МИНИСТЕРСТВО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

**МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ИНСТИТУТ
РАДИОТЕХНИКИ, ЭЛЕКТРОНИКИ И АВТОМАТИКИ
(ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ)**

Факультет электроники

Методические указания
по выполнению лабораторной работы

ИЗУЧЕНИЕ АРХИТЕКТУРЫ И ОСНОВ ПРОГРАММИРОВАНИЯ МИКРОКОНТРОЛЛЕРОВ

Для студентов специальностей

072000, 190400, 190700,
200100, 200300

МОСКВА 2003

Составители: В.И. Индришенок,
В.В. Кузнецов,
Е.Ф. Певцов

Редактор Е.Ф. Певцов

Учебно-методические указания для выполнения лабораторной работы «Изучение архитектуры и основ программирования микроконтроллеров», подготовлены преподавателями факультета электроники Московского государственного института радиотехники, электроники и автоматики. Они предназначены для студентов обучающихся по направлению 654300 и специальностям 072000, 190400, 190700, 200100, 200300, изучающих общие дисциплины по информационным технологиям, а также специализирующимся в области проектирования систем управления, сбора и обработки данных.

Печатаются по решению редакционно-издательского совета
Московского государственного института радиотехники,
электроники и автоматики (технического университета).

2003

Введение

Изобретение микропроцессора считается величайшим событием XX века. Создатели первого микропроцессора Тед Хофф, Федерико Феджина и Стен Мейзор работали в компании INTEL, их имена занесены в национальный зал славы США. Первый ЧИП микропроцессора 4004 (1971 г.) состоял из $2{,}3 \cdot 10^3$ транзисторов, работал с тактовой частотой 750 кГц, выполняя 60 тыс. операций в секунду, и стоил около 200 долларов. Характеризуя современные темпы развития микропроцессорной техники, председатель совета директоров INTEL Гордон Мур высказал следующую аналогию: «Если бы автомобилестроение эволюционировало со скоростью полупроводниковой промышленности, то сегодня "Ролс-Ройс" стоил бы три доллара, мог бы проехать полмиллиона километров на одном галлоне бензина, и было бы дешевле его выбросить, чем платить за парковку».

Интенсивное развитие микроконтроллеры (МК) получили в середине 90-х годов. Это в значительной степени обусловлено развитием высоких технологий и удешевлением массового производства СБИС. До этого времени обработка сигналов от периферийных устройств осуществлялись по идеологии единого обрабатывающего центра, связанного общей шиной с другими устройствами системы (автомобильная электроника в это время требовала прокладки до 90 кг проводов!). Цена микроконтроллеров неуклонно снижалась, и стало выгоднее встраивать их в периферийные схемы и всю необходимую обработку проводить непосредственно в этих схемах, минимально загружая центр управления и выдавая ему только необходимую информацию.

С удешевлением производства СБИС и ростом их надежности связано также все большее распространение компьютеров, позволяющих ускорить процесс обработки информации за счет специальной архитектуры, главной особенностью которых является применение разделенных физически областей памяти для хранения программ и данных. Такие компьютеры принято называть еще компьютерами с гарвардской архитектурой, поскольку их идеология была в свое время разработана Гарвардским университетом. Эти устройства принципиально отличаются от ком-

пьютеров, в которых для хранения данных и для хранения программ используется общая память. По имени университета, где они впервые были разработаны, их называют компьютеры с принстонской архитектурой, а по имени ученого, создавшего теорию их работы – компьютеры с архитектурой Фон-Неймана. То, что компьютеры Фон-Неймана получили вначале большее распространение связано, с недостаточной надежностью систем с параллельной обработкой того времени, поскольку они базировались на устройствах с малой степенью интеграции на электролампах и дискретных транзисторных схемах.

По набору команд, используемому микропроцессорами, принято разделять также компьютеры с сокращенной системой команд (RISC – Reduced Instruction Set Computers) и компьютеры с полной системой команд (CISC – Complex Instruction Set Computers).

В общем случае контроллер или микропроцессорная система управляет объектами, используя комбинированные аппаратно-программные реализации функций управления. Результатом работы являются данные на магистрали (специальной шине) и модифицированное содержимое регистров – специальных ячеек памяти данных. Функционально достижение этих результатов может быть реализовано программно с помощью микропроцессоров (МП) или аппаратно с помощью самостоятельно работающих специальных СБИС, таких как микроконтроллеры (МК), цифровые автоматы, преобразователи данных, синхронизированных с микропроцессором или привязанных к его сетке работы с помощью системы прерываний.

Микроконтроллер отличается от микропроцессора тем, что аппаратные операции, наиболее часто используемые в микропроцессорных системах управления, выполняются внутренними модулями, интегрированными на кристалле вместе с процессорным ядром. Кроме памяти программ и данных к таким модулям относятся: таймеры/счетчики, модули АЦП и ЦАП, модули управления электродвигателями и т.п. Для сегодняшних МК характерна тенденция к объединению многих функциональных модулей в одном кристалле.

