МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

ЗБІРНИК МЕТОДИЧНИХ ВКАЗІВОК

до виконання лабораторних робіт з дисципліни

ПРОЕКТУВАННЯ АВТОМАТИЗОВАНИХ СИСТЕМ МОНІТОРИНГУ НАВКОЛИШНЬОГО СЕРЕДОВИЩА

для магістрів 1 курсу навчання
спеціальності 8.04010506 – Атмосферна геофізика

ОДЕСА 2016

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

ЗБІРНИК МЕТОДИЧНИХ ВКАЗІВОК

до виконання лабораторних робіт з дисципліни

ПРОЕКТУВАННЯ АВТОМАТИЗОВАНИХ СИСТЕМ МОНІТОРИНГУ НАВКОЛИШНЬОГО СЕРЕДОВИЩА

для магістрів 1 курсу навчання
спеціальності 8.04010506 – Атмосферна геофізика

«Узгоджено»
Декан факультету магістерської
та аспірантськох підготовки
_____Боровська Г.О.

«Затверджено»
на засіданні кафедри АСМНС
протокол №___від «___»_____2016 р.
_____Б.В.Перелигін

ОДЕСА 2016

Збірник методичних вказівок для виконання лабораторних робіт з дисципліни «Проектування автоматизованих систем моніторингу навколишнього середовища» лдя магістрів 1 курсу денної форми навчання за спеціальністю 8.04010506 «Атмосферна геофізика». / Великий В.І., Перелигін Б.В. – Одеса, ОДЕКУ, 2016 р. – 57 с.

ЗБІРНИК МЕТОДИЧНИХ ВКАЗІВОК

до виконання лабораторних робіт з дисципліни

ПРОЕКТУВАННЯ АВТОМАТИЗОВАНИХ СИСТЕМ МОНІТОРИНГУ НАВКОЛИШНЬОГО СЕРЕДОВИЩА

Укладачі: к.т.н., доц. Перелигін Б.В., к.т.н., доц. Великий В.І.

Підп. до друку Формат Папір
Умовн. друк. арк. Тираж Зам. №

Надруковано з готового оригінал-макета

Одеський державний екологічний університет
65016 Одеса, вул. Львівська, 15

Зміст

Вступ

Мета методичних вказівок – допомогти студентам навчитися створювати програми для мікроконтролерів, що входять до складу блоків управління автоматизованих систем моніторингу навколишнього середовища.

Програма для мікроконтролера – це набір кодів, який записується в його спеціальну програмну пам'ять. Програму пише розробник конкретної мікропроцесорної системи.

Однак програміст ніколи не має справу з кодами. Часто програміст навіть і не замислюється про те, який код відповідає тій чи іншій команді. Справа в тому, що для людини програмування в кодах незручно.

Для людини зручніше оперувати з командами, кожна з яких має свою осмислену назву. Тому для написання програм людина використовує мови програмування.

Мова програмування – це спеціально розроблена мова, що служить посередником між машиною і людиною. Як і звичайна людська мова, будь-яка мова програмування має свій словник (набір слів) і правила їх написання.

Як слова, у мові програмування виступають:

- команди (оператори);
- спеціальні керуючі слова;
- назви регістрів;
- числові вирази.

Головна задача мови – однозначно описати послідовність дій, яку повинен виконати мікроконтролер. У той же час мова має бути зручною і зрозумілою людині.

У процесі створення програми програміст пише її текст на комп'ютері точно так само, як він пише будь-який інший текст. Потім програміст запускає спеціальну програму – транслятор.

Транслятор – це спеціальна програма, яка переводить текст, написаний програмістом, в машинні коди, тобто у форму, зрозумілу для мікроконтролера.

Написаний програмістом текст програми називається вихідним, або об'єктним кодом. Код, отриманий в результаті трансляції, називається результуючим, або машинним кодом. Саме цей код записується в програмну пам'ять мікроконтролера. Для запису результуючого коду в програмну пам'ять застосовуються спеціальні пристрої – програматори.

Усі мови програмування діляться на дві групи:

- мови низького рівня (машиноорієнтовані);
- мови високого рівня.

Типовим прикладом машиноорієнтованої мови програмування є мова Ассемблера. Ця мова максимально наближена до системи команд мікроконтролера. Кожен оператор цієї мови – це, по суті, словесна назва якої-небудь конкретної команди.

В процесі трансляції така команда замінюється кодом операції. При складанні програми мовою Асемблера програміст повинен оперувати тими ж видами даних, що і сам процесор, тобто байтами і бітами.

Специфіка мови Ассемблера полягає ще й у тому, що набір операторів для цієї мови безпосередньо залежить від системи команд конкретного мікроконтролера. Тому, якщо два мікроконтролера мають різну систему команд, то і мова Асемблера для кожного такого мікроконтролера буде своєю.

На лабораторних роботах ми будемо вивчати конкретну версію язика Асемблер для мікроконтролерів AVR.

Перші чотири лабораторні роботи присвячені вивченню програмного середовища AVR Studio.

Наступні лабораторні роботи пов'язані з мікропроцесорним управлінням різними елементами і пристроями. При цьому складність виконання роботи зростає в міру збільшення її номера і в міру засвоєння студентом необхідного обсягу знань.

Всі лабораторні роботи без винятку супроводжуються докладними інструкціями, описом знову використовуваних команд, а також підсумковими лістингами необхідних програм.

При виконанні лабораторної роботи кожен студент відповідає на теоретичні питання і, потім, після отримання допуску, практично виконує роботу.

Оцінюється лабораторна робота в рамках виділених на неї в робочій навчальній програмі балів, причому 50% цих балів припадає на оцінку готовності студента до лабораторної роботи по теоретичних питаннях і 50% – на оцінку практичного виконання роботи. При отриманні студентом позитивної оцінки за відповідь на теоретичні питання він одержує допуск до виконання лабораторної роботи, після чого практично виконує лабораторну роботу. Якщо у студента немає допуску, то і роботу він не виконує.

Після демонстрації викладачеві результатів виконання лабораторної роботи і здобуття його дозволу студент оформляє звіт, після чого захищає звіт у вигляді відповідей на питання викладача про хід виконання лабораторної роботи і про її результати.

Проводиться лабораторна робота в комп'ютерному класі на персональних електронно-обчислювальних машинах.

Лабораторна робота №1

Загальні відомості про програмне середовище AVR Studio

Мета роботи – отримати загальні відомості про програмне середовище AVR Studio, про процедуру налагодження програми і про основні види налагоджувачів програм.

В результаті проведення лабораторної роботи студенти повинні знати:

- особливості роботи і можливості програмного налагоджувача відладчика;
- особливості роботи і можливості апаратного налагоджувача;
- особливості роботи і можливості повнофункціональних програмних імітаторів електронних пристроїв;
- особливості роботи і можливості внутрішнього налагоджувача мікроконтролерів AVR;
- особливості роботи програмного середовища AVR Studio;

вміти:

- застосовувати різні види налагоджувачів залежно від ситуації;
- проводити ініціалізацію пакета «AVR Studio» фірми Atmel

1 Налагодження програми

Для того, щоб написана програма перетворилася на результуючий код і запрацювала в конкретному мікропроцесорному пристрої, її потрібно відтранслювати і «зашити» в програмну пам'ять мікроконтролера.

Однак існує ще один важливий аспект цього завдання. Справа в тому, що при написанні реальної програми, особливо якщо програма реалізує досить складний алгоритм, неможливо уникнути помилок. Помилки можуть бути самі різні. Від простої синтаксичної помилки в написанні якої-небудь команди до структурних помилок, які іноді важко виявити.

У будь-якому випадку при написанні програм зазвичай не можна обійтися без процедури налагодження. Налагодження виконується на комп'ютері за допомогою спеціальної інструментальної програми – налагоджувача. Налагоджувач дозволяє покроково виконувати налагоджувану програму, а також виконує її поетапно з використанням так званих точок зупину.

В процесі виконання програми під управлінням налагоджувача програміст може на екрані комп'ютера:

- бачити вміст будь-якого регістра мікроконтролера;
- бачити вміст RAM і EEPROM;

- спостерігати за послідовністю виконання команд, контролюючи правильність відпрацювання умовних і безумовних переходів;
-спостерігати за роботою таймерів, відпрацюванням переривань.

У процесі налагодження програміст також може спостерігати логічні рівні на будь-якому зовнішньому виході мікроконтролера, а також імітувати зміни сигналів на будь-якому вході. Процес налагодження дозволяє програмувачу переконатися в тому, що розроблювана ним програма працює так, як він задумав. Більшість помилок в програмі виявляються саме в процесі налагодження.
Існує **три основних види налагоджувачів**:

- програмні;
- апаратні;
- комбіновані програмно-апаратні.

2 Програмний налагоджувач

Визначення. ***Програмний налагоджувач*** *- це комп'ютерна програма, яка імітує роботу процесора на екрані комп'ютера. Вона не вимагає наявність реальної мікросхеми або додаткових зовнішніх пристроїв і дозволяє налагодити програму чисто віртуально*.

Однак програмний налагоджувач дозволяє перевірити тільки логіку роботи програми. За допомогою такого налагоджувача неможливо перевірити роботу схеми в режимі реального часу або роботу всього мікропроцесорного пристрою в комплексі. Тобто неможливо гарантувати правильну роботу і всіх підключених до микроконтроллера додаткових мікросхем і елементів.

3 Апаратний налагоджувач

Визначення. *Другий вид налагоджувача - апаратний налагоджувач. Основа такого налагоджувача - спеціальна плата, що підключається до комп'ютера, що працює під його управлінням і імітує роботу реальної мікросхеми мікроконтролера. Плата має виводи, відповідні виводам реальної мікросхеми, на яких в процесі налагодження з'являються реальні сигнали.*

За допомогою цих виводів налагоджувана плата може бути включена в реальну схему. Виникаючі в процесі налагодження електричні сигнали можна спостерігати за допомогою осцилографа. Можна натискати реальні кнопки і спостерігати роботу світлодіодів та інших індикаторів.

У той же самий час на екрані комп'ютера ми так само, як і в попередньому випадку, можемо бачити всю інформацію про налагоджувану програму:
- спостерігати вміст регістрів, RAM, портів вводу-виводу;
- контролювати хід виконання програми.

З апаратним налагоджувачем ми можемо так само, як і з програмним, виконувати програму в покроковому режимі і застосовувати точки зупину.

Недоліком апаратного налагоджувача є його висока вартість.

4 Повнофункціональні програмні імітатори електронних пристроїв

Існує і **третій вид налагоджувача**. Це повнофункціональні програмні імітатори електронних пристроїв. Такі програми дозволяють на екрані комп'ютера «зібрати» будь-яку електронну схему, що включає в себе самі різні електронні компоненти:

- транзистори;
- резистори;
- конденсатори;
- операційні підсилювачі;
- логічні та цифрові мікросхеми, в тому числі і мікроконтролери.

Такі програми зазвичай містять великі бази електронних компонентів і конструктор електронних схем. Зібравши схему, можна віртуально записати в пам'ять мікроконтролера програму, а потім «запустити» всю схему в роботу.

Для контролю результатів роботи схеми імітатор має віртуальні вольтметри, амперметри та осцилографи, які можна «підключати» до будь-якої точки схеми, «вимірювати» різні напруги, а також «знімати» часові діаграми.

Такі програми в даний час отримують все більше поширення. Вони дозволяють розробити будь-яку схему з мікроконтролером або без нього, без використання паяльника і реальних деталей. На екрані комп'ютера можна повністю налагодити схему і лише потім братися за паяльник.

Недоліком даного налагоджувача є те, що він потребує значних обчислювальних ресурсів. Особливо в тому випадку, коли налагоджувана схема включає як мікроконтролер, так і деяку аналогову частину. Крім того, імітатор не завжди вірно імітує роботу деяких пристроїв. Однак подібні програми мають дуже великі перспективи.

5 Внутрішній налагоджувач мікроконтролерів AVR

Ще один апаратний спосіб налагодження закладений конструктивно в деякі моделі мікроконтролерів AVR. Зокрема, мікроконтролер ATiny2313 підтримує такий спосіб налагодження.

Для забезпечення можливості апаратного налагодження такі мікроконтролери мають, по-перше, **спеціальну однопровідну лінію debugWIRE**, яка зазвичай поєднана з входом RESET. Ця лінія використовується спеціальною платою – налагоджувачем для управління мікроконтролером в процесі налагодження. Крім того, в систему команд такого мікроконтролера включена

команда break, яка може використовуватися **для створення програмних точок зупину**.

Для того, щоб використовувати подібний режим налагодження, необхідно мати у своєму розпорядженні **спеціальну налагоджувальну плату**, яка повинна підтримувати цей режим. Крім того, подібний режим повинна підтримувати і **інструментальна програма-налагоджувач**.

У процесі налагодження програміст проставляє на екрані комп'ютера в потрібних місцях програми, що налагоджується, точки зупину. Потім він запускає цю програму під управлінням налагоджувача. Налагоджувач автоматично вставляє в програму, що налагоджується, команди break в тих місцях, де програміст поставив точки зупину. А команди, які повинні бути записани в місці вставки команд break, запам'ятовує в своїй пам'яті.

Потім він автоматично «прошиває» отриманий таким чином текст програми в програмну пам'ять налагоджуваного мікроконтролера і запускає її в роботу. Мікроконтролер виконує закладену в нього програму до тих пір, поки не зустрінеться команда break. Отримавши цю команду, мікроконтролер призупиняє виконання програми і передає управління налагоджувачу.

Далі налагоджувач управляє мікроконтролером за допомогою **інтерфейсу debugWIRE**. Цей інтерфейс дозволяє читати вміст всіх регістрів мікроконтролера та інших видів пам'яті. Прочитана інформація відображається на екрані комп'ютера. Потім налагоджувач чекає команд від оператора. Під управлінням налагоджувача мікроконтролер може примусово виконати будь-яку команду зі своєї системи команд.

Це дає можливість легко реалізувати покрокове виконання про-грами, а також виконання тих команд, які були замінені на break. Все управління здійснюється за допомогою інтерфейсу debugWIRE, який дозволяє передавати інформацію як від налагоджувача в мікроконтролер, так і в зворотному напрямку.

Перевагою такого способу налагодження є те, що в даному випадку відбувається не імітація мікроконтролера, а використовується реальна мікросхема. При цьому робота в режимі налагодження найбільш повно наближається до реального режиму роботи.

Недолік – часте «перешивання» програмної пам'яті мікро-контролера. Змінювати вміст цієї пам'яті доводиться щоразу при установці нових або знятті старих точок зупину. Якщо врахувати, що допустима кількість перезапису програмної пам'яті становить 10 000 циклів, то при тривалому процесі налагодження ця кількість може вичерпатися, і мікросхема вийде з ладу.

6 Програмне середовище «AVR Studio»

Фірма Atmel, розробник мікроконтролерів AVR, дуже добре подбала про супровідь своєї продукції. Для написання програм, їх налагодження, трансляції і прошивки в пам'ять мікроконтролера фірма розробила і безкоштовно розпов-

сюджує **спеціалізовану середу розробника під назвою «AVR Studio»**. Інсталяційний пакет цієї інструментальної програми можна вільно скачати з сайту фірми. Адреса сторінки для скачування програм:

http://www.atmel.ru/Software/Software.htm.

Програмна середа «AVR Studio» – це потужний сучасний програмний продукт, що дозволяє проводити всі етапи розробки програм для будь-яких мікроконтролерів серії AVR. Пакет включає в себе спеціалізований текстовий редактор для написання програм, потужний програмний налагоджувач.

Крім того, «AVR Studio» дозволяє керувати цілим рядом зовнішніх пристроїв, що подключаються до комп'ютера. Це дозволяє виконувати апаратне налагодження, а також програмування («прошивку») мікросхем AVR.

Познайомимося докладніше з цим зручним програмним інструментом для програмістів. Програмна середа «AVR Studio» працює не просто з програмами, а з проектами. Для кожного проекту повинний бути відведений свій окремий каталог на жорсткому диску. В AVR Studio одночасно може бути завантажений тільки один проект.

