим, якщо «0».

I – загальний дозвіл переривань;

T – зберігання біта, що копіюється (для команд BLD, BST);

H – прапорець переносу між тетрадами (молодшою і старшою половинами байта, 4 біти);

S – прапорець знака;

V – прапорець переповнення додаткового коду;

N – прапорець від’ємного значення;

Z – прапорець нуля;

C – прапорець перенесення (перенесення зі старшого розряду результату).

2.3 Робота у середовищі програмування Atmel Studio

Atmel Studio IDE (Integrated Development Environment – інтегроване середовище розробки – професійний програмний продукт від компанії AVR. До версії 5 мав назву AVR Studio. Поширюється безкоштовно. Середовище дозволяє створювати програми для різних моделей 8-розрядних МК AVR

14

мовами Assembler/C/C++, а також здійснювати їхню компіляцію, симуляцію й налагодження. Далі розглядається версія Atmel Studio 7.

Подальший опис роботи у середовищі практично не відрізняється від попередніх версій. Отже, для початку роботи необхідно створити новий проект. Проект складається з одного або декількох файлів з кодом з можливістю підключення бібліотек. Вибираємо пункт меню File > New > Project… . У наступному вікні обираємо мову Assembler і задаємо назву і розташування папки проекту, слідуючи вказівкам викладача. У наступному вікні необхідно вибрати модель МК. В усіх подальших роботах будемо використовувати модель ATmega16. Новий проект має вигляд рис 2.7. Автоматично створюється файл – main.asm з мінімальним шаблоном асемблерного коду для МК, який являє сбою вічний цикл:

```
start:          // коментарі:
    inc r16     // r16=r16+1
    rjmp start  // перехід до мітки start
```

У правії частині проекту слід відзначити вікно Solution Explorer, яке містить список файлів проекту, та де можливо підключати зовнішні бібліотеки функцій, визначень і т.п. Коли текст програми готовий, можна починати збирання проекту (Build), яке складається з двох етапів:

– компіляція (Compiling) – процес перекладу тексту програми, написаної мовою програмування, в об'єктний модуль, який містить машинні команди процесора;

– компонування (Linking) – збирання виконуваного модуля з одного або декількох об'єктних модулів.

Збирання проекту запускається натисканням клавіші F7 або кнопки Build solution на панелі інструментів. Звіт про процес збирання виводиться в нижньому вікні Output. У даному випадку, після успішного збирання з'явиться повідомлення:

```
Build succeeded.
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```


15

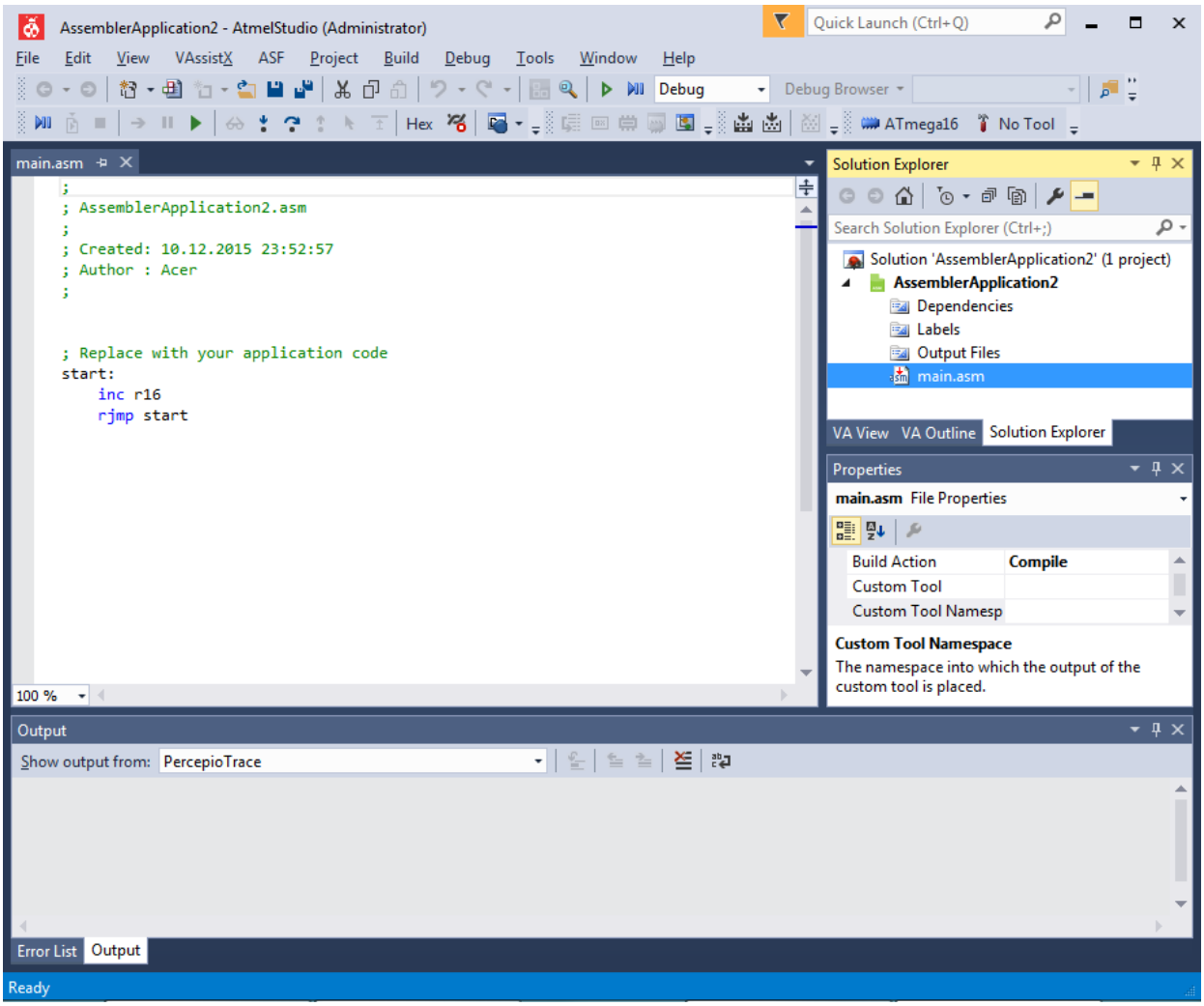


Рисунок 2.7 – Новий проект Atmel Studio 7

У випадку виявлення компілятором помилок у коді будуть видаватися відповідні повідомлення. Результатом збирання є два дубльованих файла з розширеннями .ELF і .HEX, які містять машинний код. Це і буде «прошивка», яку можна записати в МК різними способами.

Далі розглянемо режим симуляції і налагодження. Для перевірки працездатності коду або виправлення помилок (синтаксичних або логічних) існують режими виконання команд по одному рядку (трасування). Вхід у цей режим здійснюється натисканням клавіші F11 (або кнопки Step into на панелі інструментів). При першому запуску, з'явиться запит для вибору інструмента налагодження (tool). В полі Select debugger/programmer слід вибрати Simulator і повернутися до вкладки main.asm. Тепер, при запуску режиму налагодження (F11), вміст головного вікна зміниться. Зліва з'явиться вікно I/O (рис 2.8) зі списком периферійних пристроїв МК і поточним станом усіх регістрів кожного пристрою. Значення регістрів і окремі біти можна змінити вручну в реальному часі. При виборі пункту CPU нижче з'явиться список регістрів процесора.

16

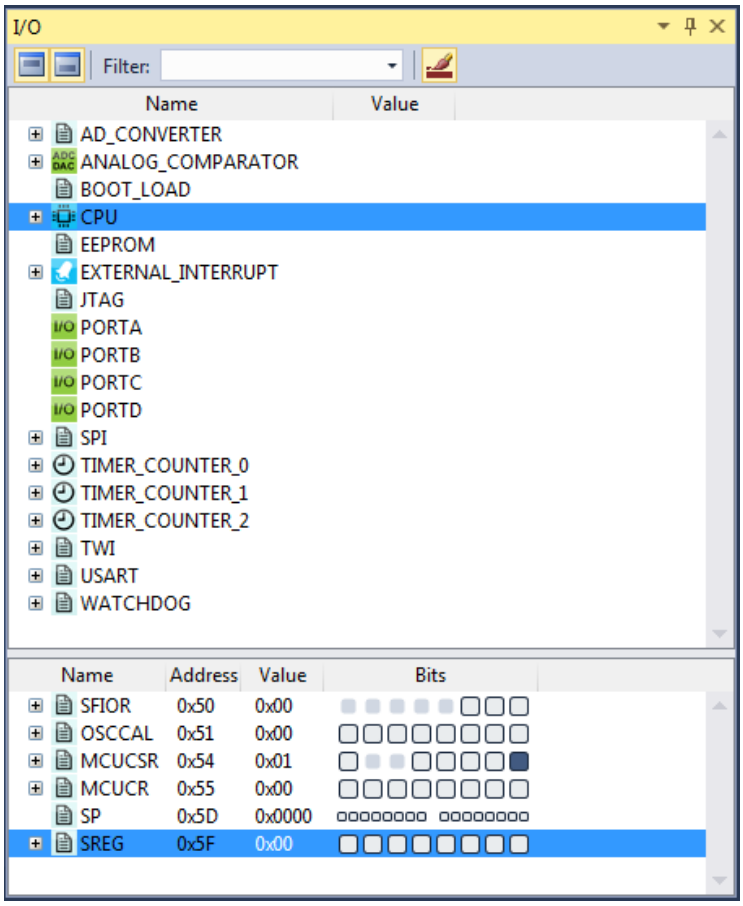


Рисунок 2.8 – Вікно I/O

Відображення вікна з вмістом регістрів R0-R31: меню Debug > > Windows > Registers (Alt + 5). У тому ж меню можна вибрати вікно Processor status (рис. 2.9), де показано поточний стан регістрів процесора, його поточна тактова частота і скільки часу пройшло від запуску МК до останньої точки зупину (Stop watch).

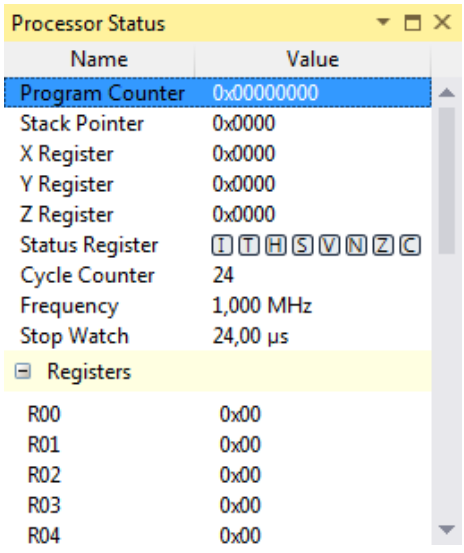


Рисунок 2.9 – Вікно Processor status

17

Також є доступне вікно Memory, де відображаються комірки вибраного типу пам'яті з можливістю зміни їх вмісту вручну.

Вікно дизасемблера (Alt + 8) показує зворотну трансляцію скомпільованої програми з машинних кодів в Асемблер. Таким чином можна оцінити роботу компілятора при написанні програми мовою С.

Далі покрокове виконання кожної команди відбувається при повторному натисканні клавіші F11. Виконання першої команди «inc r16» збільшить значення регістру R16 на одиницю. Перевірити результат її роботи можна у вікнах Registers або Processor status. Наступна команда «rjmp start» здійснить повернення на мітку start. При цьому зміниться значення регістру PC (вікно Processor status).

3 Контрольні питання

1. Назвіть основні характеристики МК AVR.
2. Яке практичне значення достоїнств Гарвардскої RISC-архітектури в МК AVR?
3. Назвіть набір периферійних пристроїв МК сімейства Mega.
4. Як побудована система пам'яті МК сімейства Mega?
5. Опишіть регістри загального призначення.
6. Опишіть регістри ввду/виводу.
7. Опишіть біти регістра стану SREG.
8. Назвіть основні можливості налагодження у середовищі AVR Studio.

4 Домашнє завдання

1. Вивчити розділ 2 (Основні положення).
2. Підготувати протокол лабораторної роботи.
3. Письмово відповісти на контрольні питання.

5 Лабораторне завдання

Для вибору індивідуального варіанта візьміть дві останні цифри залікової книжки: A B.

1. Переведіть числа A, B і AB у двійкову систему і запишіть отримані значення в протокол.
2. Створіть новий проект в Atmel Studio для МК ATmega16, мову програмування – Асемблер.
3. Виведіть на екран вікно Registers.
4. Запустіть режим симуляції і налагодження.
5. Переведіть двійкове число AB в шістнадцятковий вигляд й у вікні Registers внесіть отримане значення у регістр R16.
6. Переведіть від’ємне число –AB у додатковий код і запишіть у регістр R17.
7. Виконайте арифметичне додавання двійкових чисел, A + B. Розставте прапорці H, N, Z, S, V, C.
8. Виконайте арифметичне віднімання двійкових чисел, A – B. Розставте прапорці H, N, Z, S, V, C.

6 Оформлення протоколу

Протокол виконання лабораторної роботи включає: тему, мету роботи, виконання домашнього і лабораторного завдання, висновки.

7 Список рекомендованої літератури

1. Антонов О.С. Обчислювальна техніка та мікропроцесори : Підручник / Антонов О.С, Хіхловська І.В. – Одеса: ОНАЗ ім. О.С. Попова, 2011. – 440 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.
3. Трамперт В. AVR-RISC микроконтроллеры; пер. с нем. / Трамперт В. – К.: «МК-Пресс», 2006. – 464 с.

19

Лабораторна робота № 2

СИСТЕМА КОМАНД АСЕМБЛЕРА МК ATmega16 (Частина 1)

1 Мета роботи

Метою роботи є вивчення системи команд МК ATmega16, способів адресації операндів і роботи зі стеком. Набуття навичок роботи з командами пересилання даних, арифметичних і логічних операцій.

2 Основні положення

2.1 Способи адресації операндів

Способи адресації – це способи завдання операндів у командах (інструкціях). Операнди можна поділити на три типи:

- дані (числа);
- регістри;
- комірки пам'яті (адреси, мітки).

Інструкції асемблера можуть містити до двох операндів. Першим вказується операнд-приймач, а після коми – операнд-джерело. Асемблер не розрізняє регістр символів. Блок коментарів починається зі знака «;» і закінчується в кінці рядка. Коментарі ігноруються компілятором.

Мікроконтролери AVR сімейства Mega підтримують різні способи адресації для доступу до РЗП, РВВ і ОЗП.

Регістрова адресація. Використовується для пересилання даних з одного регістру до іншого. Відповідає командам Асемблера MOV, IN, OUT:

Для пересилання між двома РЗП використовується команда mov:

```
mov r17, r16  ;  r17 = r16
```

Для пересилання з РВВ (порту) у РЗП – команда in:

```
in R16, PORTA ;  r16 = PORTA
```

Для пересилання з РЗП у РВВ (порт) – команда out:

```
out DDRA, r16 ;  DDRA = r16
```

Безпосередня адресація. Численне значення операнда задається в команді та завантажується у вказаний регістр. Команда LDI:

```
ldi r16, 255          ; r16 = 255 (десяткове)
ldi r17, 0b10110111   ; r17 = 10110111 (двійкове)
ldi r18, 0xFF            ; r18 = FF (шістнадцяткове)
ldi r19, $FF          ; r19 = FF (шістнадцяткове)
```

Пряма адресація. Число, що вказане в команді, представляє собою адреса комірки пам'яті ОЗП, вміст якої і є операндом. Команди LDS, STS:

```
lds r16, 70 ; r16 = [70]
sts 40, r17 ; [40] = r17
```

Непряма адресація. У команді вказується найменування спареного регістру-вказівника (X, Y або Z).

Його вміст визначає адресу комірки пам'яті, вміст якої виступає як значення операнда. Має кілька різновидів. Команди LD, ST:

```
ld r16, X ;  r16 = [X]
st Y, r17 ;  [Y] = r17
```

Непряма адресація з постінкрементом. Знак «+» після регістру-вказівника означає, що його вміст збільшується на одиницю після пересилання даних:

```
ld r16, X+ ;  1) r16 = [X]
           ;  2) X = X + 1
```

Непряма адресація з предекрементом. Знак «–» перед регістром-вказівником означає, що його вміст зменшується на одиницю перед пересиланням даних:

```
st –Y, r17 ;  1) Y = Y – 1
           ;  2) [Y] = r17
```

Непряма адресація зі зміщенням. До адреси додається числове зміщення:

```
ldd r16, Y+12 ;  r16 = [Y+12]
st Z+3,r17    ;  [Z+3] = r17
```

Для **читання/запису програмної пам'яті** застосовуються команди LPM/SPM.

2.2 Стек

Стек – це особливим чином організована область ОЗП. Працює за принципом LIFO (англ. Last In, First Out – останнім зайшов, першим вийшов) (рис. 2.1).

Стек розташований в «нижній» частині ОЗП і «зростає» до верху, в бік молодших адрес. Для роботи процесора зі стеком є передбачений вказівник у вигляді регістрів вводу/виводу SPH:SPL. Це вказівник на *вершину стека*. То є байт, наступний за останнім, що був внесений у стек. Спочатку він повинен вказувати на самий останній байт пам'яті (константа RAMEND):

```
ldi r16,LOW(RamEnd)  ;ініціалізація вказівника стека
out SPL,r16          ;в нижній частині ОЗП (RamEnd),
ldi r16,HIGH(RamEnd)     ;якщо       стек       буде
використовуватися
out SPH,r16          ;в програмі
```

Такий блок команд повинен бути виконаний при ініціалізації програми, якщо в ній будуть використовуватися стек, виклики підпрограм або переривання. Однак в деяких моделях які не мають ОЗП, реалізований апаратний стек який не вимагає ініціалізації.

Команди для роботи зі стеком. Запис з регістру у стек:

21

```
push r16 ;[SPH:SPL] = R16
         ;[SPH:SPL] = [SPH:SPL] + 1
```

Читання зі стека в регістр:

```
pop r17 ;R17 = [SPH:SPL]
        ;[SPH:SPL] = [SPH:SPL] – 1
```

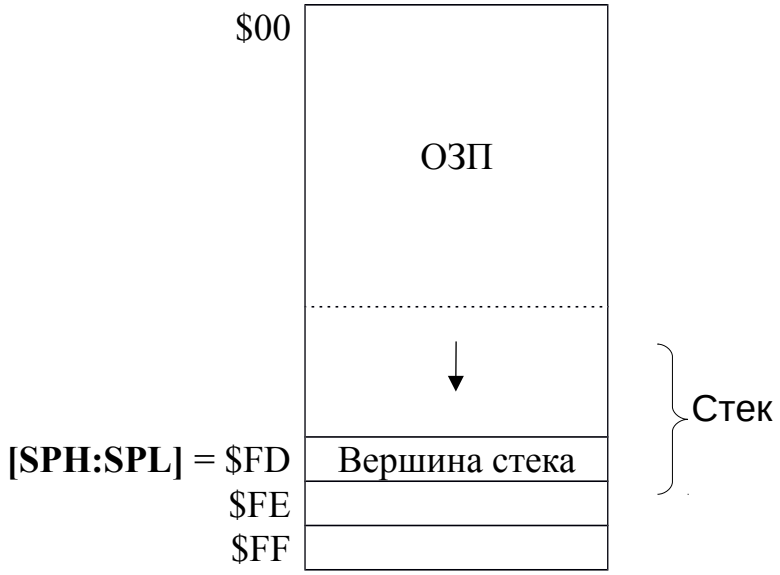


Рисунок 2.1 – Організація стека в ОЗП

Крім того, що стек дозволяє тимчасово зберігати дані, він також може бути використаний для передачі даних у підпрограми та з них. Вміст регістру PC поміщається у стек при виклику підпрограм командами CALL, RCALL, ICALL і повертається командами RET, RETI.

2.3 Команди пересилання даних

Команди пересилання даних надані в табл. 2.1. Візьмемо наступні позначення операндів для всіх подальших довідкових таблиць:
Rd: Результуючий (і вихідний) регістр у регістровому файлі.
Rr: Вихідний регістр у регістровому файлі.
b: Константа (3 біти), може бути константний вираз.
s: Константа (3 біти), може бути константний вираз.
P: Константа (5-6 бітів), може бути константний вираз.
K6; Константа (6 бітів), може бути константний вираз.
K8: Константа (8 бітів), може бути константний вираз.
k: Константа (розмір залежить від інструкції), може бути константний вираз.
q: Константа (6 бітів), 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).

Таблиця 2.1 – Команди пересилання даних

Мнемо-ніка	Опе-ранди	Опис	Операція	Прапорці	Цикли
MOV	Rd,Rr	Скопіювати регістр	Rd = Rr	–	1
MOVW	Rd,Rr	Скопіювати пару регістрів	Rd+1:Rd = = Rr+1:Rr, r,d парні	–	1
LDI	Rd,K8	Завантажити константу	Rd = K	–	1
LDS	Rd,k	Пряме завантаження	Rd = (k)	–	2*
LD	Rd,X	Непряме завантаження	Rd = (X)	–	2*
LD	Rd,X+	Непряме завантаження з постінкрементом	Rd = (X), X=X+1	–	2*
LD	Rd,–X	Непряме завантаження з предекрементом	X=X-1, Rd=(X)	–	2*
LD	Rd,Y	Непряме завантаження	Rd = (Y)	–	2*
LD	Rd,Y+	Непряме завантаження з постінкрементом	Rd = (Y), Y=Y+1	–	2*
LD	Rd,–Y	Непряме завантаження з предекрементом	Y=Y-1, Rd = (Y)	–	2*
LDD	Rd,Y+q	Непряме завантаження зі зміщенням	Rd = (Y+q)	–	2*
LD	Rd,Z	Непряме завантаження	Rd = (Z)	–	2*
LD	Rd,Z+	Непряме завантаження з постінкрементом	Rd = (Z), Z=Z+1	–	2*
LD	Rd,–Z	Непряме завантаження з предекрементом	Z=Z-1, Rd = (Z)	–	2*
LDD	Rd,Z+q	Непряме завантаження зі зміщенням	Rd = (Z+q)	–	2*
STS	k,Rr	Пряме збереження	(k) = Rr	–	2*
ST	X,Rr	Непряме збереження	(X) = Rr	–	2*
ST	X+,Rr	Непряме збереження з постінкрементом	(X) = Rr, X=X+1	–	2*
ST	–X,Rr	Непряме збереження з предекрементом	X=X-1, (X)=Rr	–	2*
ST	Y,Rr	Непряме збереження	(Y) = Rr	–	2*
ST	Y+,Rr	Непряме збереження з постінкрементом	(Y) = Rr, Y=Y+1	–	2
ST	–Y,Rr	Непряме збереження з предекрементом	Y=Y-1, (Y) = Rr	–	2
ST	Y+q,Rr	Непряме збереження зі зміщенням	(Y+q) = Rr	–	2

23

ST	Z,Rr	Непряме збереження	(Z) = Rr	–	2
ST	Z+,Rr	Непряме збереження з постінкрементом	(Z) = Rr, Z=Z+1	–	2
ST	–Z,Rr	Непряме збереження з предекрементом	Z=Z-1, (Z) = Rr	–	2
ST	Z+q,Rr	Непряме збереження зі зміщенням	(Z+q) = Rr	–	2
LPM	Нет	Завантаження з програмної пам’яті	R0 = (Z)	–	3
LPM	Rd,Z	Завантаження з програмної пам’яті	Rd = (Z)	–	3
LPM	Rd,Z+	Завантаження з програмної пам’яті з постінкрементом	Rd = (Z), Z=Z+1	–	3
ELPM	Немає	Розширене завантаження з програмної пам’яті	R0 = = (RAMPZ:Z)	–	3
ELPM	Rd,Z	Розширене завантаження з програмної пам’яті	Rd = = (RAMPZ:Z)	–	3
ELPM	Rd,Z+	Завантаження з програмної пам’яті з постінкрементом	Rd = = (RAMPZ:Z), Z = Z+1	–	3
SPM	Немає	Збереження у програмної пам’яті	(Z) = R1:R0	–	-
ESPM	Немає	Розширене збереження у програмної пам`яті	(RAMPZ:Z) = = R1:R0	–	-
IN	Rd,P	Читання порта	Rd = P	–	1
OUT	P,Rr	Запис у порт	P = Rr	–	1
PUSH	Rr	Занесення регістра у стек	STACK = Rr	–	2
POP	Rd	Витяг регістра зі стеку	Rd = STACK	–	2
* Для операцій доступу до даних кількість циклів зазначено за умови доступу до внутрішньої пам'яті даних, і не є коректною при роботі з зовнішнім ОЗП. Для інструкцій LD, ST, LDD, STD, LDS, STS, PUSH і POP необхідно додати один цикл плюс по одному циклу для кожного очікування.					

2.4 Команди арифметичних і логічних операцій

Команди арифметичних і логічних операцій зведені до табл. 2.2.

24

25

Таблиця 2.2 – Команди арифметичних і логічних операцій

Мнемо-ніка	Опе-ранди	Опис	Операція	Прапорці	Цик-ли
ADD	Rd,Rr	Сума	Rd = Rd + Rr	Z,C,N,V,H,S	1
ADC	Rd,Rr	Сума з перенесенням	Rd = Rd + Rr + C	Z,C,N,V,H,S	1
SUB	Rd,Rr	Віднімання	Rd = Rd – Rr	Z,C,N,V,H,S	1
SUBI	Rd,K8	Віднімання константи	Rd = Rd – K8	Z,C,N,V,H,S	1
SBC	Rd,Rr	Віднімання з перенесенням	Rd = Rd – Rr – C	Z,C,N,V,H,S	1
SBCI	Rd,K8	Віднімання константи з перенесенням	Rd = Rd – K8 – C	Z,C,N,V,H,S	1
AND	Rd,Rr	Логічне I	Rd = Rd · Rr	Z,N,V,S	1
ANDI	Rd,K8	Логічне I з 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	Виключне АБО	Rd = Rd EOR Rr	Z,N,V,S	1
COM	Rd	Побітова інверсія	Rd = $FF – Rd	Z,C,N,V,S	1
NEG	Rd	Зміна знака (додатковий код)	Rd = $00 – Rd	Z,C,N,V,H,S	1
SBR	Rd,K8	Встановити біт (біти) у регістрі	Rd = Rd V K8	Z,C,N,V,S	1
CBR	Rd,K8	Скинути біт (біти) у регістрі	Rd = = Rd · ($FF – K8)	Z,C,N,V,S	1
INC	Rd	Інкрементувати значення регістру	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Декрементувати значення регістру	Rd = Rd – 1	Z,N,V,S	1
CLR	Rd	Очистити регістр	Rd = 0	Z,C,N,V,S	1
SER	Rd	Встановити регістр	Rd = $FF	–	1
ADIW	Rdl,K6	Додати константу до слова	Rdh:Rdl = = Rdh:Rdl + K6	Z,C,N,V,S	2
SBIW	Rdl,K6	Відняти конс-танту зі слова	Rdh:Rdl = = Rdh:Rdl – K 6	Z,C,N,V,S	2
MUL	Rd,Rr	Множення чисел без знака	R1:R0 = Rd * Rr	Z,C	2
MULS	Rd,Rr	Множення чисел зі знаком	R1:R0 = Rd * Rr	Z,C	2

26

MULSU	Rd,Rr	Множення числа зі знаком з числом без знака	R1:R0 = Rd * Rr	Z,C	2
FMUL	Rd,Rr	Множення дробових чисел без знака	R1:R0 = = (Rd * Rr) << 1	Z,C	2
FMULS	Rd,Rr	Множення дробових чисел зі знаком	R1:R0 = = (Rd *Rr) << 1	Z,C	2
FMULSU	Rd,Rr	Множення дробового числа зі знаком з числом без знака	R1:R0 = = (Rd * Rr) << 1	Z,C	2

2.5 Приклади програм

Кожна програма для МК обов'язково містить «вічний» основний цикл, в якому може перебувати все тіло програми. Перед основним циклом, як правило, знаходиться область ініціалізації. У загальному вигляді, шаблон нової програми має наступний вигляд:

```
; ініціалізація
; підключення бібліотек, оголошення змінних, функцій і т.ін.

main:            ; мітка main

    ; ...        (основний цикл)

    rjmp main        ; повернення на мітку main
```

Приклад 1. Завантажити регістри R16 і R17 числами 14 і 21 відповідно. Реалізувати їх арифметичне додавання. Результат записати в комірку пам'яті за адресою 40. Потім, з використанням стека, відновити вихідні значення регістрів.

```
; Ініціалізація стека:
ldi r16,Low(RAMEND)      ; Обов'язкова частина коду
    out SPL,r16          ; при роботі зі стеком
    ldi r16,High(RAMEND) ; або перериваннями
    out SPH,r16          ;
; Ініціалізація змінних:
    ldi r16, 21          ; r16 = 14
    ldi r17, 11          ; r17 = 21
; Основний цикл:
```

```
main:                    ; мітка main
    push r16             ; зберегти r16 у стек
    add r16,r17          ; r16 = r16 + r17
    sts 40,r16           ; [40] = r16
    pop r16              ; повернути зі стека в r16
    rjmp main               ; повернення на мітку main
```

Приклад 2. В ОЗП за адресою 50 зберігається один байт. Задача: інвертувати його розряди використовуючи логічну операцію Виключне АБО і зберегти отримане значення.