Основные элементы архитектуры микроконтроллера

Как правило, семейства МК имеют единые базовые структуры. В качестве типичного примера семейства МК AVR рассмотрим архитектуру МК AT90S8515 (см. рис. 1). С точки зрения изучения принципов работы и программирования можно выделить следующие основные элементы МК AT90S8515:

– ***генератор тактового сигнала (GCK);***

– ***процессор*** (CPU);

– ***память программ***, предназначенная для хранения программного кода и констант;

– ***память данных***, в которой сохраняются данные и значения переменных;

– набор ***периферийных устройств*** (регистры, порты, таймер/счетчик, аналого-цифровой преобразователь и др.) для ввода и вывода данных, управляющих сигналов и выполнения других функций.

Рассмотрим назначение и организацию работы этих элементов.

МК семейства AVR – устройства синхронного типа, т.е. операции, выполняемые МК, привязаны по времени к импульсам тактового сигнала. Для МК AT90S8515 в качестве ***генератора тактового сигнала*** используется либо внешний генератор (режим EXT), либо внутренний генератор с внешним кварцевым или керамическим резонатором (режим XTAL).

Процессор микроконтроллера выполняет три основные операции, обеспечивающие выполнение программы:

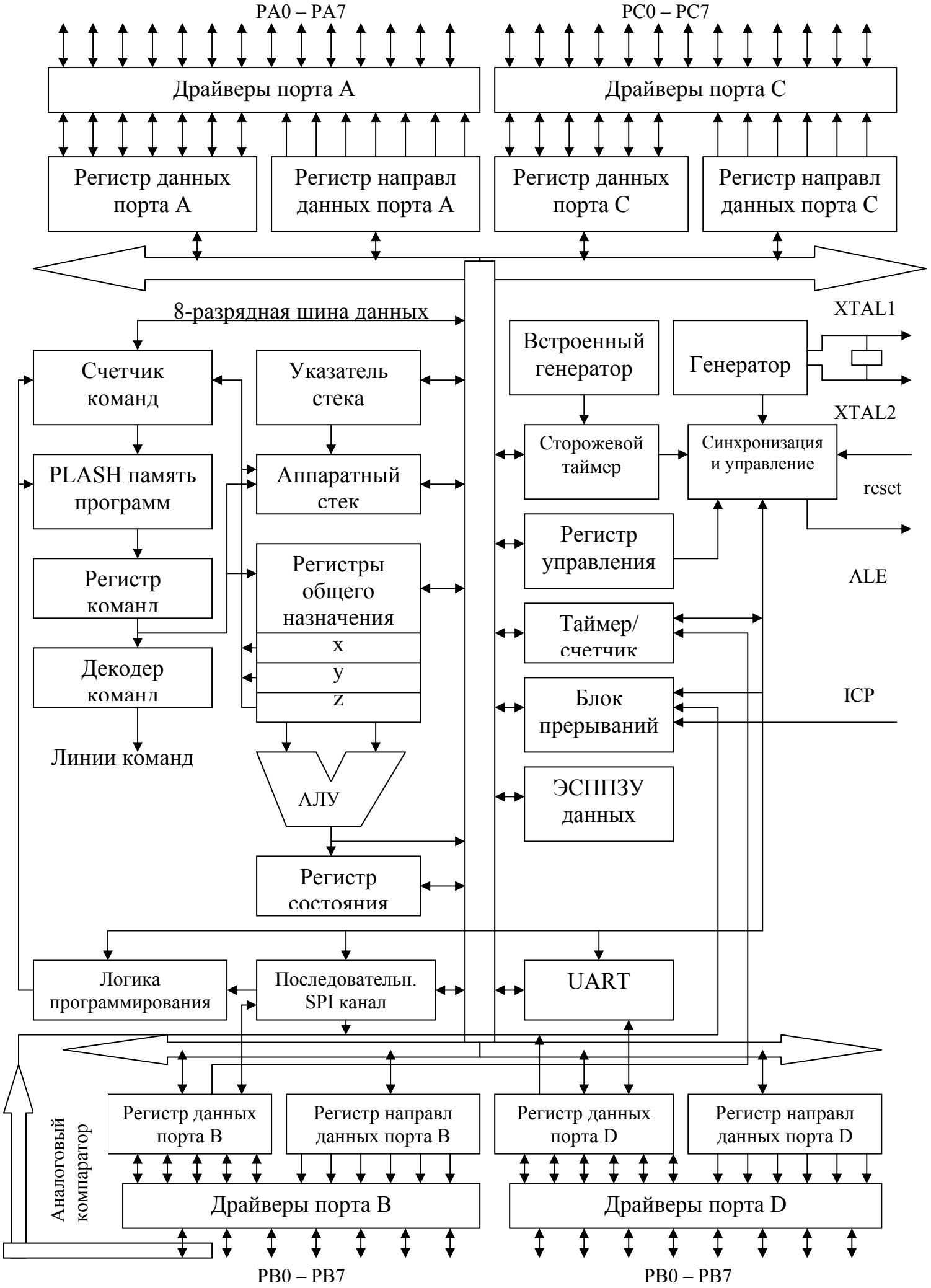
1) формирует адрес очередной команды;

2) выбирает команду из памяти;

3) организует выполнение команды.

Основные узлы процессора:

– *счетчик команд* или *программный счетчик*
– *(PC)*; *дешифратор (декодер) команд;*
– *арифметико-логическое устройство (ALU);*
– *блок регистров общего назначения РОН* (*GPR*);
– *регистр состояния МК (SREG)*;
– *регистр-указатель стека (SP)*.