При завантаженні нового проекту попередній проект автоматично вивантажується. Проект містить всю інформацію про розроблювану програму і вживаний микроконтроллер. Він складається з цілого набору файлів.

Головний з них – **файл проекту**. Він має розширення aps. Файл проекту містить відомості про тип процесора, частоту тактового генератора і т.д. Він також містить опис всіх інших файлів, що входять в проект. Всі ці відомості використовуються при налагодженні і трансляції програми.

Крім файлу *aps*, проект повинний містити хоча б один **файл з текстом програми**. Такий файл має розширення *asm*. Недостатньо просто помістити файл *asm* в директорію проекту. Його потрібно ще включити в проект. Як це робиться, ми побачимо трохи пізніше. Проект може містити кілька файлів *asm*. При цьому один з них є головним. Інші можуть викликатися з головного за допомогою оператора *.include*. На цьому закінчується список файлів проекту, які створюються за участю програміста.

Але типовий проект має набагато більше файлів. Решта файлів проекту з'являються в процесі трансляції. Якщо програма не має критичних помилок і процес трансляції пройшов успішно, то в директорії проекту автоматично з'являються такі файли: файл, що містить результуючий код трансляції в hex форматі, файл map, що містить всі символьні імена програми, що транслюється, зі своїми значеннями, лістинг-трансляції (*lst*) та інші допоміжні файли. Однак для нас буде важливий лише *hex*-файл (файл з розширенням *hex*). Саме він буде служити джерелом даних при прошивці програми в програмну пам'ять мікроконтролера.

7 Завдання для самостійної роботи

7.1 Вивчити особливості роботи і можливості програмного налагоджувача;
7.2 Вивчити особливості роботи і можливості апаратного налагоджувача;
7.3 Вивчити особливості роботи і можливості повнофункціональних програмних імітаторів електронних пристроїв;
7.4 Вивчити особливості роботи і можливості внутрішнього налагоджувача мікроконтролерів AVR;
7.5 Вивчити особливості роботи програмного середовища AVR Studio

8 Завдання до лабораторної роботи

8.1 Запротоколювати особливості роботи програмного налагоджувача;
8.2 Запротоколювати особливості роботи апаратного налагоджувача;
8.3 Запротоколювати особливості роботи повнофункціональних програмних імітаторів електронних пристроїв;
8.4 Запротоколювати особливості роботи і можливості внутрішнього налагоджувача мікроконтролерів AVR;
8.5 Запротоколювати особливості роботи програмного середовища AVR Studio

9 Зміст звіту

9.1 Мета роботи;
9.2 Підсумки виконання пунктів 8.1-8.5;
9.3 Висновки.

Контрольні питання

10.1 У чому особливості роботи і можливості програмного налагоджувача?
10.2 У чому особливості роботи і можливості апаратного налагоджувача?
10.3 У чому особливості роботи і можливості повнофункціональних програмних імітаторів електронних пристроїв?
10.4 У чому особливості роботи і можливості внутрішнього налагоджувача мікроконтролерів AVR?
10.5 У чому особливості роботи програмного середовища AVR Studio?

Лабораторна робота №2

Вивчення головної панелі програми «AVR Studio»

Мета роботи – вивчення головної панелі і її модифікацій інтерфейсу програми «AVR Studio»

В результаті проведення лабораторної роботи студенти повинні
знати:

- призначення та особливості інформаційного вікна про поточний проект;
- призначення та особливості вікна виведення повідомлень;
- призначення та особливості вікна з текстами програм на Асемблері і вікон будь-яких відкритих програмою файлів

вміти:

- отримувати інформацію про ресурсах контролерів AVR фірми Atmel;
- користуватися всілякими вікнами інтерфейсу пакета «AVR Studio» фірми Atmel

1 Опис інтерфейсу. Головна панель програми «AVR Studio»

На рис. 2.1 показано, як виглядає головна панель програми «AVR Studio». Насправді «AVR Studio» має дуже гнучкий інтерфейс, і зовнішній вигляд може сильно відрізнятися від варіанту, показаного на рисунку. Але ми будемо розглядати випадок, коли обрані установки за замовчуванням.

Головна панель програми AVR Studio розділена на три основних вікна. На рис. 2.1 вони позначені цифрами 1, 2 і 3 Перші два вікна - допоміжні. Вікно 1 надає нам повну інформацію про поточний проект. За умовчанням це вікно включає в себе три вкладки. «Корінці» цих вкладок можна побачити в нижній частині вікна.

Перша вкладка називається «Info». Вона містить довідкову інформацію по використовуваному микроконтроллеру: опис векторів переривань; опис контактів для різних корпусів та короткий опис регістрів.

Наступна вкладка називається «Project». Вона містить інформацію по поточному завантаженому проекту. Інформація представлена у вигляді дерева. Різні гілки цього дерева описують всі вихідні і результуючі файли проекту, усі мітки, процедури і файли, що приєднуються

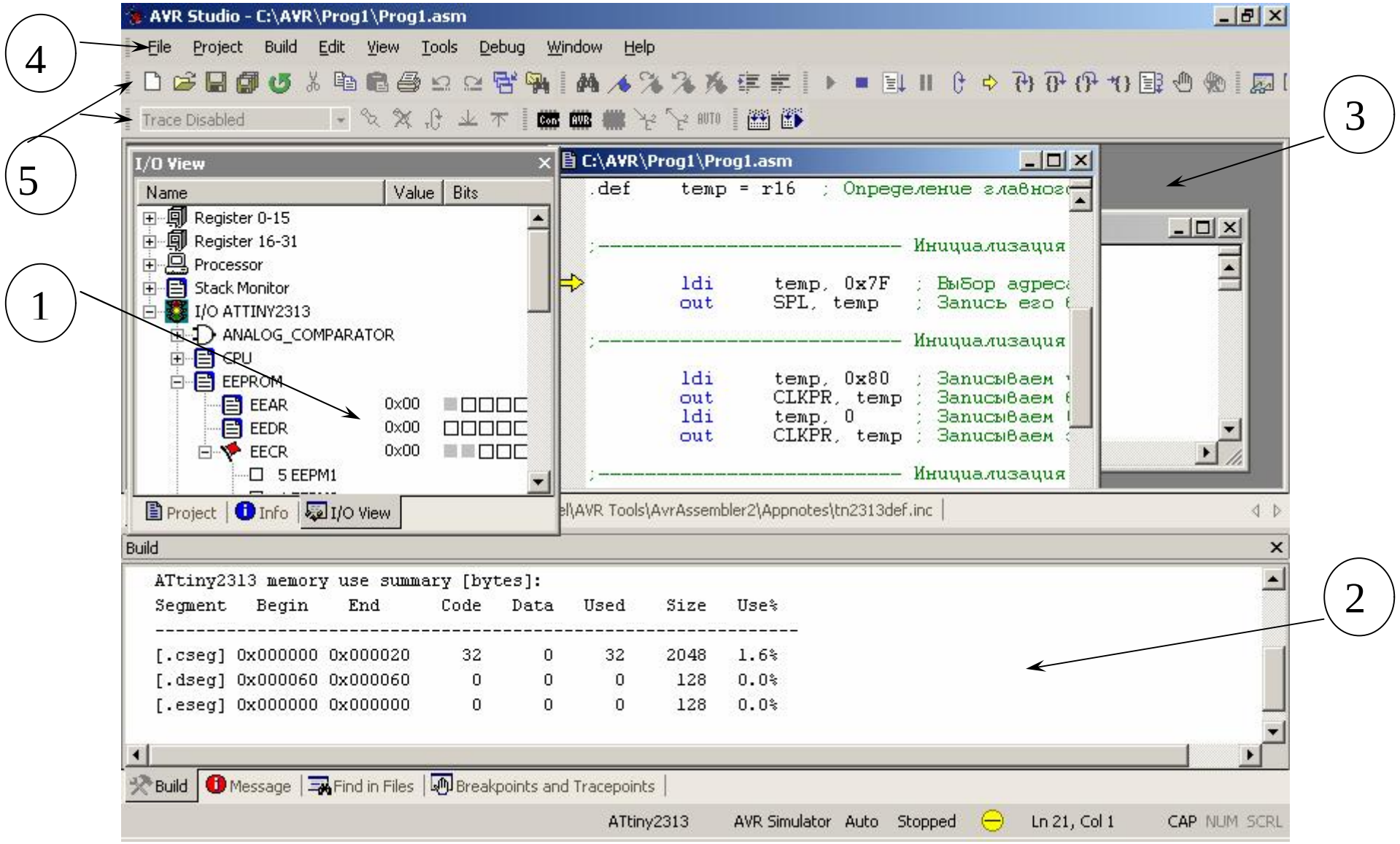

Рисунок 2.1 - Головна панель програми «AVR Studio»

Остання вкладка вікна номер 1 називається «I / O View» (перегляд вводу-виводу). Це найкорисніша вкладка. На ній в графічному вигляді показані всі ресурси мікроконтролера:

-порти введення - виведення;

- таймери;

- компаратори;

- АЦП;

- регістри загального призначення і т.д.

Вся інформація також представлена у вигляді дерева. Кожна «гілка» цього дерева – це окремий елемент. Якщо який-небудь елемент складається з інших елементів, то його можна розкрити і побачити ці елементи.

Елементи, що з'являються в результаті розкриття гілки, також можуть бути розкриті, якщо вони мають свій вміст. На рис. 2.2 в збільшеному вигляді показано дерево ресурсів мікроконтролера ATiny2313. На рисунку кілька гілок спеціально розкриті, щоб можна було побачити їх склад.

Якщо яка-небудь гілка може бути розкрита, то в своїй основі вона має квадратик з хрестиком всередині. Подвійне клацання на цьому хрестику розкриває гілку. У розкритій гілці хрестик перетворюється в тіре. Повторний подвійне клацання по квадратику закриває розкриту гілка.

На рис. 2.2 для наочності розкриті гілки всіх трьох портів вводу-виводу і регістри, пов'язані з EEPROM. Ви можете бачити:

-повний склад керуючих регістрів для кожного з пристроїв;

- їх назви та адреси;

- склад і назва кожного біта (якщо біти мають свої назви).

Також для наочності на рис. 2.2 розкрита гілка, відповідна регістру EECR, і можна бачити всі його біти.

У процесі налагодження в цьому вікні можна побачите не тільки назву і склад усіх ресурсів, але і їх вміст. Вміст буде відображатися як в шістнадцятковому коді, так і шляхом затемнення квадратиків, що відображають окремі біти конкретних регістрів.

Затемнений квадратик означає, що біт дорівнює одиниці. **Світлий квадратик** говорить про те, що біт дорівнює нулю. Розробник також можете оперативно міняти їх вміст прямо в цьому вікні. Для зміни значення біта достатньо подвійного клацання мишки у відповідному квадратику. Маються також інші способи зміни вмісту різних регістрів і комірок пам'яті в процесі налагодження.

У нижній частині головної панелі знаходиться **друге допоміжне вікно** (вікно 2 на рис. 2.1). Це вікно служить, в основному, для виведення різних повідомлень. Воно також містить ряд вкладок. За замовчуванням їх чотири.

Перша вкладка називається «Build». На вкладці «Build» відбивається процес трансляції. На цю вкладку виводяться повідомлення про різні етапи трансляції, повідомлення про синтаксичні помилки і різні попередження (Warnings).

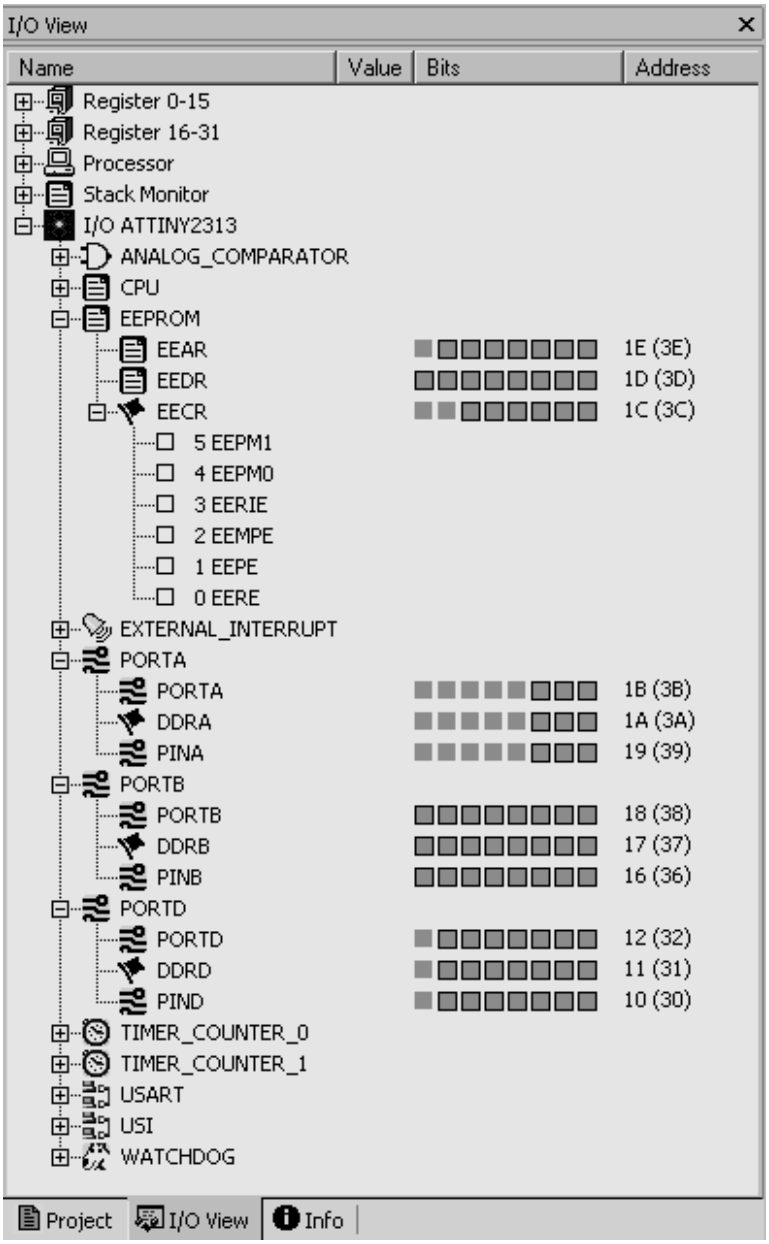

Рисунок 2.2 – Вікно ресурсів мікроконтролера

Якщо трансляція закінчується нормально (відсутні критичні помилки), то сюди ж виводяться статистичні дані про отриманий результуючий код. Ці дані показують розміри і відсоток використання всіх видів пам'яті мікроконтролера. Наприклад, після трансляції одного із завдань програма видасть таке повідомлення:

```
ATtiny2313 memory use summary [bytes]:
Segment   Begin      End      Code   Data   Used   Size   Use%
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
[.cseg]   0x000000   0x0004f2   508    758    1266   2048   61.8%
[.dseg]   0x000060   0x00009d     0     61      61    128   47.7%
[.eseg]   0x000008   0x000045     0     61      61    128   47.7%

Assembly complete, 0 errors. 0 warnings
```

Повідомлення означає, що в програмному сегменті використані комірки з адреси 0x000000 по адресу 0x0004f2. При цьому власне код програми займає 508 байт. Дані в програмній пам'яті займають 758 байт. Всього використано в програмній пам'яті 1266 байт (сума попередніх двох чисел). Розмір програмної пам'яті для цього мікроконтролера становить 2048 байт. Відсоток використання програмної пам'яті 61,8%.

Точно такі ж відомості наведені для пам'яті даних (RAM) і для EEPROM. Природно, що два останні види пам'яті не містять программного коду. Тому у відповідному стовпчику стоять нулі. Останній рядок містить повідомлення про помилки. В даному випадку повідомлення перекладається так: «Ассемблювання пройшло успішно, 0 помилок, 0 попереджень».

Друга вкладка другого вікна називається «Message». Тут виводяться різні системні повідомлення про завантаження модулів програми і т.п.