```
main:            ; мітка main
lds r16, 50      ; завантаження значення з пам'яті
    ldi r17, 255  ; формування двійкової маски: 11111111
    eor r16, r17  ; інверсія розрядів
    sts 50, r16   ; збереження результату в пам'яті
    rjmp main        ; повернення на мітку main
```

3 Контрольні питання

1. Позначте основні типи операндів команд Асемблера.
2. Які способи адресації підтримують МК AVR?
3. Як організується стек? Команди для роботи зі стеком.
4. Яка команда служить для завантаження числа в РЗП?
5. Яка команда зберігає вміст регістру в комірці ОЗП?
6. Які команди служать для обміну даними через порти?
7. Команди арифметичних операцій.
8. Команди логічних операцій.

4 Домашнє завдання

1. Вивчити розділ 2 (Основні положення).
2. Підготувати протокол лабораторної роботи.
3. Письмово відповісти на контрольні питання.

5 Лабораторне завдання

Для вибору індивідуального варіанту візьміть дві останні цифри залікової книжки: A B.

Складіть програму на Асемблері AVR, яка реалізує наступний алгоритм:

1) Арифметична операція з табл. 2.3 над числами A і B.
2) Запис результату у стек.
3) Логічна операція з табл. 2.4 над числами A і B.
4) Запис результату в ОЗП за адресою AB.

Для отриманої програми створіть проект в AVR Studio і виконайте кожний рядок у режимі налагодження (трасування). У протоколі, запишіть текст програми з коментарями до кожного рядка. У коментарях до команд арифметичних і логічних операцій запишіть відповідні значення прапорців Z, S, C.

Таблиця 2.3 – Варіанти арифметичних операцій

A	0	1	2	3	4	5	6	7	8	9
Операція	+	–	×	+	–	×	+	–	×	+

Таблиця 2.4 – Варіанти логічних операцій

B	0	1	2	3	4	5	6	7	8	9
Операція	and	or	eor	and	or	eor	and	or	eor	and

6 Оформлення протоколу

Протокол виконання лабораторної роботи включає: тему, мету роботи, виконання домашнього і лабораторного завдання, висновки.

7 Список рекомендованої літератури

1. 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.
2. Справка по Ассемблеру AVR. – http://musmu.narod.ru/atmel/avrasm_ rus.html

29

Лабораторна робота № 3

СИСТЕМА КОМАНД АСЕМБЛЕРА МК ATmega16 (Частина 2)

1 Мета роботи

Метою роботи є вивчення системи команд МК ATmega16. Набуття навичок роботи з бітовими операціями і командами розгалуження.

2 Основні положення

2.1 Бітові операції

Асемблер МК AVR має набір команд для операцій над бітами (табл. 2.1).

Логічний зсув. Зсув, за якого біт, що минає, зникає, не впливаючи на решту бітів, а на місці біта, що з'явився, записується біт 0. Біт, що минає, зберігається у прапорці перенесення C. Команди LSL, LSR. Приклад роботи операції логічного зсуву:

Маємо двійкове число, наприклад, 10101010.

При зсуві вліво на 1 біт отримаємо 01010100.

При зсуві вихідного числа вправо на 1 біт отримаємо число 01010101.

Арифметичний зсув. При цьому зсуві слово розглядається не просто як група бітів, а як ціле число в додатковому коді. При *зсуві вліво* поводиться як логічний зсув. Тому в наборі команд є тільки одна команда – арифметичний зсув вправо – ASR. При *зсуві вправо* біт, що минає, зникає, не впливаючи на решту бітів, а на місці біта, що з'явився, встановлюється біт, ой відповідає знаку числа.

Приклад роботи операції арифметичного зсуву:

Розглянемо бінарне число 11111010 = –6.

Якщо зробити зсув вліво на 1 біт, то отримаємо число 11110100 = –12. Якщо зробити зсув вихідного числа вправо на 1 біт, то отримаємо число 11111101 = –3. Легко помітити, що арифметичний зсув вліво відповідає множенню на 2, а зсув вправо – діленню на 2 з округленням.

Циклічний зсув через прапорець перенесення C. При циклічному зсуві біт, що минає, з'являється на іншому боці числа. МК AVR підтримує команди ROL і ROR. Дані операції виконують циклічний зсув над (n + 1)-бітним числом, що складається з регістру і прапорця перенесення C.

Наприклад, якщо в регістрі є число 11111010 прапорець переносу дорівнює 0. Після зсуву вліво на 1 біт в регістрі 11110101 прапорець переносу дорівнює 1. Після зсуву вправо на 1 біт в регістрі 01111101, прапорець переносу дорівнює 0.

Також є група команд для роботи з окремими бітами портів і прапорцями.

У деяких випадках, замість описаних команд, для модифікації бітів застосовують **логічні операції за маскою**. Наприклад, щоб встановити в «1» 1-й біт у числі 00000000 (другий праворуч), можна застосувати до числа операцію

логічного складання з маскою 00000010. Щоб встановити цей біт знову в «0», досить застосувати логічне множення з маскою 11111101. Відповідно, для установки одиниць у розрядах застосовується роз'єднання з маскою. Для інверсії окремих бітів може застосовуватися операція «Виключне АБО» з одиницями у відповідних розрядах маски.

Змінювати окремі біти в регістрах вводу/виводу і портах дозволяють команди SBI, CBI.

Таблиця 2.1 – Зсувні і бітові інструкції

Мнемо-ніка	Опе-ранди	Опис	Операція	Пра-пори	Цик-ли
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	Логічний зсув вправо	Rd(n) = Rd(n + 1), Rd(7) = 0, C = Rd(0)	Z,C,N,V, S	1
ROL	Rd	Циклічний зсув вліво через C	Rd(0) = C, Rd(n + 1) = Rd(n), C = Rd(7)	Z,C,N,V, H,S	1
ROR	Rd	Циклічний зсув вправо через C	Rd(7) = C, Rd(n) = Rd(n + 1), C = Rd(0)	Z,C,N,V, S	1
ASR	Rd	Арифметичний зсув 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)	–	1
BSET	s	Встановлення прапора	SREG(s) = 1	SREG(s)	1
BCLR	s	Очистка прапора	SREG(s) = 0	SREG(s)	1
SBI	P,b	Встановити біт в порту	I/O(P,b) = 1	–	2
CBI	P,b	Очистити біт в порту	I/O(P,b) = 0	–	2
BST	Rr,b	Зберегти біт з регістру в T	T = Rr(b)	T	1
BLD	Rd,b	Завантажити біт з T в регістр	Rd(b) = T	–	1
SEC	Немає	Встановити прапор переносу	C =1	C	1
CLC	Немає	Очистити флаг переносу	C = 0	C	1
SEN	Немає	Встановити прапор від’ємного числа	N = 1	N	1
CLN	Немає	Очистити прапор від’ємного числа	N = 0	N	1

31

SEZ	Немає	Встановити прапор нуля	Z = 1	Z	1
CLZ	Немає	Очистити прапор нуля	Z = 0	Z	1
SEI	Немає	Встановити прапор переривань	I = 1	I	1
CLI	Немає	Очистити прапор переривань	I = 0	I	1
SES	Немає	Встановити прапор числа зі знаком	S = 1	S	1
CLN	Немає	Очистити прапор числа зі знаком	S = 0	S	1
SEV	Немає	Встановити прапор переповнення	V = 1	V	1
CLV	Немає	Очистити прапор переповнення	V = 0	V	1
SET	Немає	Встановити прапор T	T = 1	T	1
CLT	Немає	Очистити прапор T	T = 0	T	1
SEH	Немає	Встановити прапор внутрішнього перенесення	H = 1	H	1
CLH	Немає	Очистити прапор внутрішнього перенесення	H = 0	H	1

2.2 **Команди розгалуження. Безумовні й умовні переходи**

Команди розгалуження модифікують адресу в регістрі PC (адреса наступної команди, що буде виконуватися) для передачі керування іншій команді. Розрізняють безумовні й умовні переходи (табл. 2.2). При виконанні команд умовних переходів, розгалуження здійснюється тільки при виконанні відповідної умови. Кожна умова відповідає стану одного або декількох прапорців з регістру стану SREG.

Слід зазначити, що у середовищі AVR Studio для безумовного переходу зазвичай використовують команду відносного переходу RJMP, тому що вона виконується за 2 цикли, а не за 3, як JMP. При цьому, незважаючи на те, що RJMP реалізує перехід відносно поточного значення PC, для завдання адреси переходу можна вказувати мітку, а не зсув (розрахунки зміщення бере на себе компілятор). Однак розмір зміщення не може перевищувати 2048 байт. Те саме стосується команди виклику підпрограм RCALL.

Команда порівняння CP і її різновиди (CPC, CPI) здійснює звичайне віднімання одного операнда з іншого. Однак, на відміну від команд арифметичного віднімання, результатом команд порівняння є тільки прапорці в

регістрі SREG (ознаки результату). Команда CPSE здійснює порівняння двох регістрів і, в разі рівності їх вмісту, відбувається пропуск наступної команди. Команда TST робить логічне множення єдиного операнда самого на себе. На відміну від команди AND, TST лише змінює ознаки результату. Після порівняння двох чисел у програмі, зазвичай, слідують команди умовних переходів SBcc або BRcc, де cc – код умови. Умовні переходи виконуються тільки при задоволенні відповідної умови (значення одного або декількох прапорців).

Таблиця 2.2 – Команди розгалуження і порівняння

Мнемо-ніка	Опе-ранди	Опис	Операція	Прапори	Ци кли
БЕЗУМОВНІ ПЕРЕХОДИ					
RJMP	k	Відносний перехід	PC = PC + k +1	–	2
IJMP	Немає	Непрямий перехід на (Z)	PC = Z	–	2
EIJMP	Немає	Розширений непрямий перехід на (Z)	STACK = PC+1, PC(15:0) = Z, PC(21:16) = = EIND	–	2
JMP	K	Перехід	PC = k	–	3
RCALL	K	Відносний виклик підпрограми	STACK = PC+1, PC = PC + k + 1	–	3/4
ICALL	Немає	Непрямий виклик (Z)	STACK = PC+1, PC = Z	–	3/4
EICALL	Немає	Розширений непрямий виклик (Z)	STACK = PC+1, PC(15:0) = Z, PC(21:16) = = EIND	–	4
CALL	k	Виклик підпрограм	STACK = PC+2, PC = k	–	4/5
RET	Немає	Повернення з підпрограми	PC = STACK	–	4/5
RETI	Немає	Повернення з переривання	PC = STACK	I	4/5
ПОРІВНЯННЯ					
CPSE	Rd,Rr	Порівняти, пропустити, якщо рівні	if (Rd= =Rr) PC = = PC + 2 … 3	–	1/2/ 3
CP	Rd,Rr	Порівняти	Rd – Rr	Z,C,N,V,H,S	1
CPC	Rd,Rr	Порівняти з перенесенням	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

33

TST	Rd	Перевірка на нуль або від’ємність	Rd = Rd · Rd	Z,C,N,V,S	1
УМОВНІ ПЕРЕХОДИ					
SBRC	Rr,b	Пропустити, якщо біт в регістрі очищений	if(Rr(b) == 0) PC = PC + 2 … 3	–	1/2/3
SBRS	Rr,b	Пропустити, якщо біт в регістрі встановлений	if(Rr(b) ==1) PC = PC + 2 … 3	–	1/2/3
SBIC	P,b	Пропустити, якщо біт в порту очищений	if(I/O(P,b)= =0) PC = PC + 2 … 3	–	1/2/3
SBIS	P,b	Пропустити, якщо біт в порту встановлений	if(I/O(P,b) = = 1) PC = PC + 2 … 3	–	1/2/3
BRBC	s,k	Перейти, якщо прапор в SREG очищений	if(SREG(s) = = 0) PC = PC + k + 1	–	1/2
BRBS	s,k	Перейти, якщо прапор в SREG встановлений	if(SREG(s) = = 1) PC = PC + k + 1	–	1/2
BREQ	k	Перейти, якщо дорівнює	if(Z == 1) PC = PC + k + 1	–	1/2
BRNE	k	Перейти, якщо не дорівнює	if(Z == 0) PC = PC + k + 1	–	1/2
BRCS	k	Перейти, якщо прапор перенесення встановлений	if(C == 1) PC = PC + k + 1	–	1/2
BRCC	k	Перейти, якщо прапор перенесення очищений	if(C == 0) PC = PC + k + 1	–	1/2
BRSH	k	Перейти, якщо дорівнює або більше	if(C == 0) PC = PC + k + 1	–	1/2
BRLO	k	Перейти, якщо менше	if(C == 1) PC = PC + k + 1	–	1/2
BRMI	k	Перейти, якщо мінус	if(N == 1) PC = PC + k + 1	–	1/2
BRPL	k	Перейти, якщо плюс	if(N == 0) PC = PC + k + 1	–	1/2
BRGE	k	Перейти, якщо більше або дорівнює (зі знаком)	if(S == 0) PC = PC + k + 1	–	1/2
BRLT	k	Перейти, якщо менше (зі знаком)	if(S == 1) PC = PC + k + 1	–	1/2
BRHS	k	Перейти, якщо прапор внутрішнього перенесення встановлено	if(H == 1) PC = PC + k + 1	–	1/2

34

BRHC	k	Перейти, якщо прапор внутрішнього перенесення очищено	if(H == 0) PC = PC + k + 1	–	1/2
BRTS	k	Перейти, якщо прапор T встановлено	if(T == 1) PC = PC + k + 1	–	1/2
BRTC	k	Перейти, якщо прапор T очищено	if(T == 0) PC = PC + k + 1	–	1/2
BRVS	k	Перейти, якщо прапор переповнення встановлено	if(V == 1) PC = PC + k + 1	–	1/2
BRVC	k	Перейти, якщо прапор переповнення очищено	if(V == 0) PC = PC + k + 1	–	1/2
BRIE	k	Перейти, якщо переривання дозволені	if(I == 1) PC = PC + k + 1	–	1/2
BRID	k	Перейти, якщо переривання заборонені	if(I == 0) PC = PC + k + 1	–	1/2

2.3 **Цикли**

Програми, в яких багаторазово виконуються ті ж самі ділянки коду, як правило, з різними значеннями вхідних даних, називаються *циклічними*, а послідовність команд, які повторюються – *тілом циклу.* Цикли зі свідомо заданою кількістю повторень називаються *арифметичними*. Цикли, які не мають заздалегідь заданої кількості повторень, називаються *ітераційними*. Вихід із них відбувається при виконанні заданої умови. Для реалізації арифметичних циклів потрібно організувати лічильник циклів. Якщо лічильник спадний, то при виконанні кожного циклу з його вмісту віднімається одиниця. Цикли повторюються до тих пір, поки вміст лічильника є більший нуля. Циклічна програма вважається завершеною, якщо лічильник дорівнює нулю. Рідше використовуються зростаючі лічильники. При роботі з масивами закінчувати цикли можна також при досягненні адреси його останнього елемента. Для організації лічильника циклів зазвичай використовується один із регістрів R16-R24.

2.4 Приклади програм

Приклад 1. Одновимірний масив M(I), N = $10 однобайтових елементів зі знаком розташований у сегменті даних, починаючи з адреси $20. Розсортувати елементи масиву на масив додатніх M_pos та масив від’ємних M_neg і запам'ятати додатні елементи в області пам'яті, починаючи з ефективного

35

адреси $30, а від’ємні елементи в області пам'яті, починаючи з адреси $40. Структурна схема алгоритму показана на рис 2.1.

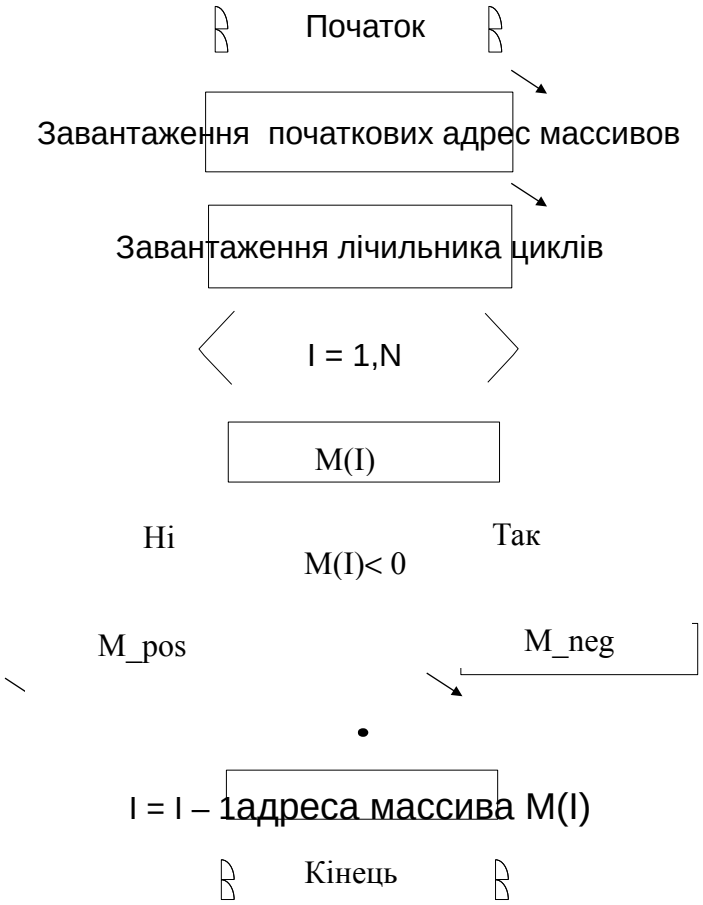


Рисунок 2.1 – Структурна схема алгоритму

Текст програми:

```
Start:
clr r31; Завантаження регістру Z числом $ 20
ldi r30, $ 20; (Початкова адреса вихідного масиву MI).
clr r27; Завантаження адреси масиву MP = $ 30
ldi r26, $ 30; в регістр X.
clr r29; Завантаження адреси масиву MNEG = $ 40
ldi r28, $ 40; в регістр Y.
ldi r19, $ 10; r19 = $ 10 (розмір вихідного масиву
; і лічильник циклу).
Cycle:
ld r16, z +; Зчитування поточного елементу M (I) в r16.
tst r16; установка прапорів
```


36

```
brmi Minus; Якщо елемент від’ємний - перехід на Minus.
st x +, r16; Збереження позитивного елемента.
rjmp Loop; Перехід на мітку Loop.
Minus:
st y +, r16; Збереження негативного елементу.
Loop:
dec r19; Зменшення лічильника циклу.
tst r19; Перевірка лічильника циклу.
brne Cycle; Якщо не нуль - наступний виток циклу.
rjmp Start; Основний цикл.
```

Приклад 2. В оперативній пам'яті розташовано довільний масив з 16 байтів без знака, починаючи з адреси $70. Підрахувати кількість парних елементів масиву за допомогою окремої процедури із записом результату в регістр R17.

Текст програми:

```
Start:
ldi r18,16
clr r27
ldi r26, $ 70
clr r17
rcall NumEven
rjmp Start
NumEven:
ld r16, X +
ldi r20,0b00000001
and r16, r20
brne Loop
inc r17
Loop:
dec r18
tst r18
brne NumEven
ret
```

3 Контрольні питання

1. Які типи зсувних операцій підтримують МК AVR?
2. Якими способами можна модифікувати окремі біти?
3. Назвіть два основних типи безумовних переходів. У чому особливість використання команди RJMP (порівняно з JMP)?
4. Назвіть команди порівняння.

37

5. Як працюють умовні переходи?
6. Опишіть команди виклику та повернення з підпрограм.
7. Способи організації циклів.

4 Домашнє завдання

1. Вивчити розділ 2 (Основні положення).
2. Підготувати протокол лабораторної роботи.
3. Відповісти на контрольні питання.
4. Для програми з прикладу 2 запишіть коментарі і складіть блок-схему алгоритму.

5 Лабораторне завдання

Виконайте налагодження і симуляцію програми з прикладу 2 в AVR Studio. Вихідний масив заповнюється довільно вручну в режимі налагодження (у вікні Memory вибрати пам'ять даних – Data).

6 Оформлення протоколу

Протокол виконання лабораторної роботи включає: тему, мету роботи, виконання домашнього і лабораторного завдання, висновки.

7 Список рекомендованої літератури

1. 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.
2. Справка по Ассемблеру AVR – http://musmu.narod.ru/atmel/avrasm_ rus.html

Лабораторна робота № 4

ПОРТИ ВВОДУ/ВИВОДУ МІКРОКОНТРОЛЕРА ATmega16

1 Мета роботи

Метою роботи є вивчення підсистеми портів вводу/виводу мікроконтролера ATmega16 і режимів їх роботи. Набуття навичок ініціалізації портів у заданому режимі, коректного підключення світлодіодів і тактових кнопкових перемикачів, усунення ефекту «брязкоту» контактів.

2 Основні положення

МК ATmega16 (рис. 2.1) є оснащений чотирма паралельними портами вводу/виводу (порти A, B, C, D). Кожен порт включає 8 ліній, кожна з яких може налаштовуватися як на введення, так і на висновок цифрових сигналів.

Вихідні буфери всіх портів, маючи симетричні навантажувальні характеристики, забезпечують високу навантажувальну здатність при будь-якому рівні сигналу. Навантажувальної здатності досить для безпосереднього управління світлодіодними індикаторами (LED). Вхідні буфери всіх виводів побудовані за схемою тригера Шмітта. Для всіх входів є можливість підключення вбудованого резистора підтягування номіналом 100 кОм між входом і шиною живлення VCC (3,3 В або 5 В).

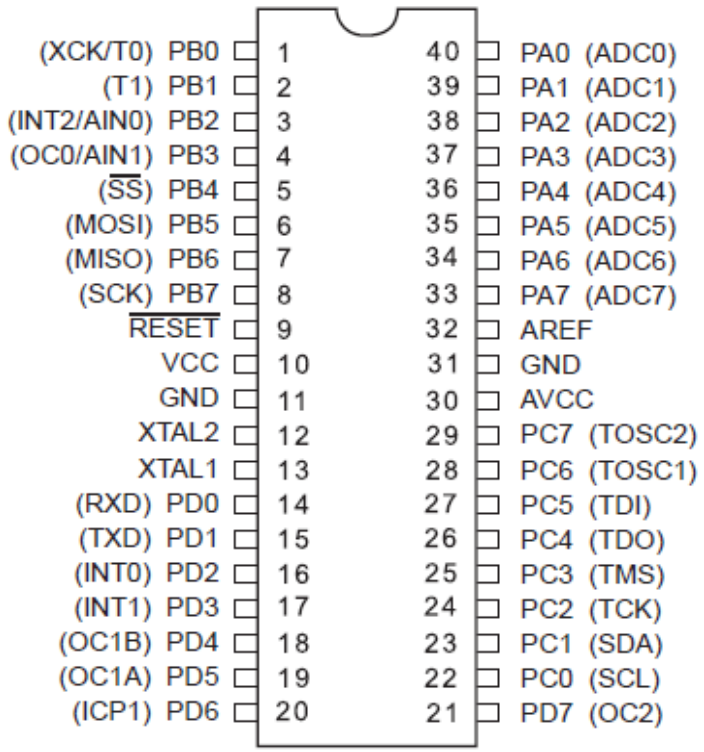


Рисунок 2.1 – Виводи мікросхеми ATmega16

39

2.1 **Регістри портів вводу/виводу**

Звернення до портів проводиться через регістри вводу/виводу. Під кожен порт в адресному просторі вводу/виводу зарезервовано по три адреси, за якими розміщуються такі регістри (табл. 2.1):

- регістри напрямку даних DDRx;
- регістри вихідних даних портів PORTx;
- вхідні регістри портів PINx.

Рівні «1» в конкретних розрядах DDRx встановлюють режим передачі даних через відповідні виводи порту *x*, рівні «0» – режим прийому. Таким чином, кожен з восьми виводів порту можна налаштувати як на передачу, так і на прийом. Розряди регістру PORTx можуть встановлюватися програмно в «0» або «1», що відповідає рівням напруги близьким до нуля і напруги живлення відповідно. За допомогою регістрів PINx здійснюється доступ до фізичних значень сигналів на виводах порту, вони є доступними тільки для читання.

Таблиця 2.1 – Регістри портів вводу/виводу ATmega16

Порт	Регістр	Адреса
A	PORTA	$1B ($3B)
	DDRA	$1A ($3A)
	PINA	$19 ($39)
B	PORTB	$18 ($38)
	DDRB	$17 ($37)
	PINB	$16 ($36)
C	PORTC	$15 ($38)
	DDRC	$14 ($37)
	PINC	$13 ($36)
D	PORTD	$12 ($32)
	DDRD	$11 ($31)
	PIND	$10 ($30)

2.2 **Режими роботи портів**

Режим виходу.

Ініціалізація режиму:

```
DDRxn = 1
```

Видача логі 4чної одиниці:

```
PORTxn = 1
```

Видача логічного нуля:

```
PORTxn = 0
```


40

Вхід з резистором підтягування. При DDRxn = 0 і PORTxn = 1 до лінії підключається внутрішній *резистор підтягування до живлення* (R = 35...120 кОм), що призводить лінію, яка є ні до чого непідключена, у стан логічної «1». Мета опору підтягування – не допустити хаотичної зміни стану на вході під впливом перешкод і наведень 50 Гц від мережі 220 В. Але якщо на вході з'явиться логічний «0» (замикання лінії на землю кнопкою або іншим приладом), то резистор не зможе утримувати напругу на лінії на рівні логічної «1» і на вході буде логічний «0».

Ініціалізація режиму:

```
DDRxn = 0
PORTxn = 1
```

Считування вхідного регістру у R16:

```
R16 = PINxn
```

Вхід без підтягування (Hi-Z) – «третій стан» або режим *високоімпедансного* входу. Такий режим є включений за замовчуванням і не задіює підтягуючий резистор, схильний до впливу перешкод і наведень від мережі. Може використовуватися для підключення певних шин.

Ініціалізація режиму:

```
DDRxn = 0
PORTxn = 0
```

Считування вхідного регістру у R16:

```
R16 = PINxn
```

Таблиця 2.2 – Стани портів вводу/виводу

DDRxn	PORTxn	Вх/Вих	Резистор підтягування	Коментар
1	0	Вихід	–	Логічний «0», ~0 В
1	1	Вихід	–	Логічний «1», ~3,3/~5 В
0	1	Вхід	Так	Вхід з підтягуванням до живлення
0	0	Вхід	Ні	Третій стан (Hi-Z)

x – найменування порту (A, B, C, D)
n – номер виводу порту (0-7)

Контакти портів вводу/виводу мікроконтролера мають додаткові функції і можуть використовуватися різними периферійними пристроями мікроконтролерів. При цьому можливі дві ситуації. В одних випадках користувач повинен самостійно задавати конфігурацію виводу, а в інших – вивід конфігурується автоматично при включенні відповідного периферійного пристрою.

2.3 Підключення світлодіодів

41

До виводів портів можна підключати світлодіодні індикатори (LED) (рис. 2.2). Вивід найбільшої довжини є анодом (+).

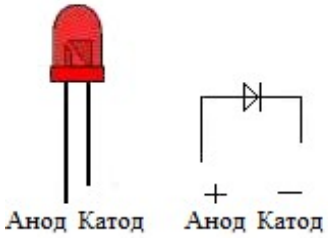


Рисунок 2.2 – Світлодіодний індикатор

При цьому потрібно враховувати реальні електричні характеристики світлодіодів. Середньостатистичні характеристики стандартних світлодіодів: падіння напруги ~2,5 В, прийнятний струм ~10 мА. Згідно з законом Ома отримуємо: R = 2,5 В / 0,010 А = 250 Ом. Отже, при підключенні світлодіоду до виводу порту мікроконтролера також необхідно послідовно підключати резистор R = ~250 Ом. Термін служби світлодіоду прямо пропорційний опору резистора. Проте, зі зростанням опору резистора, яскравість світіння знижується. Світлодіоди, що підключаються до виводів портів, можна живити як від внутрішнього джерела напруги 3,3/5 В, так і від зовнішнього (рис. 2.3).

Для схеми підключення на рис 2.4, при натисканні кнопки, на вході порту повинен встановитися рівень напруги живлення. Проте, при замиканні контакту кнопки, логічний рівень «1» на вході порту встановлюється не одномоментно.

2.4 «Брязкіт» контактів

При натисканні кнопки відбувається перехідний процес – так званий «брязкіт» контактів (рис 2.5). Таким чином, при натисканні кнопки спостерігається хаотичне перемикання контакту в проміжку часу $T_н$ ~ 20 мс. Після цього необхідно враховувати часовий інтервал для розмикання контактів (відпускання кнопки), $T_р$ ~ 100 мс.

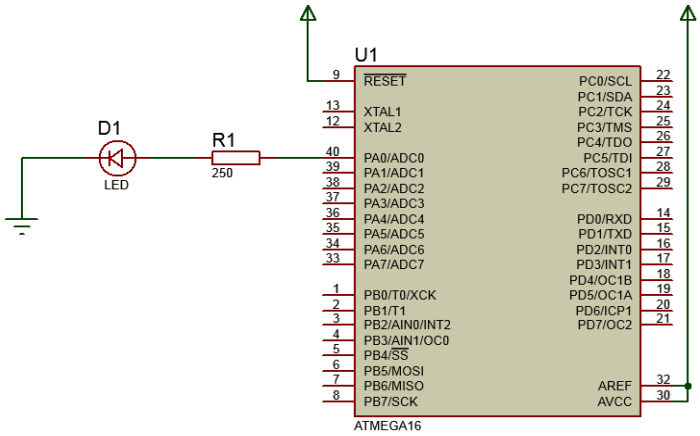
Для усунення цього явища застосовують апаратний або програмний методи. У першому випадку, використовується підключення конденсатора для погашення перехідного процесу при натисканні кнопки.

При програмному способі усунення «брязкоту», застосовується затримка часу $T_н$ ~ 20 мс після фіксації першого замикання контакту (натискання кнопки). По закінченню зазначеного часу, необхідно також очікувати поки контакт не розімкнеться (відпускання кнопки), $T_р$ ~ 100 мс.

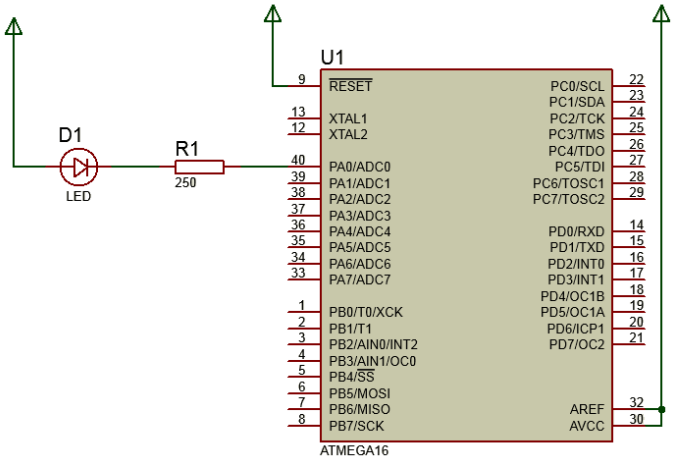
Для формування затримок часу у середовищі AVR Studio є спеціальна бібліотека – файл delay.h. При підключенні її директивою #include <utils/delay.h> стають доступними наступні функції:

```
_delay_ms (T); // T – час затримки, мс
_delay_us (T); // T – час затримки, мкс
```


42



а)



б)

Рисунок 2.3 – Підключення світлодіоду до порту мікроконтролера:
а) внутрішнє джерело напруги, режим порту – вихід;
б) зовнішнє джерело напруги, режим порту – вхід

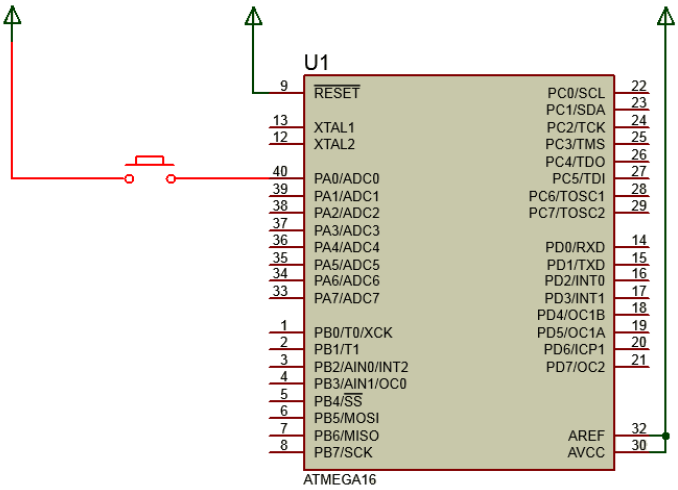


Рисунок 2.4 – Підключення кнопки до порту з внутрішнім резистором підтягування

43

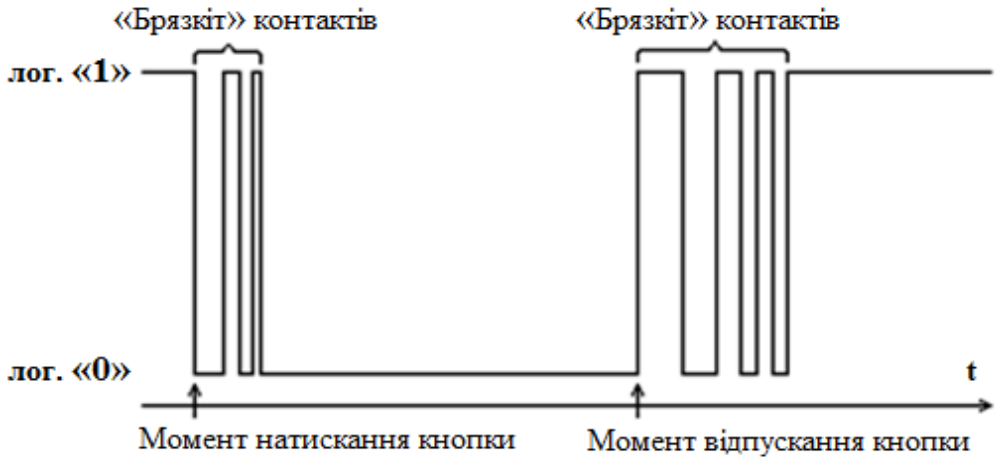


Рисунок 2.5 – Перехідні процеси при натисканні кнопки – «брязкіт» контактів

2.5 Приклад програми

До виводу 5 порта А в режимі видачі логічної «1» підключено світлодіод (анод – з боку порту) з послідовним включенням резистора номіналом 270 Ом. До виводу 0 порту В підключена кнопка в режимі входу з підтягуванням (схема на рис. 2.5). Програма реалізує наступні дії: при натисканні кнопки відбувається усунення «брязкоту» контакту і світлодіод загоряється і гаситься поперемінно з частотою 0,5 Гц.

Текст програми:

```
//Підключення стандартних бібліотек:
#include <avr/io.h> //Бібліотека вводу/виводу
#include <math.h> //Бібліотека математичних функцій
#define F_CPU 1000000UL //Тактова частота = 1 МГц
#include <util/delay.h> //Бібліотека затримки часу
//Головна функція:
int main(void)
{
    // Ініціалізація портів:
    DDRA |= (1<<5); //5-й вивід порту А - вихід
    PORTA &= ~(1<<5); //«0» на виводі 5 Порту А
    DDRB &= ~(1<<0); //0-й вивід порту В – вхід з
    PORTB |= (1<<0); //підтягуючим резистором
    while(1) //Вічний цикл
    {
        if(PINB0==0)   //Якщо   відбувається   замикання
    контакту
        {
            _delay_ms(20); //Очікування закінчення
                           //«брязкоту», 20 мс
```


44