PA0 – PA7
PC0 – PC7
Драйверы порта A
Драйверы порта C
Регистр данных порта A
Регистр направл данных порта A
Регистр данных порта C
Регистр направл данных порта C
8-разрядная шина данных
XTAL1
XTAL2
Встроенный генератор
Генератор
Счетчик команд
Указатель стека
Сторожевой таймер
Синхронизация и управление
reset
PLASH память программ
Аппаратный стек
Регистр управления
ALE
Регистр команд
Регистры общего назначения
x
y
z
Таймер/ счетчик
Декодер команд
Блок прерываний
ICP
Линии команд
АЛУ
ЭСППЗУ данных
Регистр состояния
Логика программирования
Последовательн. SPI канал
UART
Аналоговый компаратор
Регистр данных порта B
Регистр направл данных порта B
Регистр данных порта D
Регистр направл данных порта D
Драйверы порта B
Драйверы порта D
PB0 – PB7
PB0 – PB7

При работе МП под управлением программы реализуется алгоритмический принцип следования (не рассматриваются многопроцессорные системы). *Программный счетчик* является указателем на подлежащую выполнению команду и содержит адрес этой команды в памяти программ. Он работает в 3-х режимах:

1) сброс (при пуске и перезапуске МК очищается PC и выполнение программы начинается с команды, расположенной в памяти по нулевому адресу);
2) инкремент адресов (значение PC увеличивается, указывая на команду программы, следующую за выполняемой);
3) установка PC в соответствии с кодом команды условного или безусловного перехода.

Установку программного счетчика для выбора следующей команды реализует *дешифратор команд (логика программирования)*. Дешифратор, также, по коду команды вырабатывает управляющие сигналы, обеспечивающие её выполнение. Для полного выполнения *командного цикла* может требоваться несколько *тактов синхронизации*, а некоторые команды выполняются за несколько командных циклов.

Арифметико-логическое устройство (Arithmetic Logic Unit) служит исключительно для выполнения арифметических операций (сложение, вычитание, логические И и ИЛИ, сравнение, сдвиг разрядов, установка регистра состояния в соответствии с результатом операции). Данные, с которыми напрямую работает АЛУ, содержатся в РОН. Результат операции (кроме операций сравнения) помещается в РОН и, дополнительно, в *регистре состояния* МК *SREG* устанавливаются признаки результата (флаги нуля, переноса, переполнения и др.).

Регистры микроконтроллера представляют собой специальные элементы памяти, предназначенные для: 1) хранения данных внутри процессора (аккумуляторы, регистр состояния, индексные регистры и т.д.); 2) организации управления (управление прерываниями, таймерами и т.д.); 3) ввода/вывода данных (регистры данных последовательных или параллельных портов и т.д.). Для процессоров с RISC архитектурой характерно, что обращение к регистрам реализуется через указания их адресов. Адреса регист-

ров располагаются в адресном пространстве памяти данных, как это показано на рис. 2, (отображение регистров на память).

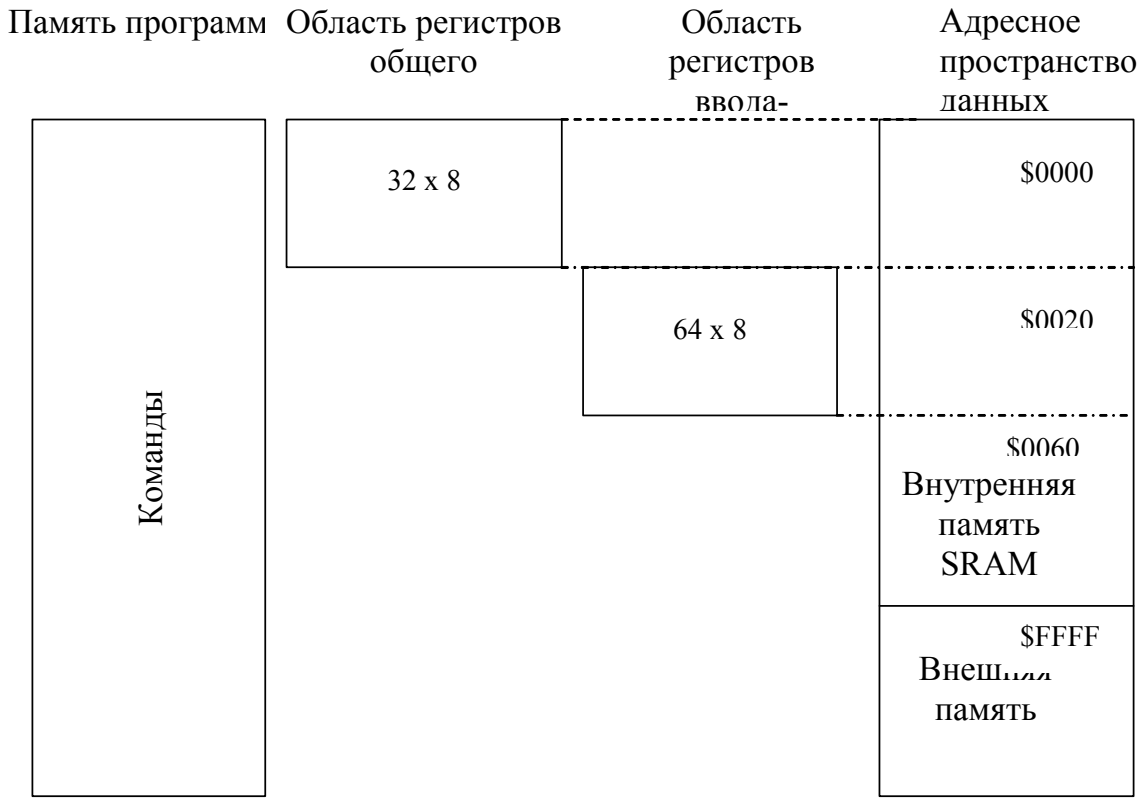


Рис.2. Распределение адресов в микроконтроллере AT90S8515.