Третя вкладка другого вікна називається «Find in Files» (пошук в файлах). В цьому вікні відображаються результати виконання команди «Пошук в Файлах». Ця команда дозволяє проводити пошук заданої послідовності символів відразу у всіх файлах проекту. По закінченні пошуку у вкладці «Find in Files" відображаються всі знайдені входження із зазначенням імені файлу і рядки, де знайдена шукана послідовність.

Остання вкладка називається «Breakpoints and Tracepoints»

(Точки зупину і точки трасування). Ці точки проставляються в тексті програми перед початком процесу налагодження і дублюються в даному вікні. Як проставляти точки зупину, ми довідаємося трохи пізніше.

Точки зупину використовуються для того, щоб призупинити виконання програми в тому чи іншому місці програми, щоб переконатися, що програма виконується правильно. При створенні точки зупину в тексті програми вона автоматично з'являється у вкладці «Breakpoints and Tracepoints».

Вкладка дозволяє побачити всі точки зупину програми в одному місці. Крім того, на вкладці проти кожного запису, який описує точку зупину, автоматично з'являється «Check box» (поле вибору), за допомогою якого можна в будь-який момент тимчасово відключити будь-яку точку зупину.

Точки трасування використовуються для управління процесом трасування.

Визначення. Трасування – *це особливий вид процесу налаштування, коли програма запускається і виконується в автоматичному режимі.*

Але в процесі роботи вона залишає повідомлення в спеціальному вікні. Повідомлення відображають кожен крок виконуваної програми. Точки трасування можуть скасувати і заново дозволити трасування на різних ділянках програми.

Програмна середа «AVR Studio» підтримує трасування тільки при роботі із платою налагодження ICE50. Це досить дорогий пристрій. Тому ми зупинимося лише на програмному налагоджувачі без застосування будь-яких апаратних засобів налагодження.

Цього цілком достатньо для розробки мікропроцесорних пристроїв практично будь-якої складності. Апаратні налагоджувачі необхідні в умовах промислового виробництва для прискорення робіт з розробки нових виробів.

Будь-яку з вкладок будь-якого вищеописаного вікна можна приховати або, навпаки, перетворити в окреме вільно переміщуване вікно. Для цього достатньо натиснути правою клавішею миші по заголовку відповідної вкладки і вибрати в меню потрібний режим. Пункт «Hide» цього меню означає «Приховане» (невидиме), «Floating» означає «Вільне» (переміщуваний), «Docking» - «Закріплене».

Для деяких користувачів буває важко повернути вкладку на місце після того, як вона перетвориться на вільно переміщуване вікно. У програмі «AVR Studio» використовується нестандартний досить оригінальний механізм управління вікнами. Припустимо, що ми випадоково перетворили на плаваюче вікно вкладку «Breakpoints and Tracepoints» вікна номер два. Подивимося, як можна поставити її на місце.

Якщо переміщати це вікно за допомогою миші (утримуючи його за заголовок), то на основній панелі програми з'являються покажчики розміщення, як це показано на рис. 2.3. Вони являють собою стилізовані стрілки синього кольору, розташовані по всьому полю головного вікна програми. Одночасно з'являються вісім таких стрілок. Чотири з них об'єднані в центральний блок, в який включена ще й кругла кнопка посередині.

Цей блок автоматично розташовується в центрі того вікна, в межах якого в даний момент переміщається курсор. На рис. 2.3 цей покажчик розташований в центрі вікна номер два. Інші чотири стрілки розташовуються зверху, знизу, праворуч і ліворуч від основного вікна програми. Досить перемістити курсор миші разом з переміщуваним плаваючим вікном на одну з цих стрілок і відпустити кнопку миші, і вікно тут же вбудується в одне з допоміжних вікон програми у вигляді вкладки або утворює нове допоміжне вікно. Причому ви ще до відпускання кнопки миші можете побачити, куди потрапить вікно.

Як тільки ваш курсор суміститься з однією із стрілок, програма зафарбує синім кольором цю область. Поекспериментуйте самі з розміщенням вікон. Пам'ятайте тільки, що стрілки перетворюють ваше плаваюче вікно у ще одне фіксоване додаткове вікно в різних частинах інтерфейсу. А кругла кнопка посеред центрального блоку перетворює плаваюче вікно в додаткову вкладку вже

існуючого вікна. Саме за допомогою цієї кнопки відірване від свого звичного місця плаваюче вікно на рис.2.3 можна повернути на своє місце.

Додаткові вікна 1 і 2 дозволяють легко змінювати свої розміри. Для зміни розміру достатньо перетягнути межу вікна за допомогою миші. Можна навіть приховати будь-яке з цих вікон, закривши всі його вкладки. Закрити вкладку можна двома способами. Або клацнути по її «корінцю» правою кнопкою миші, а в меню вибрати пункт «Hide». Або клацнути мишею в хрестик у верхньому правому куті вкладки. Відкрити закриті вкладки можна за допомогою меню «View / Toolbars».

Особливу роль відіграє вікно 2.3. Це навіть не вікно, а частина, що залишилася від головного вікна програми. Якщо закрити вікна 1 і 2, вікно 3 займе весь простір програмної панелі. У вікні 3 з'являються різні робочі вікна.

По-перше, це вікна з текстами програм на Ассемблері. А, по-друге, тут можуть з'являтися вікна будь-яких відкритих програмою файлів. Це можуть бути текстові файли або файли інших програм. Кожен такий файл за умовчанням відкривається у вигляді окремого плаваючого вікна. Для визначеності будемо називати такі вікна текстовими вікнами. Текстові вікна будуть «плавати» тільки всередині вікна 3.

Для кожного нового текстового вікна в нижній частині вікна 3 з'являється «корінець», за допомогою якого можна швидко перейти до потрібного вікна, якщо воно не знаходиться на передньому плані. При подвійному натисканні лівою кнопкою миші по заголовку будь-якого текстового вікна воно розкриється на всю ширину вікна 3. Іноді саме так зручно працювати з текстами програм.

У вікні 3 можна відкривати не тільки всі тексти ассемблерних програм поточного проекту, а й тексти програм інших проектів, а також тексти програм, написаних на інших мовах програмування. Такий прийом дуже зручний, якщо потрібно переробити програму, написану для старого мікроконтролера на старій версії Ассемблера, на новий лад. Всі відкриті текстові вікна запам'ятовуються і потім відкриваються автоматично при відкритті проекту.

Будь-яке текстове вікно має **підсвічування синтаксису**. Різні частини поміщеного туди тексту програми підсвічуються різними кольорами. Так, всі оператори Ассемблера висвічуються **блакитним** кольором. Коментарі виділяються **зеленим**. Решта тексту (параметри команд, псевдооператор, мітки, змінні і константи) залишається **чорним**. Це дуже зручно. Якщо написаний вами оператор убрався в блакитний колір, то це означає, що ви не помилилися в синтаксисі. Якщо ви написали коментар, але перед текстом коментаря забули поставити крапку з комою, то цей коментар не забарвиться в зелений колір. Таким чином, багато помилок видно вже в процесі написання програми.

Крім двох допоміжних і одного основного вікна, головна панель програми має **рядок меню** (відзначений цифрою 4 на рис. 2.1), а також декілька інструментальних панелей (відзначені цифрою 5). Як і в будь-який інший програмі під Windows, за допомогою меню викликаються всі функції програми

AVR Studio і переключаються всі її режими. Панелі інструментів дублюють часто використовувані функції меню.

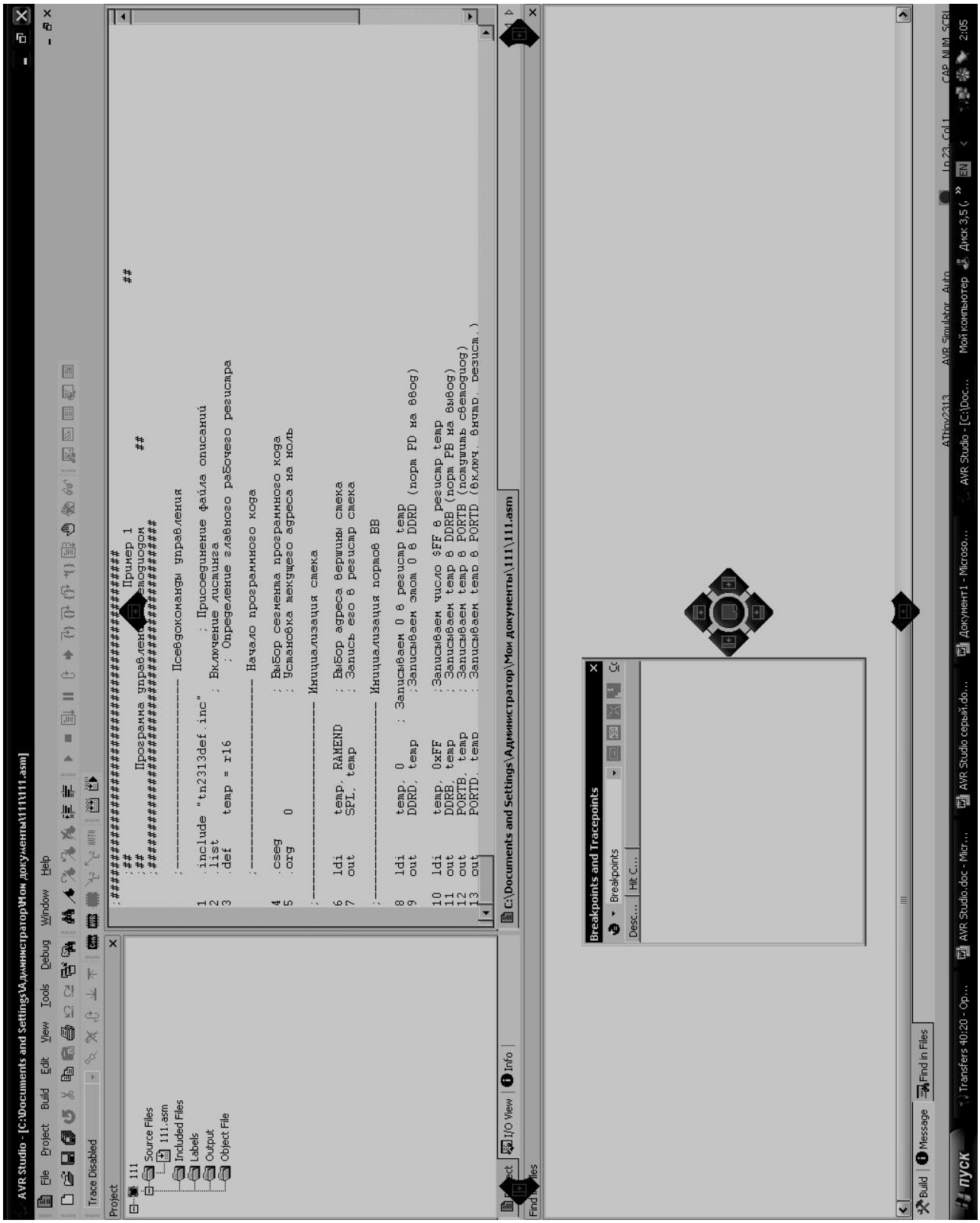

Рисунок 2.3 - Переміщення вікна

2 Завдання для самостійної роботи

2.1 Вивчити призначення та особливості інформаційного вікна про поточний проект;

2.2 Вивчити призначення та особливості вікна виведення повідомлень;

2.3 Вивчити призначення та особливості вікна з текстами програм на Асемблері і вікон будь-яких відкритих програмою файлів

3 Завдання до лабораторної роботи

3.1 Виконати операції в інформаційному вікні поточного проекту і запротоколювати їх. Підтвердити ілюстраціями;

3.2 Виконати операції у вікні виводу повідомлень і запротоколювати їх. Підтвердити ілюстраціями;

3.3 Виконати операції у вікні з текстами програм на Ассемблері і в вікнах будь-яких відкритих програмою файлів про поточний проект і запротоколювати їх. Підтвердити ілюстраціями

4 Зміст звіту

4.1 Мета роботи;

4.2 Підсумки виконання пунктів 3.1-3.3;

4.3 Висновки.

Контрольні питання

5.1 Який склад інтерфейсної панелі програмного середовища «AVR Studio»?

5.2 Яке призначення і особливості інформаційного вікна поточного проекту?

5.3 Яке призначення і особливості вікна виведення повідомлень?

5.4 Яке призначення і особливості вікна з текстами програм на Ассемблері і вікон будь-яких відкритих програмою файлів?

5.5 Що таке вікно ресурсів мікроконтролера і його можливості?

Лабораторна робота №3

Вивчення етапів створення проекту і трансляція програми

Мета роботи – вивчити етапи створення нового проекту на основі Ассемблера (Atmel AVR Assembler) і процедуру трансляції

В результаті проведення лабораторної роботи студенти повинні
знати:

- етапи створення нового проекту на основі Ассемблера (Atmel AVR Assembler);
- формати hex і ЄЄР - файлів;
- особливості процедури трансляції

вміти:

- створювати нові проекти на основі Ассемблера (Atmel AVR Assembler);
- працювати з форматами hex і ЄЄР - файлів

1 Створення проекту

Припустимо, що програма AVR Studio встановлена на комп'ютер, запущена і знаходиться в початковому стані (усі вкладки вікон 1 і 2 порожні, вікно 3 не містить відкритих файлів). Приступимо до створення нового проекту.

Для цього виберемо в меню «Project» пункт «New Project». На екрані з'явиться вікно побудовника. В поле «Project Type:» вибираємо **тип майбутнього проекту**. Програма пропонує два варіанти:

- проект на Ассемблері (Atmel AVR Assembler);
- проект на мові CI + + (AVR GCC).

Вибираємо **Ассемблер**. Потім у полі «Project name:» вибираємо ім'я проекту. Наприклад, Progl. Відразу під полем з ім'ям проекту розташовані два елемента вибору режимів – так звані «Чек-бокси» (Check box). За замовчуванням обидва чек-бокси обрані (тобто, у відповідних квадратиках проставлені «галочки»).

Перший чек-бокс (Create initialize file) визначає, чи потрібно автоматично створювати головний програмний файл. Якщо у вас вже є файл з текстом програми на Ассемблері і ви просто хочете створити проект, а потім підключити туди готовий програмний файл, зніміть відповідну «галочку». Якщо ви створюєте проект «з нуля», залиште «галочку" неторкнутої.

Другий чек-бокс (Create folder) визначає, чи потрібно автоматично створювати окремий каталог для даного проекту. Якщо ви заздалегідь вже створили потрібний каталог засобами Windows, зніміть позначку. Якщо ні, залиште. Наступне поле називається «Initial file». Воно повинно містити ім'я файлу, куди буде записуватись текст програми. За замовчуванням ім'я файлу вже вписане в

це поле. Воно відповідає імені проекту. Рекомендується залишити його без змін.

Ще одне поле, яке потребує нашого втручання, – це **поле «Location»**. Тут ви повинні вказати шлях до того місця на жорсткому диску, де зберігатиметься проект. Шлях не можна ввести безпосередньо з клавіатури. Для зміни шляху потрібно натиснути кнопку праворуч, на якій в якості назви поставлені крапки («...»).

Відкриється діалог «Select folder», за допомогою якого ви і повинні вибрати директорію. Просто увійдіть в потрібну директорію і натисніть кнопку «Select». При виборі директорії потрібно враховувати значення чек-боксу «Create folder». Якщо там стоїть «галочка», то при виборі в якості Location каталогу «c: \ AVR \ myprog», програма помістить ваш проект в каталог «c: \ AVR \ myprog \ Progl».

На цьому можна закінчити роботу з першим вікном побудовника. Але перш, ніж натискати кнопку «Next >>», зверніть увагу, що в нижній частині вікна є **ще один чек-бокс**. Він називається **«Show dialog at startup»**. При виборі цього елемента діалог створення проекту буде автоматично запускатися кожен раз при запуску програми AVR Studio.