```
            if (PINB0 == 0) //Якщо кнопка все ще
        натиснута
            {
                PORTA ^= (1<<5); //Інверсія 5-го розряду
                _delay_ms(500);  //Затримка 0,5 с
            }
            _delay_ms(100); //Затримка на відпускання
                            //кнопки, 100 мс
        }
    }
}
```

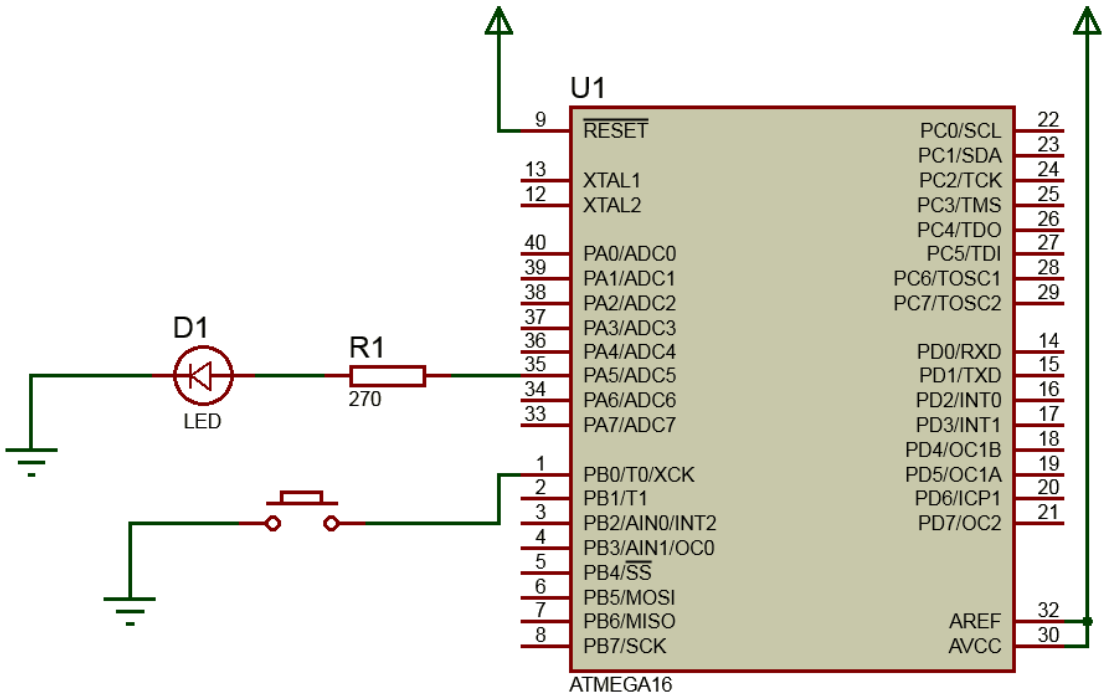


Рисунок 2.5 – Схема зі світлодіодом і кнопкою

3 Контрольні питання

1. Які порти вводу/виводу містить МК ATmega16?
2. Опишіть регістри портів та їх призначення.
3. Як задається напрямок передачі інформації в портах?
4. Опишіть можливі режими роботи виводів портів.
5. Для чого призначені внутрішні резистори підтягування портів?
6. Способи підключення світлодіоду до порту.
7. Підключення кнопки до порту. «Брязкіт» контактів і способи його усунення.

4 Домашнє завдання

45

1. Вивчити розділ 2 (Основні положення).
2. Підготувати протокол лабораторної роботи.
3. Письмово відповісти на контрольні питання.

5 Лабораторне завдання

Скласти програму мовою С ++, що реалізують такі дії:

1. Відобразити число, що задане викладачем, у двійковому вигляді на 8-ми світлодіодах, підключених до порту А.
2. При натисканні кнопки, з усуненням «брязкоту», інвертувати його і також відобразити на світлодіодах. Зобразити схему підключення в протоколі.

6 Оформлення протоколу

Протокол виконання лабораторної роботи включає: тему, мету роботи, виконання домашнього і лабораторного завдання, висновки.

7 Список рекомендованої літератури

1. 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.

46

Лабораторна робота № 5

СИСТЕМА ПЕРЕРИВАНЬ МІКРОКОНТРОЛЕРА ATmega16

1 Мета роботи

Вивчення системи переривань мікроконтролера ATmega16. Набуття навичок обробки зовнішніх переривань.

2 Основні положення

2.1 Загальні поняття

Переривання (англ. interrupt) – це сигнал, що повідомляє процесорному ядру про настання якої-небудь події. При цьому виконання поточної послідовності команд припиняється і керування передається підпрограмі – *оброблювачу переривання,* – який реагує на подію й обслуговує її, після чого повертає керування в перерваний код.

Залежно від джерела виникнення сигналу переривання поділяються на:

- *асинхронні*, або зовнішні (апаратні) – події, які надходять від зовнішніх джерел (наприклад, периферійних пристроїв) і можуть відбутися в будь-який довільний момент;
- *сінхронні*, або внутрішні – події у самому процесорі як результат порушення якихось умов при виконанні машинного коду: поділ на нуль, або переповнення стека, звернення до неприпустимих адрес пам'яті, або неприпустимий код операції;
- *програмні* (окремий випадок внутрішнього переривання) – ініціюються виконанням спеціальної інструкції в коді програми. Програмні переривання, як правило, використовуються для звернення до вбудованих функцій налагодження, драйверів та операційної системи.

Зовнішні переривання, в залежності від можливості заборони, поділяються на:

- *масковані* – переривання, які можна забороняти установкою відповідних бітів у регістрі маскування переривань;
- *немасковані* – обробляються завжди, незалежно від заборон на інші переривання.

Обробники переривань зазвичай пишуться таким чином, щоб час їх обробки був якомога меншим, оскільки під час їх роботи можуть не оброблятися інші переривання, а якщо їх буде багато (особливо від одного джерела), то вони можуть губитися.

До закінчення обробки переривання зазвичай встановлюється заборона на обробку цього типу переривання, щоб процесор не входив у цикл обробки одного переривання. Ділянки коду, які свідомо не повинні піддаватися впливу переривань, називаються *атомарними* процесами. *Пріоритизація* переривань

47

означає, що всі їхні джерела поділяються на класи і кожному класу призначається свій рівень пріоритету *запиту* на переривання. У разі одночасного виникнення різних переривань, першим буде опрацьовано переривання з найвищим пріоритетом.

Вектор переривання – закріплений за пристроєм номер, який ідентифікує відповідний обробник переривань. Вектори переривань об'єднуються в *таблицю векторів переривань*, що містить адреси обробників переривань. Місцезнаходження таблиці залежить від типу і режиму роботи процесора.

2.2 Система переривань мікроконтролера ATmega16

При виникненні переривання МК зберігає у стеку вміст регістра-лічильника команд PC (адреса наступної команди на виконання) і завантажує до нього адресу відповідного вектора переривання. За цією адресою, як правило, знаходиться команда безумовного переходу до підпрограми обробки переривання. Останньою командою підпрограми обробки переривання повинна бути команда RETI, яка забезпечить повернення в основну програму і відновлення попередньо збереженого лічильника команд.

Мікроконтролери AVR мають багаторівневу систему пріоритетних переривань. Молодші адреси пам'яті програм відведені під таблицю векторів переривання. Кожному перериванню відповідає адреса в цій таблиці. Положення вектора в таблиці також визначає і пріоритет відповідного переривання: чим менше адреса, тим вище пріоритет переривання.

У МК сімейства Mega положення таблиці векторів переривання може бути змінено. Таблиця може розташовуватися не тільки на початку пам'яті програм (за замовчуванням), а також на початку області завантажувача.

Розподіл адрес таблиці векторів переривань для ATmega16 показано в табл. 2.1.

Таблиця 2.1 – Таблиця векторів переривань ATmega16

№	Адреса	Джерело	Опис
1	$0002	INT0	Зовнішнє переривання 0
2	$0004	INT1	Зовнішнє переривання 1
3	$0006	TIMER2 COMP	Збіг таймера/лічильника T2
4	$0008	TIMER2 OVF	Переповнення таймера/лічильника T2
5	$000A	TIMER1 CAPT	Захват таймера/лічильника T1
6	$000C	TIMER1 COMPA	Збіг «A» таймера/лічильника T1
7	$000E	TIMER1 COMPB	Збіг «B» таймера/лічильника T1
8	$0010	TIMER1OVF	Переповнення таймера/лічильника T1
9	$0012	TIMER0OVF	Переповнення таймера/лічильника T0
10	$0014	SPI, STC	Передача по SPI завершена
11	$0016	USART,RXC	USART, прийом завершено

48

12	$0018	USART,UDRE	Регістр даних USART порожній
13	$001A	USART,TXC	USART, передача завершена
14	$001C	ADC	Перетворення АЦП завершено
15	$001E	EE_RDY	Готовність EEPROM
16	$0020	ANA_COMP	Аналоговий компаратор
17	$0022	TWI	Переривання від модуля TWI
18	$0024	INT2	Зовнішнє переривання 2
19	$0026	TIMER0 COMP	Збіг таймера/лічильника T0
20	$0028	SPM_RDY	Готовність SPM

Якщо переривання в роботі мікроконтролера не передбачається, то на місці таблиці векторів переривань може бути розміщена частина основної програми.

Для глобального дозволу/заборони переривань є призначений прапорець I у регістрі SREG. Для дозволу всіх переривань він повинен бути встановлений в «1», а для заборони скинутий в «0». За замовчуванням всі переривання заборонені. Індивідуальне маскування переривань виробляється установкою/скиданням відповідних розрядів регістрів масок переривань.

При виникненні переривання біт I регістру SREG апаратно скидається, забороняючи тим самим обробку наступних переривань. Проте у підпрограмі обробки переривання цей прапорець можна знову встановити в «1» для дозволу вкладених переривань. При поверненні з підпрограми обробки переривань (команда RETI) біт I встановлюється апаратно. Для встановлення і скидання біта I є такі команди Асемблера:

```
cli; I = 0, заборона всіх переривань
sei; I = 1, дозвіл всіх переривань
```

Мікроконтролери сімейства Mega підтримують чергу переривань, яка працює в такий спосіб: якщо умови генерації одного або більше переривань виникають у той час, коли прапор загального дозволу переривань скинутий, відповідні прапори встановлюються в «1» і залишаються в цьому стані до установки прапора загального дозволу переривань. Після дозволу переривань виконується їх обробка в порядку пріоритету.

Найменший час відгуку для будь-якого переривання складає 4 машинних цикли, протягом яких відбувається збереження лічильника команд в стек. Протягом наступних двох або трьох циклів виконується команда переходу до підпрограми обробки переривання. Якщо переривання відбудеться під час виконання команди, що триває кілька циклів, то генерація переривання відбудеться тільки після виконання цієї команди. Якщо ж переривання відбудеться під час перебування мікроконтролера в «сплячому» режимі, час відгуку збільшується ще на чотири машинних циклів.

49

Повернення в основну підпрограму займає також 4 цикли, протягом яких відбувається відновлення лічильника команд зі стека. Після виходу з переривання процесор завжди виконує одну команду основної програми, перш ніж обслужити будь-яке відкладене переривання.

2.3 Зовнішні переривання ATmega16

У ATmega16 мікроконтролера три зовнішніх переривання: INT0, INT1 і INT2. Ці переривання жорстко «прив'язані» до виводів PD2, PD3 і PB2 і перепризначити їх на інші висновки не можна (рис. 2.1).

Коли використовуються зовнішні переривання, виводи PD2, PD3 і PB2 конфігуруються на вхід. Проте, якщо вони є налаштовані на вихід, зовнішні переривання також будуть генеруватися при зміні їх стану, що дозволяє реалізувати програмні переривання.

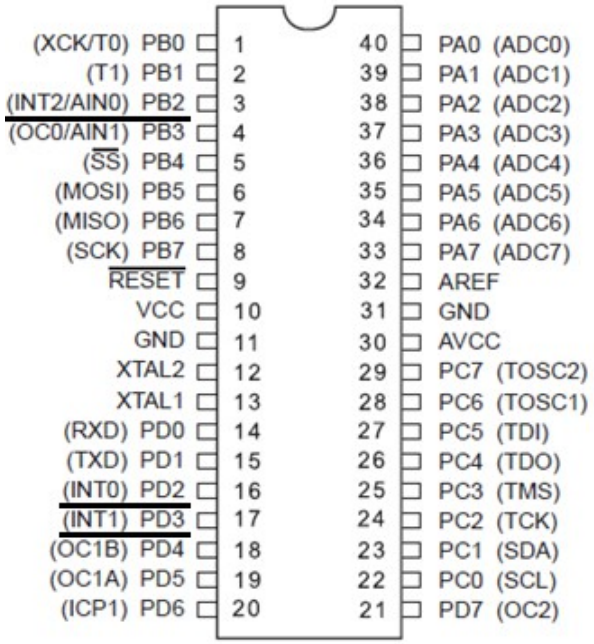


Рисунок 2.1 – Виводи зовнішніх переривань ATmega16

Для дозволу чи заборони зовнішніх переривань є призначений керуючий регістр GICR (General Interrupt Control Register) (рис. 2.2).

Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 2.2 – Регістр GICR

50

Встановлення бітів INT1, INT0 або INT2 дозволяє переривання при виникненні події на відповідному виводі МК , а скидання – забороняє. Також, потрібно встановити прапорець глобального дозволу переривань I у регістрі SREG. Без цього жодне переривання викликатися не буде.

Зовнішнє переривання може відбуватися за однієї з умов:

- за низьким рівнем на виводах INT0, INT1;
- за будь-якими змінами логічного рівня на виводах INT0, INT1;
- за спадаючим фронтом сигналу на виводах INT0, INT1, INT2;
- за зростаючим фронтом на INT0, виводах INT1, INT2.

Ілюстрація, що пояснює різницю між фронтом і рівнем сигналу, показана на рис. 2.3.

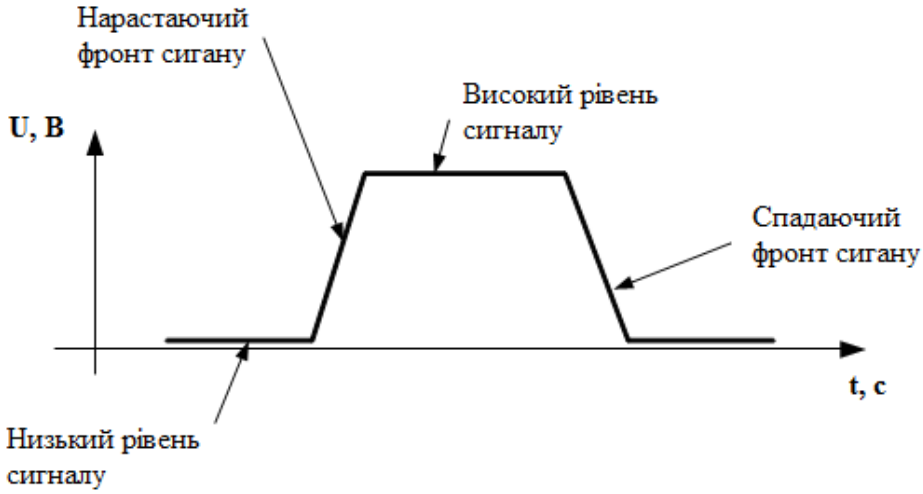


Рисунок 2.3 – Фронти сигналу

Умови генерації переривань встановлюються за допомогою конфігураційних регістрів. Для INT0, INT1 – це регістр MCUCR (MCU Control Register). Для INT2 – MCUCSR (MCU Control and Status Register)

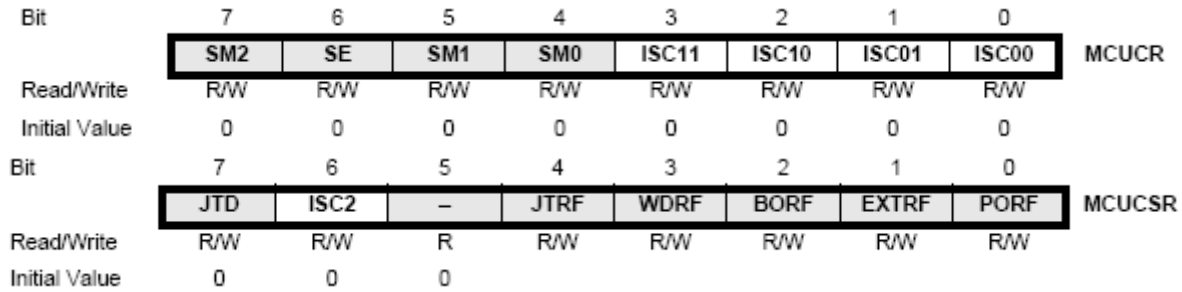


Рисунок 2.4 – Регістри MCUCR і MCUCSR

У табл. 2.2 надані можливі значення розрядів ISC01, ISC00 і відповідні їм умови генерації зовнішнього переривання.

Таблиця 2.2 – Розряди ICS01, ICS00 регістра MCUCR

ISCO1	ISCO0	Умова генерації зовнішнього переривання INT0
0	0	За низьким рівнем на INT0

51

0	1	За будь-якими змінами логічного рівня на виході INT0
1	0	За спадаючим фронтом на виході INT0
1	1	За зростаючим фронтом на виході INT0

Для переривання INT1 таблиця виглядає аналогічно, тільки керуючі розряди відповідно – ISC11 і ISC10. Переривання INT2 може відбуватися тільки по фронтах сигналу, тому для установки умов використовується всього один біт ISC2 регістру MCUCSR (табл. 2.3).

Таблиця 2.3 – Розряд ICS02 регістру MCUCR

ISC2	Умова генерації зовнішнього переривання INT2
0	За спадаючим фронтом на виході INT2
1	За зростаючим фронтом на виході INT2

При зміні значення біта ISC2 може бути згенеровано переривання INT2. Щоб цього не відбувалося, потрібно здійснювати модифікацію біта ISC2 наступним чином: заборонити зовнішнє переривання, змінити біт ISC2, скинути прапорець переривання – INTF2 (див. нижче) і знову дозволити переривання INT2.

Виявлення фронтів сигналів на виводах INT0/INT1 здійснюється сінхронно, тобто за сигналом тактового генератора. Мінімальна тривалість вхідного імпульсу, що гарантує генерацію переривання, становить один період тактового сигналу МК AVR.

Зовнішні переривання INT0/INT1, які сконфігуровані на спрацьовування за низьким рівнем, обробляються асинхронно. Для генерації переривання, рівень повинен утримуватися до закінчення виконання поточної команди. Якщо після обробки переривання рівень ще утримується, переривання буде викликано знову.

Виявлення перепадів сигналу на виводі INT2 теж здійснюється асінхронно. Мінімальна тривалість імпульсу, що гарантує генерацію переривання, становить 50 нс. Регістр, що має відношення до зовнішніх переривань, – це статусний регістр GIFR (General Interrupt Flag Register). У ньому містяться прапорці, що встановлюються в разі формування запиту на зовнішнє переривання.

Прапорці скидаються апаратно, коли викликаються обробники переривань. Також їх можна скинути програмно, записавши в регістр одиниці. Причому скидання потрібно робити шляхом перезапису регістру GIFR, а не операцією побітового АБО:

Неправильно: GIFR | = (1 << INTF1).

Правильно: GIFR = (1 << INTF1).

52

Bit	7	6	5	4	3	2	1	0	
	INTF1	INTF0	INTF2	–	–	–	–	–	GIFR
Read/Write	R/W	R/W	R/W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 2.5 – Регістр GIFR

2.4 Приклад програми

Для схеми на рис. 2.6 скласти програму на Асемблері або С ++ для МК AVR ATmega16, що реалізує попереміння перемикання стану двох світлодіодів за перериванням INT0 при натисканні кнопки (з усуненням «брязкоту» контакту).

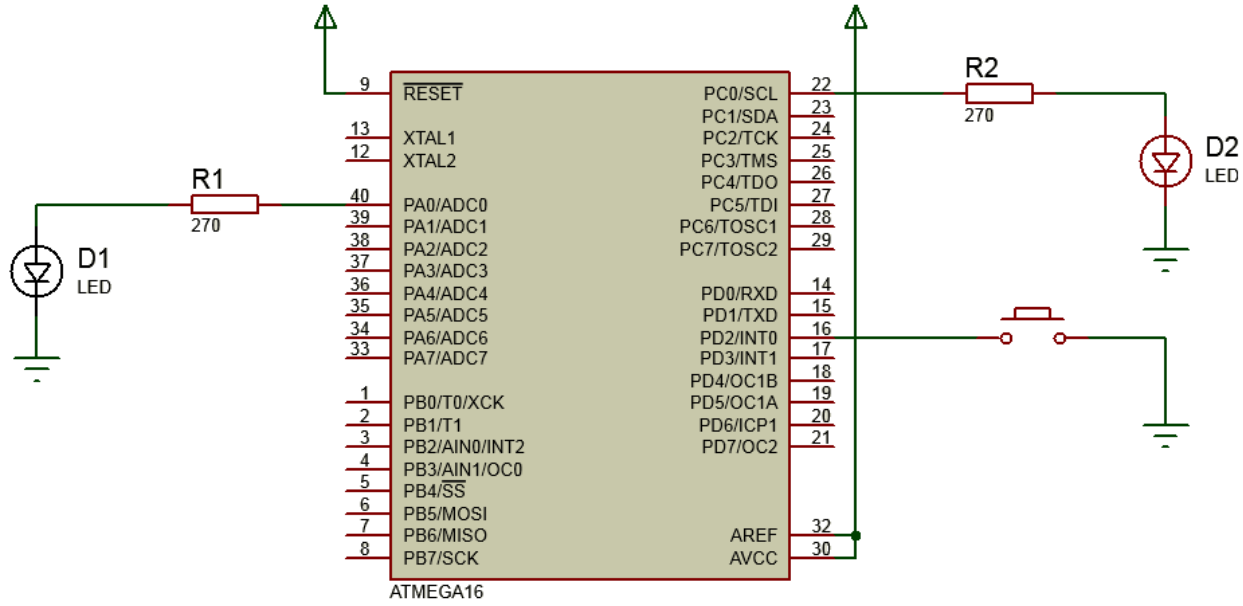


Рисунок 2.6 – Схема керування світлодіодами
за перериванням INT0 від кнопки

Текст програми на Асемблері:

```
.cseg
.org 0
rjmp Init
.org 2
rjmp EXT_INT0

.def temp1 = r16
.def temp2 = r17
```


53

```
Init:
ldi temp1,0b00000000
out DDRD,temp1
ldi r22,0b11111111
out PORTD,r22
ldi temp1,0b00000001
out DDRA,temp1
out DDRC,temp1
out PORTA,temp1
ldi r19,0x01
ldi r20,0b00000000
ldi r21,0b00000001
ldi r18,0b01000000
out GICR,r18
sei

main:
; Вічний цикл
rjmp main

EXT_INT0:
cli

ldi temp1,251
ldi temp2,26
Loop20:
dec temp1
brne Loop20
dec temp2
brne Loop20

out PORTC, r20
eor r20, r19
out PORTA, r21
eor r21, r19

ldi temp1,224
ldi temp2,130
Loop100:
dec temp1
brne Loop100
dec temp2
brne Loop100
nop
```


54

```
sei
reti
```

Текст програми на C++:

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <math.h>
#define F_CPU 1000000UL
#include <util/delay.h>
void Init(void)
{
    DDRA |= (1<<0);
    DDRC |= (1<<0);
    PORTA |= (1<<0);
    PORTC &= ~(1<<0);
    DDRD &= ~(1<<2);
    PORTD |= 0xff;

    SREG |= (1<<7);
    GICR |= (1<<6);
}

ISR(INT0_vect)
{
    asm("cli");
_delay_ms(20);
PORTA ^= (1<<0);
    PORTC ^= (1<<0);
    _delay_ms(100);
    asm("sei");
}

int main(void)
{
    Init();  //Initialization

    while(1)
    {

    }
}
```


55

3 Контрольні питання

1. Переривання та їх типи.
2. Вектор і таблиця переривань. Ініціалізація векторів.
3. Команда Асемблера для повернення з переривання і число тактів її виконання.
4. Як заборонити/дозволити все переривання?
5. Зовнішні переривання мікроконтролера ATmega16 й умови їх спрацьовування.
6. Як можуть використовуватися прапорці зовнішніх переривань?

4 Домашнє завдання

1. Вивчити розділ 2 (Основні положення).
2. Підготувати протокол лабораторної роботи.
3. Відповісти на контрольні питання.

5 Лабораторне завдання

1. Скласти схему рис. 2.6 у середовищі Proteus.
2. Створити проект в AVR Studio для програми з п. 2.4 на Асемблері або C++.
3. Скомпілювати програму й отримати файл з розширенням .hex у папці \Debug проекту AVR Studio.
4. Встановити отриманий файл як прошивку мікроконтролера на схемі в проекті Proteus.
5. Перевірити працездатність схеми і за наявності помилок налагодити програму.
6. Записати програму в протокол і розставити коментарі в рядках.

6 Оформлення протоколу

Протокол виконання лабораторної роботи включає: тему, мету роботи, виконання домашнього і лабораторного завдання, висновки.

7 Список рекомендованої літератури

1. 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.

56

57

Підписано до друку 07.11.2016 р.
Формат 60/88/16. Обсяг 3,5 друк. арк.
Тираж 100 прим. Зам. № 5960.
Віддруковано у редакційно-видавничому центрі ОНАЗ ім. О.С. Попова
м. Одеса, вул. Ковалевського, 5
тел. 70-50-494

58