Блок *регистров общего назначения* содержит 32 восьмиразрядных регистра с шестнадцатиричными адресами $00 - $1F в пространстве памяти данных. К ним можно обращаться и по именам R0 – R31. Шесть регистров с именами от R24 до R31 могут образовывать пары для хранения шестнадцатиразрядных слов, причем регистр с четным номером хранит младший байт, а регистр с нечетным номером – старший байт. Этим спаренным регистрам присвоены имена X, Y, Z. Они могут использоваться как индексные регистры в командах обращения к памяти данных. Регистр Z может также использоваться для чтения из памяти программ отдельных байтов, что позволяет хранить в ней таблицы данных.

Дополнительно из объема памяти данных может быть выделена специальная область - *стек,* организованный по принципу

"последний вошел - первый вышел" или Last In First Out (LIFO). При этом в процессорах с Гарвардской архитектурой операции со стеком могут производиться одновременно с выполнением других действий.

Для увеличения объема хранения данных к МК может быть подключена *внешняя память*. Имеется два варианта подключения: 1) как к микроконтроллеру; 2) как к устройству ввода/вывода.

Память программ предназначена для хранения кодов команд программы и констант и представляет собой электрически перепрограммируемое постоянное запоминающее устройство – FlashROM (выполненное по специальной полупроводниковой технологии). Объем памяти программ в МК AT90S8515– 8 Кбайт. Ячейка памяти содержит 16 двоичных разрядов. При чтении кодов команд адрес в ПЗУ программ поступает из счетчика команд. При чтении констант адрес поступает из спаренного регистра ***Z***, при этом он должен содержать адрес байта. Отличительной особенностью МК является то, что для большинства из них реализована возможность перепрограммирования в аппаратуре, где они должны работать (функция *Downloading*, перепрограммирование в системе или *In-System Programming*).

Память данных включает в себя регистровый файл РОН, область регистров ввода/вывода и оперативное запоминающее устройство статического типа (*Static Random Access Memory - SRAM*). Оперативное запоминающее устройство предназначено для временного хранения данных, получаемых в ходе работы программы. Адреса SRAM начинаются с $60. Объем встроенной в МК *SRAM* обычно ограничен (512 байт в AT90S8515). Ячейка памяти содержит 8 разрядов. Единственным типом операций является обмен данными между SRAM и РОН – загрузка (**load**) байта в РОН и сохранение (**store**) байта в SRAM. Адрес байта при обращении к памяти данных может быть указан в коде команды (прямая адресация) или предварительно записан в пару регистров ***X***, ***Y*** или ***Z*** (косвенная адресация).

Память данных может быть расширена до 64 Кбайт при подключении внешних микросхем статической памяти к портам МК. При этом имеются возможности работы с внешней памятью

как с периферийным устройством, когда протокол обмена данными реализуется программно, и как со встроенной памятью, когда обращение к обоим видам памяти производится с помощью одинаковых команд. В последнем случае внешняя память выбирается, если адрес больше верхнего адреса встроенной SRAM.

Регистры ввода-вывода – это блок из 64-х байтов, в котором содержатся как регистры управления процессором, так и регистры интерфейса ввода-вывода. Существует два способа обращения к регистрам ввода-вывода и РОН:

1) прямое обращение к каждой из этих областей данных (используется в основном при выполнении арифметических операций);
2) объединение всех трех областей регистров и памяти в общее адресное пространство данных.

Как это показано на рис. 2, в адресное пространство МК, помимо адресов, по которым выполняется обращение к ячейкам ОЗУ данных, включены 32 адреса для обращения к РОН (адреса от $00 до $1F) и 64 адреса для обращения к регистрам ввода-вывода (адреса от $20 до $5F). Первой ячейке SRAM соответствует адрес $60. Адрес для обращения к РОН по команде обращения к SRAM совпадает с номером регистра, а адрес для обращения к регистру ввода-вывода зависит от вида команды. В командах **LOAD** и **STORE** используются общие адреса всего пространства памяти данных, включая РОН и регистры ввода/вывода; а в командах **IN** и **OUT**, применяемых для обмена данными между РОН и этими регистрами, и в командах очистки CBI или установки SBI бит указывается номер регистра – адрес в области ввода/вывода (номера с $00 по $3F).

Адреса некоторых регистров, общих для всех моделей AVR, в пространстве регистров ввода-вывода приведены в таблице. Как видно, если применяются команды LOAD или STORE, то к абсолютному адресу следует прибавить начальное смещение $20.

Таким образом, структуру процессора можно представить в упрощенном виде, приведенном на рис. 3. Такое представление удобно для программирования, поскольку оно отражает все доступные операнды и на ней может быть прослежено выполнение всех команд данного семейства МК.