Для переходу до наступного етапу побудови проекту натисніть кнопку «Next >>». Вміст вікна побудовника зміниться. З'являться два великих поля під загальною назвою «Select debug platform and device» (Вибір налагоджувальної платформи і мікроконтролера). У списку Налагоджувальних платформ («Debug platform») перераховані всі налагоджувальні плати, які підтримує дана програма.

Ми не будемо використовувати зовнішніх плат, тому виберемо пункт «AVR Simulator» (Програмний імітатор AVR). В поле «Device» вибираємо потрібний тип мікросхеми. У нашому випадку це ATiny2313. Тепер всі налаштування закінчені. Для завершення процесу натисніть кнопку «Finish». Після натискання цієї кнопки програма створює проект і записує його в обрану вами директорію.

Відразу після створення новий проект складається всього з двох файлів:

- власне файл проекту Progl .aps;
- файл, куди буде поміщений текст програми на Ассемблері Progl .asm.

Файл тексту програми автоматично відкривається у вікні 3. Причому він поки абсолютно порожній. Тепер ви можете **приступати до набору цього тексту**. Якщо мова йде про програму Progl, то просто наберіть текст програми, розроблений для виконання лабораторної роботи, наприклад першої. При наборі тексту ви можете користуватися всіма можливостями, які зазвичай надає будь-який сучасний текстовий редактор.

Вбудований текстовий редактор програми AVR Studio підтримує всі необхідні сервісні функції:

- виділення текстових фрагментів;
- вирізання;

- копіювання;
- вставку;
- перетягування мишею;
- пошук і заміну, і багато іншого.

Для управління всіма цими можливостями використовується стандартний інтерфейс, знайомий вам по багатьом текстовим редакторам, зокрема, по популярному редактору Microsoft Word. Набраний тест програми не забудьте записати на диск за допомогою команди «Save» меню «File» або за допомогою відповідної кнопки на панелі інструментів (). Кнопка [] дозволяє записати відразу всі відкриті текстові файли.

Для програм, електронні версії яких вже маються в Інтернеті, проекти створювати не обов'язково. Досить завантажити файл з електронними версіями програм з потрібного сайту, наприклад http://book.microprocessor.by.ru, розпакувати архів і помістити його вміст в будь-яку відповідну директорію.

Наприклад, в директорію c: \ AVR \ myprog \. Після розпакування у вас з'явиться цілий набір директорій, в кожній з яких поміщений свій проект. Причому архів може містити не тільки проекти на Асемблері, але і на CI. Будь-який проект на Асемблері можна відкрити за допомогою пункту «Open Project» меню «Project».

2 Трансляція програми

2.1 Формати файлів

Після того, як текст програми набраний і записаний на жорсткий диск, необхідно провести трансляцію програми. В процесі трансляції створюється результуючий (об'єктний) файл, який представляє собою ту ж програму, але в машинних кодах, призначену для запису в програмну пам'ять мікроконтролера. Результуючий файл має розширення *hex*.

Крім *hex*-файлу транслятор створює ще кілька допоміжних файлів. І головне, файл з розширенням *eep*. Цей файл має точно таку ж внутрішню структуру, як файл *hex*. А містить він інформацію, призначену для запису в EEPROM. Така інформація з'являється в тому випадку, коли в тексті програми змінним, розміщеним в сегменті *eeprom*, присвоєні початкові значення. Якщо в лабораторних роботах цього не робити, то файл з розширенням *eep* у всіх проектах буде порожній (буде містити лише завершальний рядок).

Тепер трохи розберемося з форматом файлів *hex* і *eep*. В обох випадках застосовується так званий HEX-формат, який практично є стандартом для запису результатів транслювання різних програм. Він підтримується практично всіма трансляторами з будь-якої мови програмування.

В принципі, програмісту не обов'язково знати структуру цього формату. Досить розуміти, що в *hex*-файлі певним способом закодована програма в ма-

шинних кодах. Саме цей файл використовується програматором для «прошивки» програмної пам'яті мікроконтролера. Будь-який програматор підтримує *hex*-формат і розпізнає записані туди коди автоматично. Наведемо короткий опис *hex*-формату.

2.2 Формат HEX-файлу

Якщо ви подивитеся вміст такого файлу за допомогою редактора «Блокнот», то ви побачите, що це текстовий файл, в якому дані закодовані у вигляді текстових рядків. Нижче наведено приклад вмісту *hex*-файлу, отриманого в результаті трансляції одного з варіантів програми Progl:

```
: 020000020000FC
: 100000000FE70DBF00E806BD00E006BD01BB0FEF26
: 1000100007BB08BB02BB00E808B900B308BBFDCFB3
: 00000001FF
```

Як бачите, даний файл складається з чотирьох рядків. Перший і останній рядки несуть службову інформацію. Наявність першого рядка необов'язкова. Система AVR Studio при трансляції програми завжди додає в *hex*-файл перший рядок саме такого змісту. Останній рядок - це стандартний кінець для будь-якого *hex*-файлу.

Два рядки, що залишилися, якраз і містять інформацію про коди програми. У кожному такому рядку закодований ланцюжок байтів і адреса в пам'яті, де ці байти повинні розміщуватися.

Рядок починається з двокрапки. Двокрапка – обов'язковий елемент, який служить для ідентифікації *hex*-формату. Всі інші символи в рядку – це шістнадцяткові числа, записані злито без пробілів. Окремі числа відрізняють за їх позиції в рядку. Так, перші два знаки займає шістнадцяткове число, що означає довжину ланцюжка.

У нашому випадку довжина обох ланцюжків дорівнює 0x10 (тобто 16) байт. Наступні чотири символи – це початкова адреса, куди ці байти повинні бути поміщені. Перший ланцюжок буде розміщений в пам'яті, починаючи з нульової адреси. Другий ланцюжок - з адреси 0x0010. Чергові два знака займає код виду рядка. В рядках, що цікавлять нас, він дорівнює «00», що означає, що ці рядки призначені для запису даних (в першому рядку такий код дорівнює «02», а в останньому «01»).

Відразу після коду виду рядка починаються власне дані. Кожен байт даних займає два знаки. Найостанніші два символу – це контрольна сума. Вона розраховується за спеціальною формулою з використанням значень всіх байтів ланцюжка і служить для перевірки на відсутність помилок.

2.3 Процедура трансляції

Але **повернемося до процедури трансляції**. Для того, щоб запустити процес трансляції поточного проекту, потрібно вибрати в меню «Build» пункт, який теж називається «Build», або натиснути кнопку  . Тривалість процесу трансляції залежить від розмірів програми. Відразу ж після початку процесу вкладка «Build» у вікні 2 виходить на передній план. В процесі трансляції сюди виводяться службові повідомлення. До таких повідомленнями відносяться: повідомлення про завершення різних етапів трансляції, повідомлення про помилки (Error), а також попередження (Warning).

У готовій налагодженій програмі помилок і попереджень бути не повинно. Якщо програма виявить критичну помилку (Error), то процес трансляції буде призупинений, і результуючі файли створені не будуть. В цьому випадку необхідно усунути помилки і повторити трансляцію.

Транслятор не в змозі знайти всі види помилок. Він знаходить тільки явні помилки, які можна знайти автоматично. До таких помилок належать:

- помилки синтаксису (неправильне написання імені команди);
- неправильна кількість параметрів у оператора;
- спроба використання неописаних змінних і т. п.

Наприклад, повідомлення «Unknown instruction or macro» означає, що знайдена «Невідома інструкція або макрокоманда».

Попередження – це теж помилки, але не критичні. При виникненні некритичної помилки процес трансляції завершується як звичайно. Всі результуючі файли створюються в повному обсязі. Однак перш ніж зашивати таку програму в мікроконтролер, ретельно проаналізуйте повідомлення і постарайтеся визначити, як воно вплине на результати роботи. У будь-якому випадку, краще змінити програму таким чином, щоб усунути всі попередження.

Всі повідомлення у вкладці «Build» з'являються по мірі їх надходження. Для наочності кожне повідомлення позначено кольоровим кружечком на початку рядка:

- повідомлення про помилки позначаються червоним кружечком;
- попередження – жовтим кружечком;
- повідомлення про успішне виконання кожного чергового етапу трансляції позначаються зеленим кружечком.

Якщо повідомлення не вміщуються у вікно, то вони ховаються у верхній його частині. Однак, використовуючи смугу прокрутки, їх завжди можна переглянути. У разі успішного завершення процесу трансляції в якості останнього повідомлення виводиться статистична інформація (див. розділ 1.2).

Кожне повідомлення про помилку у вкладці «Build» містить точну вказівку місця в програмі, де сталася ця помилка. При цьому вказується

- ім'я файлу;
- номер рядка;
- фрагмент тексту програми, що містить помилку;
- її розшифровка.

Для того, щоб швидко перейти до фрагмента програми, який містить цю помилку, достатньо подвійного клацання по повідомленню про помилку. Вікно з текстом програми вийде на передній план, і в цьому вікні автоматично відобразиться потрібну ділянку тексту. На лівій межі вікна навпроти рядка, що містить помилку, ви побачите синю стрілочку – покажчик помилки.

Іноді програма невірно визначає місце, де виникла помилка. Це відбувається через недосконаліть аналізатора синтаксису. Справа в тому, що дуже складно розробити ідеальний алгоритм аналізу помилок. Якщо в якому-небудь рядку транслятор показує помилку, а ви помилок не спостерігаєте, подивіться на попередні рядки. Можливо, помилка десь там.

3 Завдання для самостійної роботи

3.1 Вивчити етапи створення нового проекту на основі Ассемблера (Atmel AVR Assembler);

3.2 Вивчити формати *hex* і *eep* - файлів;

3.3 Вивчити особливості процедури трансляції

4 Завдання до лабораторної роботи

4.1 Виконати зазначені в розділі 1 методички операції по створенню нового проекту на основі Ассемблера (Atmel AVR Assembler). Підтвердити роздруківками;

4.2 Виконати зазначені в розділах 2.1, 2.2 методички операції по вивчений-нію форматів *hex* і *eep* - файлів. Підтвердити іллюстраціями;

4.3 Виконати зазначені в розділі 2.3 методички операції з вивчення особливостей процедури трансляції. Підтвердити іллюстраціями

5 Зміст звіту

5.1 Мета роботи;

5.2 Підсумки виконання пунктів 4.1-4.3;

5.3 Висновки.

6 Контрольні питання

6.1 Які варіанти виконання проекту пропонує програма?

6.2 Які режими визначають перший і другий чек-бокси?

6.3 Як описати функції поля «Location» ?

6.4 Які режими визначає чек-бокс «Show dialog at startup»?
6.5 З яких файлів складається новий проект після свого створення?
6.6 Які сервісні функції підтримує вбудований текстовий редактор програми «AVR Studio»?
6.7 Які об'єктні файли створює транслятор?
6.8 Як описати процедуру трансляції програми ?
6.9 Які існують помилки та попередження в процесі трансляції ?

Лабораторна робота №4

Налагодження розроблюваної програми

Мета роботи – вивчити процес налагодження розроблюваної програми, включаючи покроковий метод і метод застосування точок зупину

В результаті проведення лабораторної роботи студенти повинні
знати:

- можливі помилки алгоритму та його реалізації;
- етапи процесу налагодження;
- покроковий метод налагодження;
- метод застосування точок зупину;
- роботу з вікном перегляду та зміни вмісту введених змінних;
- варіанти виправлення помилок за допомогою стандартних засобів редагування

вміти:

- виконувати покрокове налагодження програми;
- проводити налагодження програми методом застосування точок зупину.

1 Помилки алгоритму та його реалізації

Якщо всі помилки виправлені і відсутні попередження, то це означає, що програма успішно відтрансльована. В принципі, її можна записувати в програмну пам'ять і випробувати її роботу «в залізі». Але в більшості випадків відсутність синтаксичних помилок ще не означає відсутність помилок як таких. Можна написати команду правильно, але не ту. Але головна неприємність – помилки алгоритму або його реалізації.

Програміст може упустити який-небудь крок чи неправильно поставити умови. Всі можливі помилки алгоритму перерахувати складно. Але в результаті програма може працювати неправильно або зовсім не працювати. З цієї причини перед тим, як записувати програму в програмну пам'ять мікроконтролера, необхідно спробувати виявити всі ці помилки.

Взагалі, процес написання програми відсотків на 60-70 складається з пошуку та усунення помилок. І основна кількість помилок виявляється при налагодженні програми. По англійськи процес налагодження називається Debug – процес позбавлення від помилок (дослівно - процес лову блох).

2 Етапи процесу налагодження

Процес налагодження починається з перекладу програми у відповідний

режим. Якщо проект відкритий, а всі його програми записані і оттрансльовані, то для переходу в режим налагодження виберіть пункт «Start Debugging» в меню «Debug» або натисніть кнопку  на панелі завдань.

Програма почне **процес підготовки**. Процес тривалий. Поки йде підготовка, у нижній частині основної панелі рухатиметься смуга, що показує відсоток виконання операції. По закінченні процесу підготовки програма переходить **в новий режим**. У вікні 1 на передній план виходить вкладка «I / O View» (див. рис. 2.1), яка тепер буде використовуватися для перегляду вмісту всіх регістрів. Причому зовнішній вигляд цієї вкладки трохи змінюється. Для кожного елемента в дереві ресурсів з'являється поле, що відображає його вміст.

У вікні 2 на передній план виходить вкладка «Breakpoints and Tracepoints», де тепер будуть відображатися всі точки зупину. В панелі інструментів активізуються всі інструменти, які стосуються режиму відладки (до цього вони були неактивні). У вікні 3 на перший план виходить текст головного програмного файлу. На лівій межі вікна цього файлу з'являється жовта стрілка - покажчик поточної виконуваної команди. Покажчик встановиться в початок програми (навпроти першої виконуваної команди). Тепер все готово для налагодження.

Налагодження може виконуватися різними методами. Найпростіший метод – покрокове виконання. Для того, щоб зробити один крок, виберіть у меню «Debug» пункт «Step into» («Крок в») або натисніть кнопку  на панелі інструментів.

Можна також просто натиснути кнопку «F11». В результаті програма виконає одну поточну команду. Покажчик поточної команди (жовта стрілка) переміститься в наступну позицію. Вміст регістрів зміниться відповідно до виконаної операції.

Ви можете це перевірити, знайшовши потрібний регістр у вікні 1. Переконавшись, що команда виконана правильно, робіть наступний крок. І так далі. При цьому ви можете простежити послідовність виконання операцій, правильність виконання умовних переходів і багато іншого.

У будь-який момент ви можете **вручну змінити** вміст будь-якого з елементів в дереві ресурсів. Причому, можна змінювати як вміст будь-якого окремого розряду, так і всього регістра в цілому. Для зміни вмісту розряду досить клацнути за допомогою миші по одному з квадратиків, що символізує потрібний розряд (див. рис. 4.1).

При цьому стан квадратика зміниться на протилежне (одиниця зміниться на нуль, або навпаки). Для зміни значення всього регістра необхідно провести подвійне клацання мишею по зображенню вмісту регістра (в шістнадцятковому вигляді). Відкриється вікно вмісту.

У цьому вікні ви можете вибрати одну з чотирьох форм представлення числа (шістнадцяткове, десяткове, вісімкове або двійкове) і змінити це значення в вибраному вами форматі. Потім натисніть кнопку «Ok» і зміна «запишеться» у відповідний регістр.

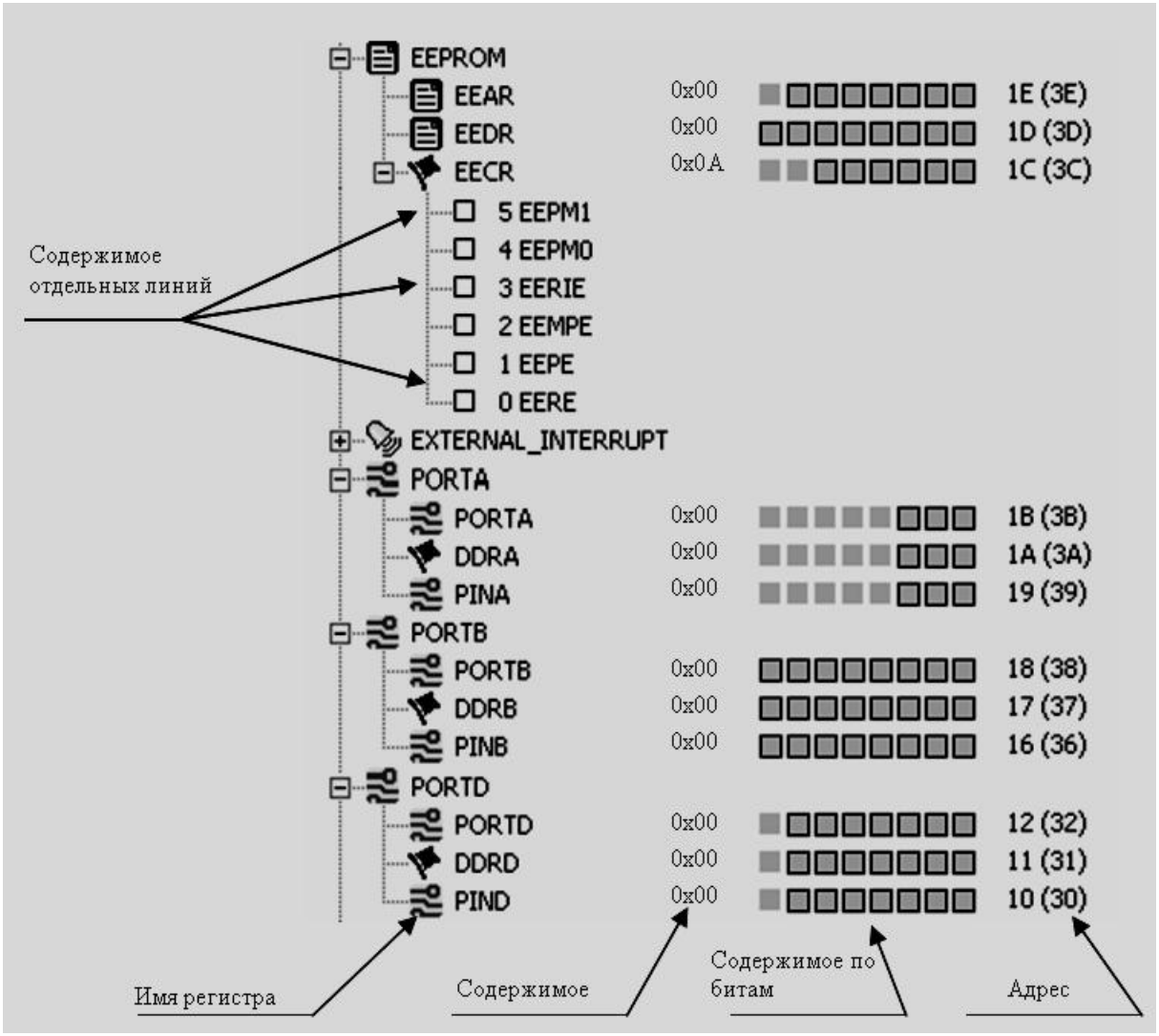

Рисунок 4.1 – Зміна вмісту регістрів

Змінюючи вміст регістру, ви можете моделювати різні ситуації. Наприклад, імітувати зміну сигналів на вході порту або примусово змінювати значення лічильного регістру таймера, щоби не чекати, поки він дорахує до потрібного значення.

Крім директиви «Крок в», є ще кілька її модифікацій. Їх призначення та способи виклику наведені в табл. 4.1.

Директива «Крок через» використовується в тому випадку, якщо при покроковому виконанні програми зустрінеться команда виклику підпрограми. Якщо ви не хочете покроково виконувати всю підпрограму, ви можете виконати її за один крок. При цьому бажано, щоб підпрограма не містила помилок.

Директива «Крок з» застосовується в тому випадку, якщо ви все ж увійшли в підпрограму, але потім зрозуміли, що її покрокове виконання зайве.

Взявши дану директиву, можна за один крок виконати всі команди підпрограми, що залишилися.

Таблиця 4.1 – Директиви покрокового виконання програми

Назва	**Пункт меню “Debug”**	**Кнопка**	**Гаряча клавиша**	**Опис**
Крок в	Step into		F11	Виконати чергову команду
Крок через	Step over		F10	Виконати чергову команду
Крок з	Step out		Shift+F11	Завершити поточну підпрограму
Виконати до	Run to cursor		Ctrl+F10	Виконувати з поточного рядка і до рядка, де стоїть курсор

Директива «Виконати до» застосовується в тому випадку, коли будь-яка частина програми не оформлена у вигляді підпрограми, але її бажано виконати за один крок. В цьому випадку в кінці вибраного фрагменту ви можете встановити текстовий курсор (миготливу вертикальну смужку) і вибрати директиву «Виконати до». Налагоджувач за один крок виконає всі команди, починаючи з поточної (відміченої жовтою стрілкою) і аж до текстового курсора. Команда в рядку з курсором виконуватися не буде. Вона стане поточною (на неї тепер буде вказувати жовта стрілка).

3 Застосування точок зупину

Покроковий метод налагодження зручний для налагодження невеликих нескладних програм або окремих ділянок великої програми. Але уявіть собі, що ваша програма містить цикл, який повиний бути виконаний велику кількість разів. Для того, щоб перевірити правильність виконання всього цього циклу в покроковому режимі, вам довелося б дуже довго клацати мишкою. У подібних випадках застосовуються точки зупину (Breakpoint).

Визначення. **Точка зупину** – *це спеціальна мітка, яку в налагоджувальному режимі програміст може поставити проти будь-якого рядка програми.*

Потім програма запускається під управлінням налагоджувача. Але це – не реальна робота. Це лише імітація роботи мікроконтролера. Програма виконується рядок за рядком, поки в черговому рядку не зустрінеться точка зупину. Виявивши таку точку, налагоджувач призупиняє виконання програми.

Виглядає це таким чином, начебто за один крок ви виконали великий шматок програми. Тепер ви можете знову переглянути і змінити вміст будь-якого регістру. А потім продовжити налагодження. Причому, ви можете про-

довжити її як в покроковому режимі, так і запустити програму в режимі автоматичного виконання до наступної точки зупину.

Для управління точками зупину програма має кілька вбудованих директив, які показані в таблиці 4.2.

Таблиця 4.2 – Директиви управління точками зупину

Назва	**Пункт меню “Debug”**	**Кнопка**	**Гаряча клавиша**	**Опис**
Поставити точку зупину	Toggle Breakpoint		F9	Поставити (зняти) точку зупину в рядку, де знаходиться курсор
Прибрати всі точки зупину	Remove all Breakpoints		-	Прибрати всі поставлені раніше точки зупину
Створити програмну точку зупину	New Breakpoints/ Program Breakpoint	-	-	Створити точку зупину шляхом завдання програмної умови
Створити точку зупину за даними	New Breakpoints/ Data Breakpoint	-	-	Створити точку зупину шляхом завдання умови за даними

Для того, **щоб поставити точку зупину** у якому-небудь рядку програми, потрібно спочатку помістити в цей рядок текстовий курсор. Потім вибрати директиву «Поставити крапку зупину» (див. Табл. 4.2). Точка зупину виглядає як коричневий кружечок навпроти вибраного рядку програми на лівій межі текстового вікна.

Якщо помістити курсор в рядок, де вже є точка зупину, і виконати ще раз директиву «Поставити точку зупину», то точка прибирається. Прибрати відразу всі поставлені точки зупину можна за допомогою директиви «Прибрати всі точки зупину».

Другий спосіб проставляння точок зупину – завдання їх через меню. Призначений для цього пункт «New Breakpoints» меню «Debug» має два підпункти. За допомогою підпункту «Program Breakpoint» можна встановлювати програмні точки зупину. Тобто точно такі, які ми ставили попереднім способом.

Відмінність способу постановки точок через меню в тому, що їх місце розташування в програмі ви визначаєте шляхом заповнення полів у спеціальній формі. У цій формі, окрім номера рядку або адреси програми, де ви хочете поставити точку зупину, ви можете вказати кількість проходів.

Для цього вам необхідно заповнити поле «Break execution after: - hits» («Зупинити виконання після: - проходів»). Якщо число в цьому полі не дорівнює нулю, то програма зупиниться в даній точці зупину не з першого разу, а лише тоді, коли пройде через неї вказану кількість разів.

Якщо ви встановили вашу точку зупину не через меню, а напряму в тексті програми, ви все одно можете викликати описаний вище діалог і змінити в ньому кількість проходів, клацнувши; мишею по рядку з описом потрібної точки зупину у вкладці «Breakpoints and Tracepoints».

За допомогою підпункту «Data Breakpoint» пункту «New Breakpoints» меню «Debug» можна задавати точки зупину за даними.

При виборі цього пункту меню відкривається діалог, в якому ви можете вибрати будь-яку із змінних вашої програми або будь-який ресурс мікроконтролера (зі списку що відкривається) і поставити точку зупину за зверненням до цієї змінної (ресурсу).

Програма дозволяє вибрати цілий ряд умов, при яких настане останов програми. За замовчуванням останов відбувається при будь-якому зверненні до цієї змінної як в режимі читання, так і в режимі запису. Ви можете вибрати іншу умову. Наприклад, при рівності змінної певному значенню. Вибір умови проводиться за допомогою поля «Break when:» («Зупинитися якщо:») і поля «Access type:» («Тип доступу»). Ім'я змінної вибирається за допомогою поля «Location».

Діалог проставлення точок зупину обох видів можна викликати не тільки через меню. У верхній лівій частині вкладки «Breakpoints and Tracepoints» для цього є спеціальна кнопка.

Таблиця 4.3 – Директиви управління процесом налагодження

Назва	**Пункт меню “Debug”**	**Кнопка**	**Гаряча клавиша**	**Опис**
Запустити	Run		F5	Запуск автоматичного виконання програми з поточної команди
Зупинити	Break		Ctrl+F5	Зупинка автоматичного виконання програми
Скидання	Reset		Shift+F5	Початковий стан (скидання мікроконтролера)
Закінчити налагодження	Stop Debugging		Ctrl+Shuft+F5	Закінчити налагодження

Після того, як ви проставили все точки зупину, ви можете запускати програму в режимі автоматичного виконання. Для управління налагоджувачем в цьому режимі програма AVR Studio також має декілька спеціальних директив

(див. Табл. 4.3). Запуск автоматичного виконання програми проводиться за допомогою директиви «Пуск».

Поки програма знаходиться в режимі автоматичного виконання, новий стан регістрів не відображається. Покажчик поточної команди також відсутній. В нижньому рядку головної панелі програми в правій її стороні знаходиться індикатор стану. У режимі останова це жовтий кружечок з рискою посередині. Зліва від нього знаходиться слово «Stopped» (Зупинено). В режимі автоматичного виконання програми жовтий кружечок перетворюється в зелений з хрестиком всередині. Замість слова «Stopped» з'являється слово «Running» (Запущений).

Якщо ви неправильно поставили точку зупину або забули її поставить, програма буде знаходитися в режимі автоматичного виконання нескінченно довго. Для дострокової зупинки програми використовується директива «Зупинити». Якщо в процесі налагодження програми знадобиться почати все спочатку (імітувати скидання мікроконтролера), це можна зробити за допомогою директиви «Скидання». Після завершення налагодження програми необхідно перейти в режим редагування. Для цього служить директива «Закінчити налагодження».

4 Перегляд і зміна вмісту введених змінних

Для оперативного перегляду та зміни вмісту введених користувачем змінних в процесі налагодження можна відкрити спеціальне вікно. Для цього достатньо вибрати пункт Watch в меню View. Вікно має чотири вкладки. Тому можна мати чотири різних набору змінних.

Для того, щоб включити яку-небудь змінну в поточне вікно Watch, необхідно встановити курсор миші на ім'я цієї змінної в тексті програми і натиснути праву кнопку миші. Припустимо, ви встановили курсор на змінну temp. Тоді в меню, ви побачите пункт Add Watch: «temp». Виберіть цей пункт, і змінна буде включена в список Watch.

Точно так само можна оперативно переглядати вміст усіх носів. Для цього виберіть пункт «Memory» в меню «View». Відкриється нове вікно під назвою «Memory». За умовчанням в цьому вікні у вигляді дампа буде подано вміст програмної пам'яті.

За допомогою списку, що випадає в лівій верхній частині цього вікна, можна вибрати інший вид пам'яті – пам'ять даних (Data), EEPROM або навіть вміст RAM або портів вода / виводу. У процесі налагодження ви завжди будете бачити в цьому вікні всі зміни вибраної частини пам'яті. Якщо ви бажаєте бачити одночасно вміст відразу декількох видів пам'яті, то ви можете відкрити друге і навіть третє подібне вікно. Для цього виберіть пункт «Memory2» або «Memory3» в меню «View».

5 Виправлення помилок

Таблиця 4.4 – Директиви роботи з закладками

Назва	Пункт меню “Debug”	Кнопка	Гаряча клавиша	Опис
Поставити закладку	Toggle Bookmark		Ctrl+F2	Поставити (зняти) закладку в рядку, де знаходиться курсор
Поставити до наступної закладки	-		F2	Перемістити курсор до наступного рядка із закладкою
Перейти до попередньої закладки	-		Shift+F2	Перемістити курсор до попереднього рядка із закладкою
Прибрати всі закладки	Remove Bookmarks		Ctrl+Shift+F2	Видалити всі поставлені раніше закладки

У тому випадку, якщо ви захочете доопрацювати програму або написати нову, вам доведеться багато разів переписувати її, шукати різні фрагменти, замінювати їх на інші і т. п. Редактор програми AVR Studio дає повний спектр стандартних засобів редагування. Одне з таких засобів – це проставлення закладок. Поставивши закладку в будь-якому місці в тексті програми, ви можете спокійно гортати цей текст далі. У разі необхідності ви можете в будь-який момент повернутися до закладки.

В табл. 4.4 приведені всі директиви роботи із закладками.

Для створення нової закладки потрібно встановити в потрібному рядку текстів курсор і вибрати директиву «Поставити закладку». При повторному виклику цієї директиви в тому ж рядку, закладка прибирається. Поставивши декілька закладок, можна пересуватися по них за допомогою директив «Перейти до наступної закладки» і «Перейти до попередньої закладки». З допомогою відповідної директиви можна прибрати всі закладки.

6 Завдання для самостійної роботи

6.1 Вивчити можливі помилки алгоритму та його реалізації;
6.2 Вивчити етапи процесу налагодження;
6.3 Вивчити особливості покрокового методу налагодження;
6.4 Вивчити особливості налагодження методом застосування точок зупину;
6.5 Вивчити варіанти виправлення помилок за допомогою стандартних засобів редагування

7 Завдання до лабораторної роботи

7.1 Запротоколювати етапи процесу налагодження;
7.2 Запротоколювати особливості покрокового методу налагодження;
7.3 Запротоколювати особливості налагодження методом застосування точок зупину;
7.4 Запротоколювати варіанти виправлення помилок за допомогою стандартних засобів редагування;

8 Зміст звіту

8.1 Мета роботи;
8.2 Підсумки виконання пунктів 1-5;
8.3 Висновки.

9 Контрольні питання

9.1 Які існують етапи процесу налагодження програми?
9.2 У чому полягають особливості покрокового методу налагодження?
9.3 Які існують особливості налагодження методом застосування точок зупину?
9.4 Які існують варіанти виправлення помилок за допомогою стандартних засобів редагування?

Лабораторна робота №5

Мікропроцесорне управління світлодіодним індикатором в найпростішому режимі комутації

Мета роботи – отримати навички розробки найпростіших програм шляхом розробки мікропроцесорного пристрою управління одним світлодіодним індикатором за допомогою однієї кнопки. При натисканні кнопки світлодіод повинен запалитися, при відпуску – згаснути

В результаті проведення лабораторної роботи студенти повинні
знати:

- особливості роботи мікроконтролера ATiny2313;
- функціонування досліджуваної принципової електричної схеми;
- алгоритм роботи мікропроцесорної системи;
- загальні принципи побудови програм на асемблері;
- використовувану версію AVR-асемблера

вміти:

- складати найпростіші програми на AVR-ассемблере;
-користуватися програмним комплексом «AVR Studio» фірми Atmel

1 Принципова електрична схема

З практичної точки зору розроблювальний пристрій має лише методичне значення. Однак рішення подібних задач дозволить надалі перейти до побудови реальних мікропроцесорних систем.