Регистры ввода-вывода	Область данных	Имя регистра	Назначение
$3F	$5F	SREG	регистр состояния
$3E	$5E	SPH	указатель стека (старший байт)
$3D	$5D	SPL	указатель стека (младший байт)
$3B	$5B	GIMSK	регистр маски прерываний
$3A	$5A	GIFR	регистр запросов прерываний
$35	$55	MCUCR	регистр управления микроконтроллера

Кроме параллельных портов ввода/вывода в группу ***периферийных устройств*** входят: *последовательные порты* SPI (Serial Peripheral Interface), UART (Universal Asynchronous Receiver-Transmitter), *таймеры-счетчики* общего назначения и другие устройства.

В случае выборки команды по неправильному адресу поведение МП перестает быть предсказуемым. Чтобы отслеживать такие ситуации, возникающие, например, при воздействии электрических помех, в состав МК включают специальное устройство ***(сторожевой таймер),*** инициирующее сброс МК (т.е. обращение по адресу запуска), если содержимое таймера не будет обновлено в течение заданного промежутка времени.

Для долговременного сохранения данных при отключенном электропитании, записанных при программировании МК или получаемых в ходе выполнения программы, используется 8-ми разрядное ПЗУ на основе ***EEPROM*** *(Electrical Erased Programmable Memory)*.

EEPROM имеет обособленное адресное пространство. При обращении к EEPROM адрес предварительно записывается в расположенные в области ввода вывода регистры адреса этой памяти: младшая часть адреса в регистр EEARL (номер $1E) и старшая в регистр EEARH (номер $1F).

Байт, предназначенный для записи, а также, получаемый при чтении заносится в регистр данных EEDR (номер $1D). Управление процедурами записи или чтения осуществляется через регистр управления EECR (номер $1C).

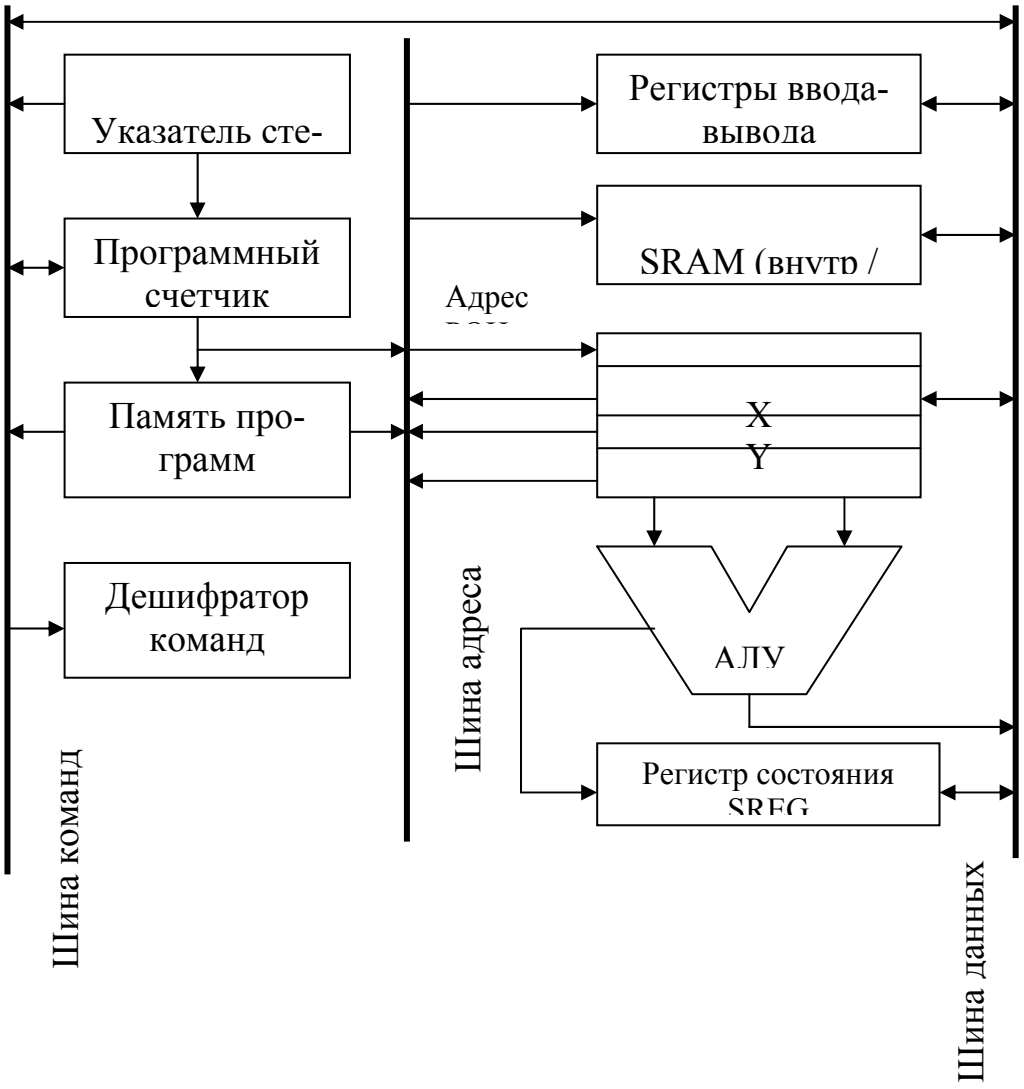


Рис. 3 Функциональная схема микроконтроллера с регистрами, доступными для программирования.