Розглянемо особливості роботи принципової електричної схеми, здатної виконати описану вище задачу.

Згідно із завданням, до мікроконтролера потрібно підключити світлодіод і кнопку управління. Для підключення до мікроконтролера AVR будь-яких зовнішніх пристроїв використовуються порти введення-виведення. Причому кожен такий порт здатний працювати або на введення, або на виведення.

Найзручніше світлодіод підключити до одного з портів, а кнопку – до іншого. При цьому керуюча програма повинна буде налаштувати порт, до якого підключений світлодіод, на виведення, а порт, до якого підключена кнопка, на введення. Інших особливих вимог до микроконтроллеру немає.

Очевидно, що необхідний мікроконтролер, який має не менше двох портів. Даним умовам задовольняють багато мікроконтролери AVR, в тому числі і мікроконтролер ATiny2313. Ця мікросхема відноситься до сімейства «Tiny» і займає якесь проміжне місце між сімейством «Tiny» і сімейством «Mega».

Ця мікросхема містить два основних і один додатковий порт вводу-виводу, має як восьмирозрядний, так і шістнадцятирозрядний таймер / лічильник. Має оптимальні розміри (20-вивідний корпус).

Мікроконтролер має порт A, який включається тільки в особливому режимі і буде розглядатися в подальшому, і два основних порти вводу-виводу (порт B і порт D). Для управління світлодіодом ми будемо використовувати молодший розряд порту B (лінія PB.0), а для зчитування інформації з кнопки управління використовуємо молодший розряд порту D (лінія PD.0). Повна схема пристрою, що дозволяє вирішити поставлену задачу, приведена на рис. 5.1.

Для підключення кнопки S1 використана класична схема. У початковому стані контакти кнопки розімкнуті. Через резистор R1 на вхід PD.0 мікроконтролера подається позитивна напруги живлення, що відповідає сигналу логічної одиниці.

При замиканні кнопки напруга падає до нуля, що відповідає логічному нулю. Таким чином, зчитуючи значення сигналу на відповідному виведенні порту, програма може визначати момент натискання кнопки.

Незважаючи на простоту даної схеми, мікроконтролер AVR дозволяє її ще більш спростити. Резистор R1 можна виключити, замінивши його внутрішнім навантажувальним резистором мікроконтролера. Справа в тому, що мікроконтролери серії AVR мають вбудовані навантажувальні резистори для кожного розряду порту. Головне при написанні програми – не забути включити програмним шляхом відповідний резистор.

Підключення світлодіода також виконане за класичною схемою. Це – безпосереднє підключення до виходу порту. Всі виходи мікроконтролера розраховані на безпосереднє управління світлодіодом з струмом споживання до 20 мА. У ланцюг світлодіода включений струмообмежуючий резистор R3.

Для того, щоб запалити світлодіод, мікроконтролер повинен подати на вивід PB.0 сигнал логічного нуля. В цьому випадку напруга, прикладена до ланцюжка R2, VD1, виявиться рівною напрузі живлення, що викличе струм через світлодіод, і він загориться. Якщо ж на вивід PD.0 подати сигнал логічної одиниці, падіння напруги на світлодіоді і резисторі виявиться рівним нулю, і світлодіод згасне.

Крім ланцюга підключення кнопки і ланцюга управління світлодіодом, на схемі є ще кілька ланцюгів. Це стандартні ланцюги, що забезпечують нормальну роботу мікроконтролера. Кварцовий резонатор Q1 забезпечує роботу вбудованого тактового генератора. Конденсатори C2 і C3 – це ланцюги узгодження кварцового резонатора.

Елементи Cl, R2 – це стандартний ланцюг початкового скидання. Такий ланцюг забезпечує скидання мікроконтролера в момент включення живлення. Раніше подібний ланцюг був обов'язковим атрибутом будь-якої мікропроцесорної системи. Однак сучасні мікроелектронні технології дозволяють виключити обидві ці ланцюги (зовнішній кварц і ланцюг початкового скидання).

Справа в тому, що більшість мікроконтролерів AVR, крім тактового генератора із зовнішнім кварцовим резонатором, містять внутрішній RC-генератор,

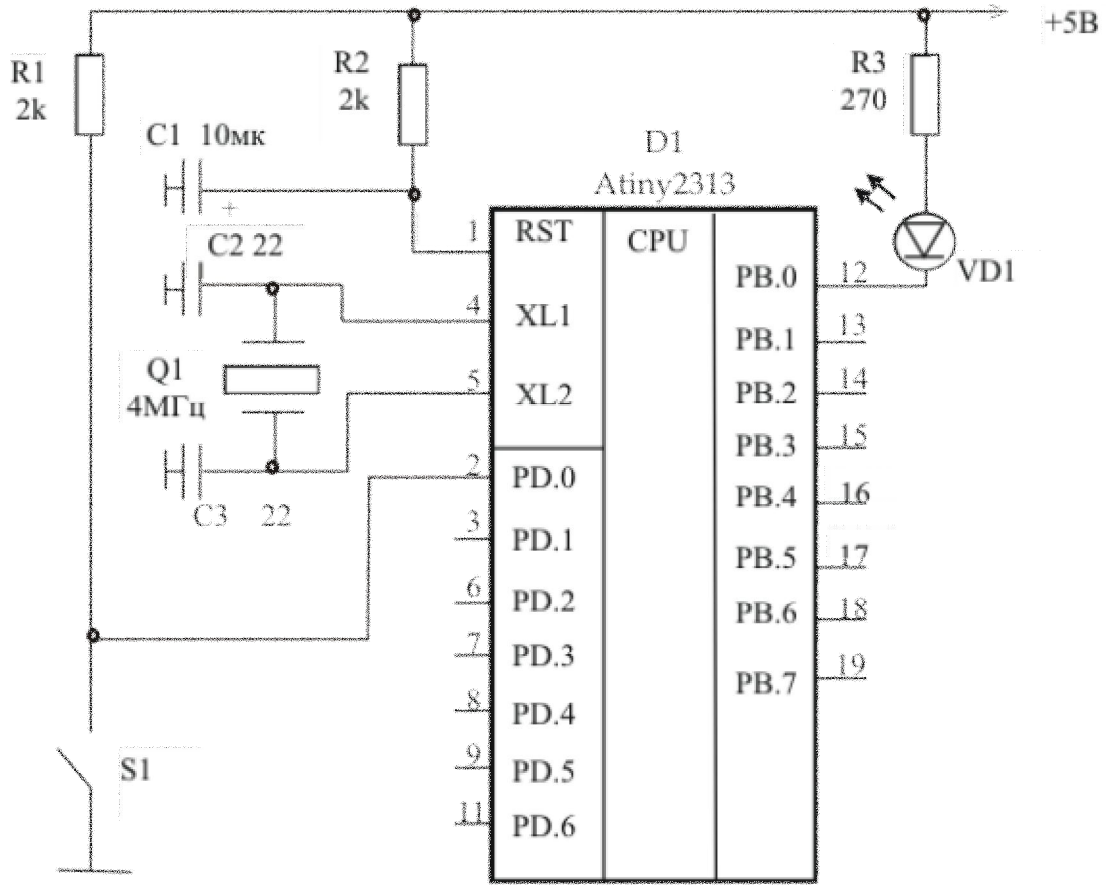

Рисунок 5.1 – Принципова схема мікроконтролерного пристрою з одним світлодіодом і однією кнопкою

який не потребує ніяких зовнішніх ланцюгів. Якщо не пред'являються високі вимоги до точності і стабільності частоти генератора, що задає, то мікросхему можна перевести в режим внутрішнього RC-генератора і відмовитися як від зовнішнього кварцу (Q1), так і від конденсаторів узгодження (С2 і С3).

Ланцюг початкового скидання теж можна виключити. Будь-який мікроконтролер AVR має внутрішню систему скидання, яка в більшості випадків забезпечує стабільне скидання при включенні живлення. Зовнішні ланцюги скидання застосовуються тільки за наявності особливих вимог до тривалості імпульсу скидання. А це буває лише в тих випадках, коли мікроконтролер працює в умовах великих перешкод і нестабільного живлення.

Всі описані вище перемикання виробляються за допомогою відповідних fuse-перемикачів. Робота з ними показана в [1]. Тоді три звільнених виводів мікроконтролера можуть бути використані як додатковий порт (порт А). Але в даному випадку в цьому немає необхідності.

Спростимо схему, показану на рис.5.1, з урахуванням описаних вище можливостей. Від зовнішнього кварцу поки відмовлятися не будемо. Він буде необхідний при виконанні лабораторних робіт, що вимагають формування часових інтервалів. Удосконалена схема зображена на рис. 5.2.

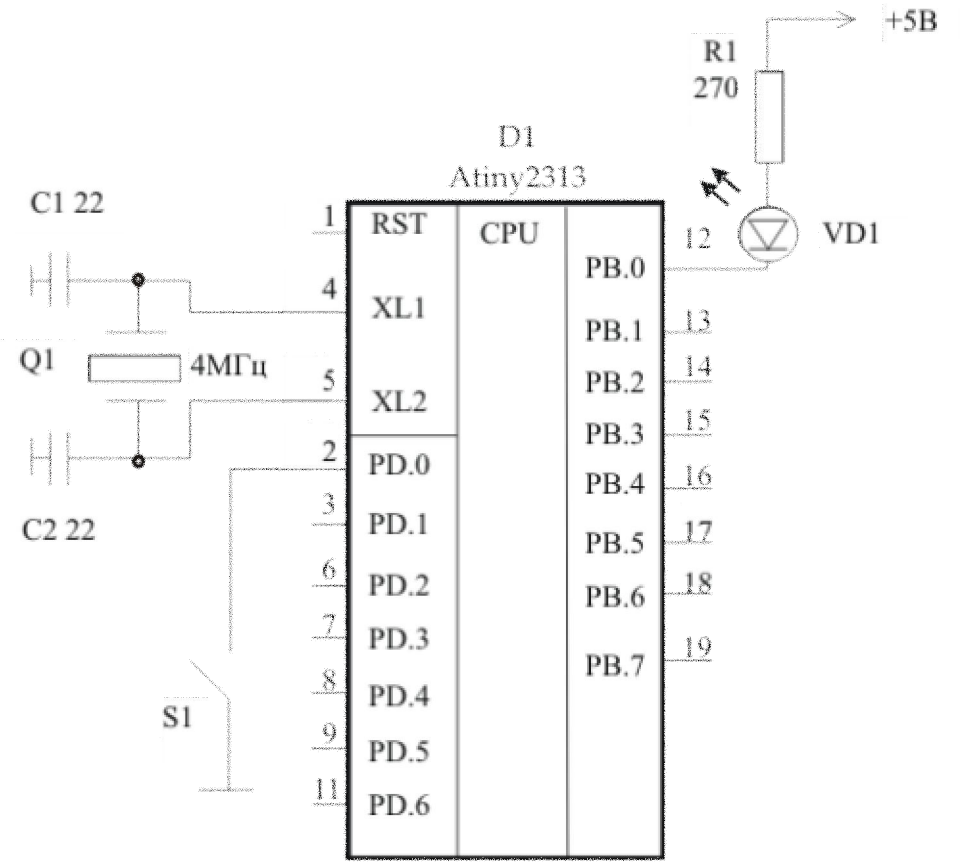

Рисунок 5.2 – Удосконалена схема для лабораторної роботи

2 Алгоритм

Отже, зі схемою ми визначилися. Тепер потрібно приступити до розробки програми. Розробка будь-якої програми починається з розробки алгоритму.

Визначення. Алгоритм – *це послідовність дій, яку повинен виробити наш мікроконтролер, щоб досягти необхідного результату. Для простих задач алгоритм можна просто описати словами. Для більш складних задач алгоритм малюється в графічному вигляді.*

В нашому випадку алгоритм такий: Після операцій початкового налагодження портів мікроконтролер повинен увійти в безперервний цикл, в процесі якого він повиний опитувати вхід, підключений до нашої кнопки, і залежно від її стану керувати світлодіодом. Опишемо це докладніше.

Операції початкового налаштування:

- встановити початкове значення для вершини стека мікроконтролера;
- налаштувати порт B на вивід інформації;
- подати на вихід PB.0 сигнал логічної одиниці (згасити светодіод);
- конфігурувати порт D на введення;
- включити внутрішні навантажувальні резистори порту D.

Операції, що становлять тіло циклу:

- прочитати стан молодшого розряду порту PD (PD.0);
- якщо значення цього розряду дорівнює одиниці, вимкнути світлодіод;
- якщо значення розряду PD.0 дорівнює нулю, включити світлодіод;
- перейти на початок циклу.

3 Програма на Ассемблері

Для створення програм ми використовуємо версію Ассемблера, запропоновану розробником мікроконтролерів AVR – фірмою Atmel. А також скористаємося програмним комплексом «AVR Studio», розробленим тією ж фірмою і призначеним для створення, редагування, трансляції і налагодження програм для AVR на Асемблері. Детальніше програма «AVR Studio» описана в попередньому розділі.

Головна мета лабораторних робіт - навчитися створювати програми. Вивчення мови програмування буде відбуватися таким чином. В методичному посібнику наводиться приблизний варіант тексту програми для кожного конкретного завдання, а потім докладно описуються всі його елементи і пояснюється, як програма працює. Студент повинен скласти свій варіант програми.

Текст можливого варіанту програми, який реалізує поставлену вище задачу, приведений в лістингу 5.1. Перш ніж приступити до опису даного прикладу, необхідно нагадати кілька загальних понять про мову Асемблер.

Програма на Асемблері являє собою набір команд і коментарів (іноді команди називають інструкціями). Кожна команда займає один окремий рядок. Їх допускається перемежати порожніми рядками. Команда обов'язково містить оператор, який виглядає як ім'я виконуваної операції.

Деякі команди складаються тільки з одного оператора. Інші ж команди мають один або два операнди (параметри). Операнди записуються в тому ж рядку відразу після оператора, через пропуск. Якщо операндів два, їх записують через кому. Так, в рядку 6 нашої програми записана команда завантаження константи в регістр загального призначення. Вона складається з оператора ldi і двох операндів – temp і RAMEND.

У разі необхідності перед командою допускається ставити мітку. Вона складається з імені мітки, що закінчується двокрапкою. Мітка служить для іменування цього рядка програми. Потім це ім'я використовується в різних командах для звернення до поміченого рядку.

При виборі імені мітки необхідно дотримуватися таких правил:

- ім'я має складатися з одного слова, який містить тільки латинські букви і цифри;
- допускається також застосовувати символ підкреслення;
- першим символом мітки обов'язково повинна бути буква або символ підкреслення.

Рядок 16 нашої програми містить мітку з іменем main. Мітка не обов'язково повинна стояти в рядку з оператором. Допускається ставити мітку в будь-якому рядку програми. Крім команд і міток, програма містить коментарі.

Визначення. Коментар – *це спеціальний запис в тілі програми, призначена для людини. Комп'ютер в процесі трансляції програми ігнорує всі коментарі. Коментар може займати окремий рядок, а може стояти в тому ж рядку, що й команда. Починається коментар з символу «крапка з комою». Все, що знаходиться після крапки з комою до кінця поточного рядка програми, вважається коментарем.*

Якщо у вже готовій програмі ви поставите крапку з комою на початку рядка перед якою-небудь командою, то даний рядок для транслятора як би зникне. З цього моменту транслятор буде вважати увесь цей рядок коментарем. Таким чином, можна тимчасово відключати окремі рядки програми в процесі налагодження (тобто при пошуку помилок у програмі).