Программирование микроконтроллеров

Микроконтроллеры типа AVR серий AT90SXXXX и их модели имеют сокращённые (по сравнению с процессорами универсальных ЭВМ) наборы команд – инструкций. Это вместе с особенностями архитектуры позволяет значительно сократить число машинных тактов, затрачиваемых на исполнение команды, и повысить скорость выполнения типичных для МК программ.

Для программирования используется как **язык ассемблера**, так и языки высокого уровня (в частности, **язык Си**). Наиболее доступной интегрированной средой программирования на языке ассемблера является программный пакет **AVR Studio**. Он вклю-

чает в себя текстовый редактор, транслятор-ассемблер, отладчик-симулятор и программу загрузки Flash-памяти и EEPROM через специальное устройство-программатор. Язык ассемблера состоит из набора команд-инструкций, в которых можно использовать простые операторные выражения, и директив, используемых на этапе трансляции-ассемблирования.

Базовый **набор команд** подразделяется на арифметико-логические, передачи данных, переходов и битовые. Команда состоит из кода операции и одного или двух операндов. Ряд команд не содержит операндов. Операндом может являться: либо значение, непосредственно заносящееся в команду; либо адрес ячейки, содержащей значение, - прямая адресация; либо адрес ячейки, содержащей адрес, - косвенная адресация. Список и подробное описание команд содержится в [1-6] а также в справочном разделе среды программирования AVR Studio. Ниже рассматриваются примеры команд в таком виде, как они записываются в программе.

В **арифметических** и **логических командах** обязательным операндом является один из рабочих (операционных) регистров. В мнемокоде этот операнд указывается первым и является как регистром источником, так и регистром назначения, в который заносится результат операции (за исключением операций сравнения). В соответствии с результатом операции устанавливаются значения флагов в регистре состояния **SREG**.

add r16, r20	Сложение данных, находящихся в регистрах **r16** и **r20** ..
adc tmp, r21	Сложение данных, находящихся в регистрах **tmp** и **r20,** и ; значения бита переноса **C** в регистре флагов состояния **SREG**. (**tmp** –имя регистра, предварительно присвоенное директивой ассемблера, например, **.def tmp=r17)**.
inc tmp	Увеличение значения в регистре **tmp** на 1.
com tmp	Инвертирование разрядов (дополнение до единицы).
neg tmp	Смена знака (дополнение до двух).
cp tmp, level	Сравнение - вычитание значения **level** из значения **tmp**. Флаги **SREG** изменяются в соответствии с результатом. Операнды-регистры не изменяются.
cpi tmp, up2	Сравнение значения **tmp** и константы **up2**,предварительно объявленной директивой ассемблера (**.equ up2=170**).
ori tmp, $30	Поразрядное ИЛИ значения **tmp** и непосредственно заданного шестнадцатиричного числа. Результат помещается в регистр **tmp**.

В командах этой группы с непосредственной адресацией (два последних примера) могут использоваться только регистры с **r16** по **r31**. В остальных командах допустимы все рабочие регистры.

Команды передачи данных содержат два операнда, первый из которых – операнд назначения, второй – источник. Один из операндов должен быть рабочим регистром, другой может быть регистром, непосредственным значением, прямым или косвенным адресом данных.

mov Rd, Rr	Передача данных между регистрами. **Rd, Rr** –имена регистров.
ldi Rd, K	Загрузка регистра непосредственным значением. **K** – число или имя константы (директива **.equ**).
lds Rd, k	Прямая загрузка регистра из памяти данных SRAM с адреса **k** , заданного числом или именем (директива **.equ**).
sts k, Rd	Прямая передача байта в адрес **k** из регистра.
ld Rd, X	Косвенная загрузка регистра из памяти данных SRAM с адреса, заданного содержимым регистровой пары **X**.
ld Rd, X+	Косвенная загрузка регистра с последующим увеличением ; содержимого **X** на 1.
ld Rd, -X	Косвенная загрузка регистра с предварительным уменьшением **;** содержимого **X** на 1.
st X, Rd,	Косвенная передача содержимого регистра в ячейку памяти данных SRAM с адресом, заданным содержимым **X**.

Команды с **косвенной** адресацией и регистровой парой, примеры которых приведены выше, могут использовать любой из сдвоенных регистров- **X, Y, Z**.

В трёх следующих командах один или оба операнда отсутствуют.

lpm	Загрузка из памяти программ (FLASH) байта, адресуемого **Z**, в **R0**.
push Rr	Заталкивание содержимого регистра **Rr** в стек.
pop Rd	Выталкивание байта из стека в регистр **Rd**

Две команды выполняют передачу байта между рабочим регистром и каким-либо портом - регистром ввода/вывода.

in Rd, $19	Загрузка рабочего регистра байтом из регистра ввода вывода, заданного его номером.
out Port, Rd	Загрузка байта из рабочего регистра в порт, заданный именем.

Команды переходов предназначены для изменения последовательности выполнения операторов программы. По этим командам производится изменение указателя адреса следующей

выполняемой команды - **PC**. Ряд команд изменяет **PC** безусловно, другие – при выполнении некоторого условия. При безусловных переходах новое значение **PC** задаётся как адрес метки последовательности операторов, к выполнению которых надо перейти, значение этого адреса вычисляется в ходе ассемблирования программы.

rjmp label Безусловный относительный переход. В мнемокоде **label** – имя метки, к которой требуется перейти, в код команды заносится число, на которое изменяется **PC**.

rcall label Вызов подпрограммы (относительный). В мнемокоде **label** – метка подпрограммы. Адрес команды, следующей за командой вызова подпрограммы, заносится в стек.

ret Возврат из подпрограммы. **PC** принимает значение извлекаемого из стека адреса команды, следующей за командой вызова подпрограммы.

reti Возврат из подпрограммы обработки прерывания. **PC ;**принимает значение извлекаемого из стека адреса команды, ;следующей за командой выполненной до прерывания.

Ряд команд **условных переходов** пропускают следующую команду, а другие производят переход к метке при выполнении условия, отражающегося в названии операции. В командах второго вида условием является значение флага в регистре состояния, установленное в результате предшествующей операции.

cpse Rd, Rr Сравнение регистров и пропуск команды, если равны.

sbrc Rr, 7 Пропуск команды, если в рабочем регистре бит, заданный номером или именем, равен 0.

sbis Port, bit Пропуск команды, если в регистре ввода/вывода бит, заданный номером или именем, равен 1.

brbs 3, label Переход на метку **label** , если в регистре состояния **SREG** бит, заданный номером или именем, равен 1.

brne label Переход на метку **label** , если “не равно” (если в результате предшествующей операции в регистре состояния **SREG** флаг равенства нулю не установлен – **Z**=0).

Команды операций с битами устанавливают (логическая единица) или сбрасывают (нуль) отдельные биты регистра состояния или портов ввода/вывода. К этой же группе относятся команды сдвига бит в регистре.

bclr bit Очистить – установить равным 0 бит, заданный номером или именем, в регистре состояния **SREG**.

sez Установить в 1 флаг **Z** в регистре состояния **SREG**.

sbi Port, bit Установить в 1 бит в регистре ввода/вывода.

lsl Rd	Логический сдвиг разрядов регистра влево, старший разряд переносится **lsl** во флаг переноса **C**, в младший заносится 0.
rol Rd	Циклический сдвиг-вращение влево. В отличие от команды **lsl,** в младший разряд заносится прежнее значение **C**.
asr Rd	Арифметический сдвиг вправо семи старших разрядов – целочисленное деление на 2.

В программе на Ассемблере вместе с командами содержатся **директивы**, используемые на этапе трансляции программы в машинный код, но сами не транслируются. Выше уже упоминались директивы: **.def** , присваивающая имена регистрам, и **.equ** , присваивающая имена и значение константам. При трансляции команд вместо имён подставляются адреса или значения. Практически любая программа содержит директиву **.include** , по которой во время трансляции подключается дополнительный файл –"исходник". Например, директива **.include "8515def.inc"** подключает расположенный в той же директории, что и программа, файл, в котором как константы определены имена регистров ввода/вывода и их отдельных бит, а также имена адресов векторов прерывания. Эти имена фактически стандартны для рассматриваемых микроконтроллеров и используются как в технических описаниях, так и в примерах программ. Ниже даётся описание ещё нескольких директив.

.cseg, **.dseg,** **.eseg**	отмечают в исходном тексте начало сегментов с программным кодом, данными для программной памяти (Flash) и данными для электрически перепрограммируемой памяти (EEPROM).
.byte	резервирует указанное параметром директивы число байт во встроенной оперативной памяти данных (SRAM).
.db, **.dw**	определяют одну или последовательность одно- или двухбайтовых констант в памяти программ, данных или EEPROM в зависимости от места-сегмента нахождения директивы.
.set	присваивает имя и значение метке, имя может использоваться в выражениях Ассемблера, значение может быть изменено другой такой же директивой.
.org	присваивает значение, заданное параметром, программному счетчику **PC** или счетчику положения в SRAM.

В Ассемблере в качестве одного из операндов команды может использоваться **выражение**. Выражение может содержать операнды, операторы и функции. В роли операндов могут быть метки, переменные и константы, введённые с помощью директив, а также непосредственные значения. Тип операций – арифметические, логические и отношения. Функции выполняют простей-

шие действия такие, как выделение байтов или слов из многобайтовых выражений, вычисление степени и логарифма с основанием 2. Более подробное описание директив и выражений имеется в AVR Studio.

Задание по лабораторной работе "Изучение архитектуры и основ программирования микроконтроллеров"

1. Следуя приведенным ниже указаниям (см. раздел Работа над проектом), выполните действия по разработке и загрузке в память МК простой программы, выполняющей инкрементирование значения, находящегося в одном из регистров PORTB, реализуя тем самым двоичный счет. Наблюдайте работу программы по изменениям состояний светодиодов, подключенных к этому порту.
2. Модифицируйте разработанную программу так, чтобы через PORTA прочесть состояния переключателей на плате лабораторного макета и вывести через регистры PORTB эти данные, так, что включенное состояние светодиодов соответствовало бы переключателям, подключающим входы МК к напряжениям, соответствующим логическим единицам. Считая, что младший бит находится слева, определите по светодиодам закодированное переключателями число (номер вашего варианта). Указание: для устойчивой работы порта на чтение данных инициализируйте его как входное устройство, подключенное к верхнему ТТЛ уровню.
3. Модифицируйте программу работу МК так, чтобы через светодиоды выводилось число, равное номеру варианта плюс число вашего рождения. Пример: вариант №13, а родились Вы 26 мая. На светодиодах должно появиться число 100111, где 1 - означает включено, а 0 – выключено (четвертый и пятый справа светодиоды не горят).
4. Напишите и отладьте программу, обрабатывающую прерывание таким образом, чтобы полоска светодиодов зажигалась последовательно слева-направо при нажатии кнопки INT1 и справа-налево при нажатии кнопки INT2.