Крім операторів, в мові Асемблер застосовуються псевдооператори, або директиви. Якщо оператор – це якийсь еквівалент реальної команди мікроконтролера і в процесі трансляції замінюється відповідним машинним кодом, який поміщається в файл результату трансляції, то директива, хоча за формою і нагадує оператор, але не є командою процесора.

Визначення. Директиви – *це спеціальні допоміжні команди для транслятора, що визначають режими трансляції та реалізують різні допоміжні функції.*

Далі з конкретних прикладів можна зрозуміти, про що йде мова. В даній конкретній версії Ассемблера директиви виділяються особливим чином. Ім'я кожної директиви починається з точки. (Дивись лістинг 5.1, рядки з 1 по 5).
При написанні програм на Асемблері прийнято дотримуватися особливої форми запису:

- програма записується в кілька колонок (див. Лістинг 5.1);
- аналогічні елементи різних команд прийнято розміщувати один під одним;
- найперша (ліва) колонка зарезервована для міток;
- якщо мітка відсутня, місце в колонці пустує;
- наступна колонка призначена для запису операторів;
- потім йде колонка для операндів;
- залишився простір (крайня колонка справа) призначене для коментарів.

В деяких випадках, наприклад, коли текст команди дуже довгий, допускається порушувати цей порядок. Але, по можливості, потрібно оформляти програму саме так. Оформлена подібним чином програма більш наочна і набагато краще читається.

Після розгляду загальних принципів побудови програми на Асемблері можна приступити до докладного опису конкретної програми, наведеної в лістингу 5.1. Почнемо з опису команд, що входять до неї.

4 Директиви

.include
Приєднання до поточного тексту програми іншого програмного тексту.

Такий прийом використовується практично у всіх існуючих мовах програмування. При складанні програм часто буває так, що в абсолютно різних програмах доводиться застосовувати абсолютно однакові програмні фрагменти. Для того, щоб не переписувати ці фрагменти з програми в програму, їх прийнято оформляти у вигляді окремого файлу з таким розрахунком, щоб цей файл могли використовувати всі програми, де цей фрагмент буде потрібний.

У мові Асемблер для приєднання фрагменту до програми використовується псевдооператор .include. У якості параметра для цієї директиви має бути вказане ім'я файлу, що приєднується. Якщо такий оператор поставити в будь-якому місці програми, то фрагмент, що міститься в файлі, що приєднується, в процесі трансляції як би вставляється в те саме місце, де знаходиться оператор. Наприклад, у програмі на лістингу 5.1 в рядку 1 в основний текст програми вставляється текст з файлу tn2313def. inc.

Файл tn2313def. inc - *це файл описів. Він містить опис всіх регістрів і деяких інших параметрів мікроконтролера ATiny2313. Цей опис знадобиться нам для того, щоб в програмі ми могли звертатися до кожного регістру по його імені. Про те, як робляться такі описи, ми поговоримо при розгляді конкретних програм.*

.list
Включення генерації лістингу.

В даному випадку **лістинг** – *це спеціальний файл, в якому відбивається весь хід трансляції програми.* Такий лістинг повторює весь текст вашої програми, включаючи всі приєднані фрагменти. Проти кожного рядка програми, яка містить реальну команду, поміщуються відповідні їй машинні коди. Там же показуються всі знайдені в процесі трансляції помилки. За замовчуванням лістинг не формується. Якщо вам потрібний лістинг, включіть цю команду в вашу програму.

.def
Макровизначення.

Ця команда дозволяє присвоювати різним регістрам мікроконтролера будь-які осмислені імена, що спрощують читання і розуміння тексту програми. В нашому випадку нам знадобиться один регістр для тимчасового зберігання різних величин. Виберемо для цієї мети регістр r16 і присвоїмо йому найменування temp від англійського слова temporary – тимчасовий.

Дана команда виконується в рядку 3 (див. Лістинг 5.1). Тепер в будь-якому місці програми замість імені r16 можна застосовувати ім'я temp. Це необхідно для наочності і читаності програми. У даній програмі ми будемо використовувати лише один регістр, і переваги такого перейменування тут не дуже видно. Але уявіть, що ви використовуєте безліч різних регістрів для зберігання самих різних величин. В цьому випадку привласнення осмисленого імені дуже полегшує програмування. До речі, саме таким чином визначено імена всіх стандартних регістрів в файлі tn2313def. inc.

.cseg

Псевдооператор вибору програмного сегменту пам'яті.

Мікроконтролер для зберігання даних має три види пам'яті: пам'ять програм (Flash), оперативну пам'ять (SRAM) і енергонезалежну пам'ять даних (EEPROM). Програма на Ассемблері повинна працювати з будь-яким з цих трьох видів пам'яті. Для цього в Ассемблері існує поняття «сегмент пам'яті». Існують директиви, що оголошують кожен такий сегмент:

- сегмент коду (пам'яті програм) ..... .cseg;
- сегмент даних (SRAM) .... .dseg;
- сегмент EEPROM .... .eseg.

Після оголошення кожного такого сегмента він стає поточним. Це значить, що всі наступні оператори відносяться виключно до оголошеного сегменту. Оголошений сегмент буде залишатися поточним доти, поки не буде оголошений який-небудь інший сегмент.

Тільки в сегменті коду Ассемблер описує команди, які потім у вигляді кодів будуть записані в пам'ять програм. В інших двох сегментах використовуються директиви розподілу пам'яті і директиви опису даних. До сегментів dseg і eseg ми ще повернемося. Зараз же розглянемо сегмент cseg.

Так як команди в програмній пам'яті повинні розташовуватися по порядку, одна за одною, то їх розміщення зручно автоматизувати. Програміст не вказує, за якою адресою в пам'яті повинна бути розташована та чи інша команда. Програміст просто послідовно пише команди.

А вже транслятор автоматично розміщує їх в пам'яті. Для цього використовується поняття «покажчик поточної адреси». Покажчик поточної адреси не має відношення до регістру адреси мікроконтролера і взагалі фізично не існує. Це просто поняття, що використовується в мові Асемблер. Покажчик допомагає транслятору розмістити всі команди програми по осередках пам'яті.

За умовчанням вважається, що на початку програми значення поточного покажчика дорівнює нулю. Тому перша ж команда програми буде розміщена за нульовою адресою. У міру трансляції програми покажчик зміщується в бік збільшення адреси. Якщо команда має довжину в один байт, то після її трансляції покажчик зміщується на одну клітинку. Якщо команда складається з двох байтів, на дві. Таким чином розміщуються всі команди програми.

.org

Примусове позиціонування покажчика поточної адреси.

Іноді необхідно розмістити який-небудь фрагмент програми в програмній пам'яті не відразу після попереднього фрагмента, а в конкретному місці програмної пам'яті. Наприклад, починаючи з якого-небудь заздалегідь визначеного адреси. Для цього використовують директиву .org.

Вона дозволяє примусово змінити значення покажчика поточної адреси. Оператор .org має всього один параметр – нове значення покажчика адреси. Приміром, команда .org 0x10 встановить покажчик на адресу 0x10. Транслятор автоматично стежить, щоб при переміщенні покажчика ваші фрагменти програми не налізали одна на одну. У разі недотримання цієї умови транслятор видає повідомлення про помилку.

У нашій програмі команда позиціонування покажчика застосовується всього один раз. У рядку 5 покажчик встановлюється на нульовій адресі. В даному випадку директива org має чисто декларативне значення, тому що на початку програми значення покажчика і так дорівнює нулю.

5 Оператори

ldi

Завантаження в SRAM числової константи.

У рядку 6 програми (Лістинг 5.1) за допомогою цієї команди в регістр temp (r l6) записується числова константа, що дорівнює максимальній адресі SRAM. Ця константа має ім'я RAMEND. Її значення описано в файлі tn2313def. inc. У нашому випадку (для мікроконтролера ATiny2313) значення RAMEND дорівнює $7F.

Як можна бачити з лістингу 5.1, оператор ldi має два параметри:

- перший параметр – це ім'я SRAM, куди міститься наша константа;
- другий параметр – значення цієї константи.

Зверніть увагу, що в команді спочатку записується приймач інформації, потім її джерело. Такий же порядок ви побачите в будь-який інший команді, що має два операнди. Це загальне правило для мови Асемблер.

out

Вивідення вмісту SRAM в регістр вводу-виводу (PBB).

Команда також має два параметри:

-перший параметр - ім'я PBB, що є приймачем інформації;

- другий параметр - ім'я SRAM, що є джерелом.

У рядку 7 програми вміст регістра temp виводиться в PBB з ім'ям SPL.

in

Введення інформації з регістра вводу-виводу.

Має два параметра. Параметри ті ж, що і в попередньому випадку, але джерело і приймач міняються місцями. У рядку 16 програми вміст регістра PORTD поміщується в регістр temp.

rjmp

Команда безумовного переходу.

Команда має всього один параметр – адресу переходу. У рядку 18 програми оператор безумовного переходу передає управління на рядок, позначений міткою main – на рядок 16. Цей рядок демонструє використання мітки.

Насправді у якості параметра оператора rjmp повинна виступати так звана відносна адреса переходу. Тобто, число байт, на яке потрібно зміститися вгору або вниз від поточної адреси. Напрямок зміщення (вгору або вниз) – це знак числа. Він визначається старшим бітом. Мова Асемблера позбавляє програміста від необхідності підрахунку величини зсуву. Досить у потрібному рядку програми поставити мітку, а в якості адреси переходу вказати її ім'я, і транслятор сам обчислить значення цього параметру.

При використанні команди *rjmp* існує одне обмеження. Відповідна команда мікроконтролера кодується за допомогою одного шістнадцятирозрядного слова. Для вказівки величини зміщення вона використовує всього дванадцять розрядів. Тому така команда може викликати перехід в межах ± 2 Кбайт. Якщо ви розташуєте мітку занадто далеко від оператора *rjmp*, то при трансляції програми це викличе повідомлення про помилку.

6 Опис програми (лістинг 5.1)

Текст програми починається шапкою з назвою програми. Шапка являє собою кілька рядків коментарів. Шапка на початку програми допомагає відрізняти програми одну від іншої. Крім назви програми, в шапку можна помістити її версію, а також дату написання.

Найперша команда програми – це псевдокоманда include, яка приєднує до основного тексту програми файл описів (див. Лістинг 5.1 рядок 1). У стандартному пакеті AVR-Studio є цілий набір подібних файлів описів. Для кожного мікроконтролера серії AVR - свій окремий файл. Всі стандартні файли описів знаходяться в директорії «C: \ Program Files \ Atmel \ AVR Tools \ AvrAssembler \ Appnotes \». Програмісту потрібно лише вибрати потрібний файл і включити подібний рядок у свою програму. Врахуйте, що без приєднання файлу описів подальша програма працювати не буде.

Лістинг 5.1

```
;################################################
;##                    Пример 1                  ##
;##        Программа управления светодиодом      ##
;################################################

;------------------------ Псевдокоманды управления

1       .include "tn2313def.inc"        ; Присоединение файла описаний
2       .list                           ; Включение листинга
3       .def    temp = r16              ; Определение главного рабочего регистра

;------------------------ Начало программного кода

4       .cseg                           ; Выбор сегмента программного кода
5       .org    0                       ; Установка текущего адреса на ноль

;------------------------ Инициализация стека

6       ldi             temp, RAMEND    ; Выбор адреса вершины стека
7       out             SPL, temp       ; Запись его в регистр стека

;------------------------ Инициализация портов ВВ

8       ldi             temp, 0         ; Записываем 0 в регистр temp
9       out             DDRD, temp      ;Записываем этот 0 в DDRD (порт PD на ввод)

10      ldi             temp, 0xFF      ;Записываем число $FF в регистр temp
11      out             DDRB, temp      ; Записываем temp в DDRB (порт PB на вывод)
12      out             PORTB, temp     ; Записываем temp в PORTB (потушить светодиод)
13      out             PORTD, temp     ; Записываем temp в PORTD (включ. внутр. резист.)

;------------------------ Инициализация компаратора

14      ldi             temp, 0x80      ; Выключение компаратора
15      out             ACSR, temp

;------------------------ Основной цикл

16 main:in              temp, PIND      ; Читаем содержимое порта PD
17      out             PORTB, temp     ; Пересылаем в порт PB
18      rjmp            main            ; К началу цикла
```

Для мікроконтролера ATiny2313 файл описів має назву tn2313def. inc. Якщо файл описів знаходиться у зазначеній вище директорії, то в команді .include достатньо лише вказати його повне ім'я (з розширенням). Вказувати повний шлях необов'язково.

Призначення команди .list (рядок 2) вже відомо. Зупинимося на команді макроозначення .def (рядок 3). Ця команда, як вже говорилося, привласнює регістру r16 ім'я temp. Далі в програмі регістр temp використовується для тимчасового зберігання проміжних величин. Обраний саме регістр r16, бо регістри, починаючи з r0 і закінчуючи r15, мають менше можливостей. Наприклад, в рядку 14 програми регістр temp використовується в команді ldi. Однак команда ldi не працює з регістрами r0 – r15. Саме з цієї причини ми і вибрали r16.

Наступні дві команди (рядки 4, 5) докладно описані на початку цього розділу. Вони служать для вибору програмного сегмента пам'яті і установки початкового значення покажчика.

У рядках 6 і 7 виробляється ініціалізація стека. В регістр стека SPL записується адресу його вершини. В якості адреси обраний самий верхній адреса SRAM. Для позначення цієї адреси в даній версії Ассемблера існує спеціальна константа з ім'ям RAMEND. Значення цієї константи визначається у файлі описів (в нашому випадку у файлі tn2313def. Inc). Для мікроконтролера Atiny2313 константа RAMEND дорівнює 0x7F.

Одним рядком записати константу в регістр стека неможливо, так як в системі команд мікроконтролерів AVR відсутня подібна команда. Відсутню команду ми замінюємо двома іншими. І тут нам стане в нагоді регістр temp. Він послужить в даному випадку передавальною ланкою. Спочатку константа RAMEND поміщується в регістр temp (рядок 6), а потім вже вміст temp поміщується в регістр SPL (рядок 7).

У рядках 8-12 проводиться налаштування портів введення-виведення.

Раніше ми прийняли, що порт PD у нас працюватиме на введення, а порт PB – на виведення. Для вибору потрібного напрямку передачі інформації запишемо керуючі коди у відповідні регістри DDRx. У всі розряди регістра DDRD запишемо нулі (настройка порту PD на введення), а в усі розряди регістра DDRB запишемо одиниці (настройка порту PB на вивід). Крім того, нам потрібно включити внутрішні навантажувальні резистори порту PD. Для цього ми запишемо одиниці (тобто число 0xFF) в усі розряди регістру PORTD. І, нарешті, в момент старту програми бажано погасити світлодіод. Для цього ми запишемо одиниці в розряди порту PB.

Всі описані вище дії з налаштування порту також виконуються з використанням проміжного регістра temp. Спочатку в нього заноситься нуль (рядок 8). Нуль записується тільки в регістр DDRD (рядок 9). Потім в регістр temp заноситься число 0xFF (рядок 10). Це число по черзі записується в регістри DDRB, PORTB, PORTD (рядки 11,12,13).

Рядки 14 і 15 включені в програму для перестраховки. Справа в тому, що вбудований компаратор мікроконтролера після системного скидання залишається включений. І хоча переривання при цьому відключені і спрацьову-

вання компаратора не може вплинути на роботу нашої програми, ми все ж відключимо компаратор. Саме це і робиться в рядках 14 і 15.

Тут вже знайомим нам способом з використанням регістра temp проводиться запис константи 0x80 у регістр ACSR. Регістр ACSR призначений для управління режимами роботи компаратора, а константа 0x80, записана в цей регістр, відключає компаратор.

Налаштуванням компаратора закінчується підготовча частина програми. Підготовча частина займає рядка 1-15 і виконується всього один раз після включення живлення або після системного скидання.

Рядки 16-18 становить основний цикл програми.

Визначення. Основний цикл – *це частина програми, яка повторюється багато разів і виконує всі основні дії.*

В нашому випадку, згідно з алгоритмом, дії програми полягають у тому, щоб прочитати стан кнопки і перенести його на світлодіод. Є багато способів перенести вміст молодшого розряду порту PD в молодший розряд порту PB. В нашому випадку реалізований найпростіший варіант. Ми просто переносимо одночасно всі розряди. Для цього достатньо двох операторів.

Перший з них читає вміст порту PD і запам'ятовує цей вміст в регістрі temp (рядок 16). Наступний оператор записує це число в порт PB (рядок 17). Завершує основний цикл програми оператор безумовного переходу (рядок 18). Він передає управління по мітці main.

В результаті три оператори, що складають тіло циклу, повторюються нескінченно. Завдяки цьому нескінченному циклу всі зміни порту PD тут же потрапляють в порт PB. З цієї причини, якщо кнопка S1 не натиснута, логічна одиниця зі входу PD0 за один прохід циклу передається на вихід PB0. І світлодіод не світиться. При натисканні кнопки S1 логічний нуль зі входу PD0 надходить на вихід PB0, і світлодіод загоряється.

Ця ж сама програма без будь-яких змін може обслуговувати до семи кнопок і таку ж кількість світлодіодів. Додаткові кнопки підключаються до ліній PD1-PD6, а додаткові світлодіоди (кожен зі своїм струмообмежувальним резистором) – до виходів PB1-PB7. При цьому кожна кнопка буде керувати своїм власним світлодіодом. Це стало можливим, тому що всі висновки кожного з двох портів ми налаштували однаково (дивись рядки 8-13).

7 Завдання для самостійної роботи

7.1 Вивчити досліджувану принципову електричну схему;
7.2 Повторити опис інтерфейсу – головну панель програмного комплексу «AVR Studio»;
7.3 Повторити програмну модель, структуру та особливості функціонування мікроконтролера ATiny2313;

7.4 Повторити структуру та особливості роботи портів AVR-мікроконтролерів;
7.5 Виконати наступні пункти завдання до лабораторної роботи: 8.1 - 8.4.

8 Завдання до лабораторної роботи

8.1 Скласти підготовчу частину програми для роботи пристрою;
8.2 Скласти основний цикл програми;
8.3 Розробити програму для роботи заданого пристрою відповідно до наведеного прикладу;
8.4 Привести опис програми;
8.5 За допомогою програмного комплексу «AVR Studio» створіть власний проект.

9 Зміст звіту

9.1 Мета роботи;
9.2 Принципова схема мікропроцесорного пристрою;
9.3 Лістинг програми з п.8.3 відповідно до прийнятих вимог
9.4 Опис програми;
9.5 Висновки.

10 Контрольні питання

10.1 Яке призначення навісних елементів мікропроцесорного пристрою на рис. 2.1?
10.2 Як вдосконалити принципову схему на рис. 2.1? Відповідь обгрунтуйте.
10.3 Як і для чого здійснюється настройка портів мікроконтролера?
10.4 Яким чином описуються процеси в схемі, що супроводжують процес запалювання і гасіння світлодіода ?
10.5 Яким чином словесно описати алгоритм роботи мікропроцесорного пристрою ?
10.6Як описати порядок оформлення програм на асемблері ?
10.7 Що таке оператор і операнди ассемблерної команди ?
10.8 Які висновки можна зробити із порівняння функції оператора і директиви в мові Асемблер ?
10.9 Як розробити підготовчу частину програми ?
10.10 Які існують команди основного циклу програми ?

Система команд AVR-мікроконтролерів

У таблиці 1 наведені умовні позначення, що використовуються при описі команд AVR-мікроконтролерів, а у таблиці 2 – система команд AVR-мікроконтролерів.

Таблиця 1 – Позначення, що використовуються при описі команд

Позначення, символ	Опис
Регістр стану	
SREG	Регістр стану мікроконтролера
C	Прапор пренесення (1-й розряд регістра)
Z	Прапор нуля (1-й розряд регістра SREG)
N	Прапор негативного значення (2-й розряд регістра SREG)
V	Прапор переповнення додаткового коду (3-й розряд регістра SREG)
S	Прапор знаку (4-й розряд регістра SREG); S=N⊕V
H	Прапор половинного перенесення (5-й розряд регістра SREG)
T	Прапор користувача (6-й розряд регістра SREG)
I	Прапор загального дозволу переривань (7-й розряд регістра SREG)
Регістри та операнди	
R_d	Регістр-приймач (іноді також регістр-джерело) в регістровому файлі
R_r	Регістр-приймач в регістровому файлі
K	Константа (дані)
k	Адреса-константа
b	Номер розряду РЗП або РВВ (0...7)
s	Номер розряду регістра стану SREG (0...7)
X, Y, Z	Регістри-вказівники (X=R27:R26; Y=R29:R28; Z=R31:R30)
I/O	Регістр введення/виведення
P	Адреса в просторі введення/виведення
q	Зміщення при відносній непрямій адресації (6-розрядне значення)
,	Роздільник між назвою (адресою) регістра і номером розряду
Операції	
•	Логічне І
∨	Логічне АБО
⊕	Виключне АБО
Система	
PC	Лічильник команд (програмний лічильник)
STAC	Поточний рівень стека
SP	Вказівник стека
Прапори	
0	Прапор скидається крмандою в „0”
1	Прапор встановлюється командою в „1”
-	Команда не впливає на стан прапора

Табліця 2 – Система команд AVR-мікроконтролерів

Мнемонічне позначення	Опис	Операція
АРИФМЕТИЧНІ І ЛОГІЧНІ КОМАНДИ		
ADD Rd,Rr	Скласти без переносу	Rd ← Rd + Rr
ADC Rd,Rr	Скласти з переносом	Rd ← Rd + Rr + C
ADIW Rd,K	Скласти безпосереднє значення зі словом	Rdh:Rdl ← Rdh:Rdl+K
SUB Rd,Rr	Відняти без переносу	Rd ← Rd - Rr

SUBI Rd,K	Відняти безпосереднє значення	Rd ← Rd - K
SBC Rd,Rr	Відняти з переносом	Rd ← Rd - Rr - C
SBCI Rd,K	Відняти безпосереднє значення з переносом	Rd ← Rd - K - C
SBIW Rd,K	Відняти безпосереднє значення зі слова	Rdh:Rdl ← Rdh:Rdl-K
AND Rd,Rr	Виконати логічне AND	Rd ← Rd • Rr
ANDI Rd,K	Виконати логічне AND з безпосереднім значенням	Rd ← Rd • K
OR Rd,Rr	Виконати логічне OR	Rd ← Rd v Rr
ORI Rd,K	Виконати логічне OR з безпосереднім значенням	Rd ← Rd v K
EOR Rd,Rr	Виконати виключне OR	Rd ← Rd ⊕ Rr
COM Rd	Виконати доповнення до одиниці	Rd ← $FF – Rd
NEG Rd	Виконати доповнення до двох	Rd ← $00 – Rd
SBR Rd,K	Установити біти в регістрі	Rd ← Rd v K
CBR Rd,K	Очистити біти в регістрі	Rd ← Rd • ($FF - K)
INC Rd	Інкрементувати	Rd ← Rd + 1
DEC Rd	Декрементувати	Rd ← Rd – 1
TST Rd	Перевірити на нуль чи мінус	Rd ← Rd • Rd
CLR Rd	Очистити регістр	Rd ← Rd ⊕ Rd
SER Rd	Установити всі біти регістра	Rd ← $FF
CP Rd,Rr	Порівняти	Rd – Rr
CPC Rd,Rr	Порівняти з урахуванням переносу	Rd – Rr – C
CPI Rd,K	Порівняти з константою	Rd – K
КОМАНДИ ПЕРЕХОДУ		
RJMP k	Перейти відносно	PC ← PC + k + 1
IJMP	Перейти побічно	PC ← Z
JMP k	Перейти	PC ← k
RCALL k	Викликати підпрограму відносно	PC ← PC + k + 1
ICALL	Викликати підпрограму побічно	PC ← Z
CALL k	Виконати довгий виклик підпрограми	PC ← k
RET	Повернутися з підпрограми	PC ← STACK
RETI	Повернутися з переривання	PC ← STACK
CPSE Rd,Rr	Порівняти і пропустити якщо дорівнює	if (Rd = Rr)PC ← PC + 2 or 3
SBRC Rd,b	Пропустити якщо біт у регістрі очищений	if (Rr(b)=0)PC ← PC + 2 or 3
SBRS Rd,b	Пропустити якщо біт у регістрі встановлений	if (Rr(b)=1)PC ← PC + 2 or 3
SBIC P,b	Пропустити якщо біт у регістрі I/O очищений	if (P(b)=0)PC ← PC + 2 or 3
SBIS P,b	Пропустити якщо біт у регістрі I/O установлений	if (P(b)=1)PC ← PC + 2 or 3
BRBS s,k	Перейти якщо біт у регістрі статусу встановлений	if (SREG(s) = 1) then PC←PC+k +1
BRBC s,k	Перейти якщо біт у регістрі статусу очищений	if(SREG(s) = 0) then PC←PC+k + 1
BREQ k	Перейти якщо дорівнює	if (Z = 1) then PC ← PC + k + 1
BRCS k	Перейти якщо прапор переносу встанов-	if (C = 1) then PC ← PC + k + 1

	лений	
BRNE k	Перейти якщо не дорівнює	if (Z = 0) then PC ← PC + k + 1
BRCC k	Перейти якщо прапор переносу очище-ний	if (C = 0) then PC ← PC + k + 1
BRSH k	Перейти якщо дорівнює чи більше(без знака)	if (C = 0) then PC ← PC + k + 1
BRLO k	Перейти якщо менше (без знака)	if (C = 1) then PC ← PC + k + 1
BRMI k	Перейти якщо мінус	if (N = 1) then PC ← PC + k + 1
BRPL k	Перейти якщо плюс	if (N = 0) then PC ← PC + k + 1
BRGE k	Перейти якщо чи більше дорівнює(з урахуванням знака)	if (N ⊕ V= 0) then PC ← PC + k + 1
BRLT k	Перейти якщо менше ніж (зі знаком)	if (N ⊕ V= 1) then PC ← PC + k + 1
BRHS k	Перейти якщо прапор напівпереносу встановлений	if (H = 1) then PC ← PC + k + 1
BRHC k	Перейти якщо прапор напівпереносу очищений	if (H = 0) then PC ← PC + k + 1
BRTS k	Перейти якщо прапор T установлений	if (T = 1) then PC ← PC + k + 1
BRTC k	Перейти якщо прапор T очищений	if (T = 0) then PC ← PC + k + 1
BRVS k	Перейти якщо переповнення встановле-не	if (V = 1) then PC ← PC + k + 1
BRVC k	Перейти якщо переповнення очищене	if (V = 0) then PC ← PC + k + 1
BRIE k	Перейти якщо глобальне переривання дозволене	if (I = 1) then PC ← PC + k + 1
BRID k	Перейти якщо глобальне переривання заборонене	if (I = 0) then PC ← PC + k + 1
КОМАНДИ ПЕРЕДАЧІ ДАНИХ		
MOV Rd,Rr	Копіювати регістр	Rd ← Rr
LDI Rd,K	Завантажити безпосереднє значення	Rd ← K
LD Rd,X	Завантажити побічно	Rd ← (X)
LD Rd,X+	Завантажити побічно инкрементировав згодом	Rd ← (X), X ← X + 1
LD Rd,-X	Завантажити побічно декрементировав попередньо	X ← X - 1, Rd ← (X)
LDDRd,Y+q	Завантажити побічно сосмещением	Rd ← (Y + q)
LDS Rd,k	Завантажити безпосередньо із СОЗУ	Rd ← (k)
ST X,Rr	Записати побічно	(X) ← Rr
ST X+,Rr	Записати побічно инкрементировав зго-дом	ST (X) ← Rr,X ←X+ 1
ST -X,Rr	Записати побічно декрементировав по-передньо	X ← X - 1, (X) ← Rr
STD Y+q,Rr	Записати побічно зі зсувом	(Y + q) ← Rr
STS k,Rr	Завантажити безпосередньо в СОЗУ	(k) ← Rr
LPM	Завантажити байт пам'яті програм	R0 ← (Z)
ELPM	Розширене завантаження пам'яті про-грам	R0 ← (RAMPZ + Z)
IN Rd,P	Завантажити дані з порту I/O в регістр	Rd ← P
OUT P, Rr	Записати дані з егистра в порт I/O	P ← Rr
PUSH Rr	Помістити регістр у стек	STACK ← Rr; SP← SP-1

POP Rd	Завантажити регістр зі стека	SP← SP+1, Rd ← STACK
КОМАНДИ РОБОТИ З БІТАМИ		
SBI P,b	Установити біт у регістр I/O	I/O(P,b) ← 1
CBI P,b	Очистити біт у регістрі I/O	I/O(P,b) ← 0
LSL Rd	Логічно зрушити вліво	Rd(n+1) ← Rd(n), Rd(0) ← 0
LSR Rd	Логічно зрушити вправо	Rd(n) ← Rd(n+1), Rd(7) ← 0
ROL	Зрушити вліво через перенос	Rd(0)←C,Rd(n+1)← Rd(n), C←Rd(7)
ROR Rd	Зрушити вправо через перенос	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)
ASR Rd	Арифметично зрушити вправо	Rd(n) ← Rd(n+1), n=0..6
SWAP Rd	Поміняти нібли місцями	Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0)
BSET s	Установити прапор	SREG(s) ← 1 SREG(s)
BCLR s	Очистити прапор	SREG(s) ← 0 SREG(s)
BLD Rd,b	Завантажити T прапор у біт регістра	Rd(b) ← T
BST Rd,b	Переписати біт з регістра в прапор T	T ← Rr(b)
CLC	Очистити прапор переносу	C← 0
SEC	Установити прапор переносу	C ← 1
SEN	Установити прапор негативного значення	N ← 1
CLN	Очистити прапор негативного значення	N ← 0
SEZ	Установити прапор нульового значення	Z ← 1
CLZ	Очистити прапор нульового значення	Z ← 0
SEI	Установити прапор глобального переривання	I ← 1
CLI	Очистити прапор глобального переривання	I ← 0
SES	Установити прапор знака	S ← 1
CLS	Очистити прапор знака	S ← 0
SEV	Установити прапор переповнення	V ← 1
CLV	Очистити прапор переповнення	V ← 0
SET	Установити прапор T	T ← 1
CLT	Очистити прапор T	T ← 0
SEH	Установити прапор підлоги переносу	H ← 1
CLH	Очистити прапор підлоги переносу	H ← 0
NOP	Виконати холосту команду	None
SLEEP	Установити режим SLEEP	None
WRD	Скинути сторожовий таймер	None

Література

1. Евстифеев А.В. Микроконтроллеры AVR семейства Mega. Руководство пользователя. – М.: Издательский дом «Додэка – XXI», 2007. – 592 с.: ил.
2. Евстифеев А.В. Микроконтроллеры AVR семейства Tiny. Руководство пользователя. – М.: Издательский дом «Додэка – XXI», 2007. – 432 с.: ил.
3. Баранов В.Н.Применение микроконтроллеров AVR: схемы, алгоритмы, программы. – М.: Издательский дом «Додэка – XXI», 2004. – 288 с.: ил.
4. Швец В.А. и др. Одноплатные микроконтроллеры. Проектирование и применение. – К.: «МК-Пресс», 2006. – 304 с., ил.
5. Бродин В.Б., Калинин А.В. Системы на микроконтроллерах и БИС программируемой логики – М.: Издательство ЭКОМ, 2002. – 400 с.: ил.
6. Великий В.І. Мікропроцесорні системи в САУ: Курс лекцій. Навч. посібник. – О.: Наука і техніка, 2006. – 192 с.: ил.
7. Великий В.І., Препелиця Г.П. Мікропроцесорні системи обробки даних та управління. Навчальний посібник. – Одеса: Вид-во «ТЭС», 2004. – 212 с.: ил

Електронні копії літературних джерел 1 – 5 можно получити в лабораторії кафедри, посібники 6,7 – у бібліотеці університету