Работа над проектом

Разработка программы. Для выполнения этого примера вам потребуется создать папку "***Code***" на диске **C:**. В эту папку вы должны поместить файл "***8515def.inc***", который содержит все установки микроконтроллера, необходимые для компиляции. Этот файл включен в комплект приложений документации и ПО AVR000.

Шаг 1 – Запуск программы AVR Studio. Запустите на выполнение программу AVR Studio. После запуска на экране появится окно, подобное изображенному на рис. 4.

Шаг 2 – Создание нового проекта

В данном примере продемонстрировано создание простой программы, выполняющей инкрементирование значения, находящегося в одном из регистров PORTB, реализуя тем самым двоичный счет.

Чтобы создать новый проект, выберите пункт "**New**" функции "**Project**" в соответствующем разделе главного меню. На экран будет выведено диалоговое окно, показанное на рис. 4.

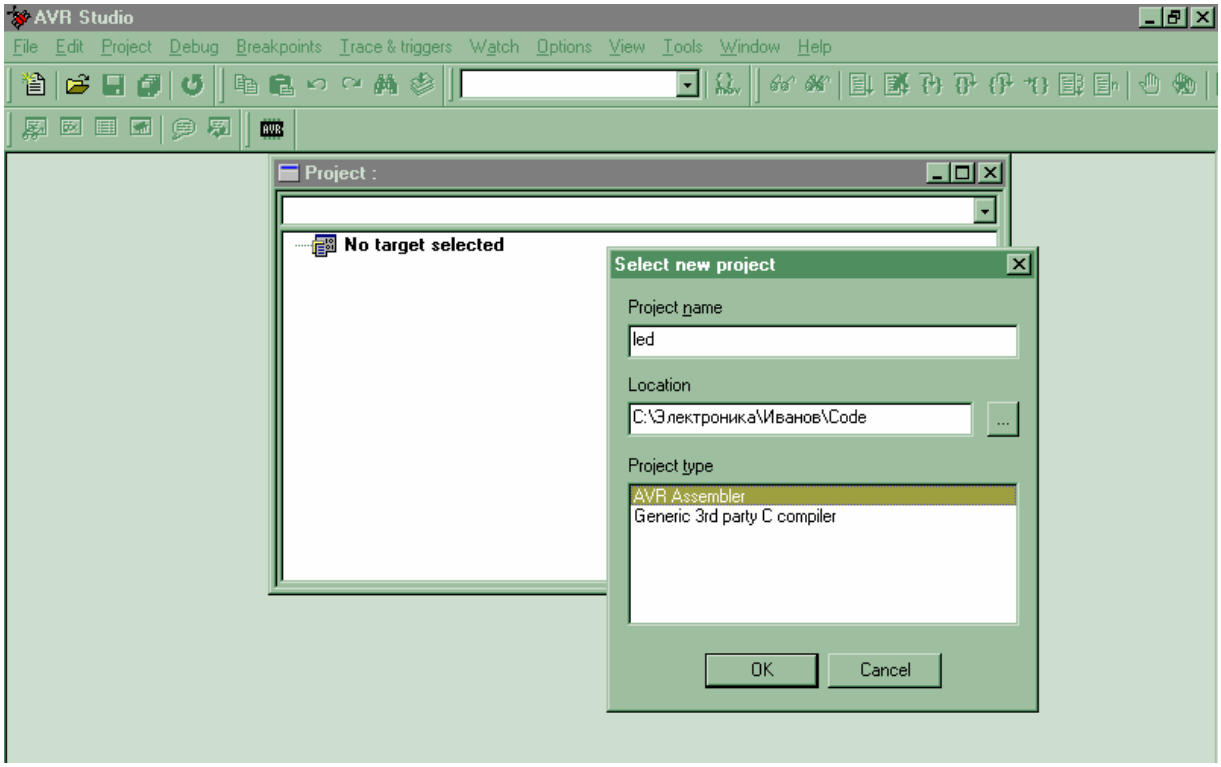


Рис. 4. Окно программы AVR Studio в начале работы над проектом.

В этом диалоговом окне вы должны задать имя проекта (***led***). Вы также должны указать расположение проекта в файловой системе: ***C:\Code***. Если соответствующей папки нет, AVR Studio создаст ее автоматически.
Далее следует выбрать тип проекта:

- ***AVR Assembler:*** в этом случае AVR Studio использует для компиляции проекта программу Ассемблер. Никаких дополнительных действий пользователя больше не потребуется. Этот вариант используется в данном примере.
- ***Generic 3rd party C compiler:*** в этом случае требуется дополнительно конфигурировать AVR Studio, применяя программы внешнего компилятора и компоновщика связей проекта.

Для продолжения работы нажмите {OK}.Диспетчер проекта выдаст на экран новое окно проекта (рис. 5). В нем будут показаны все файлы, связанные с данным проектом. В данном случае таких файлов нет.

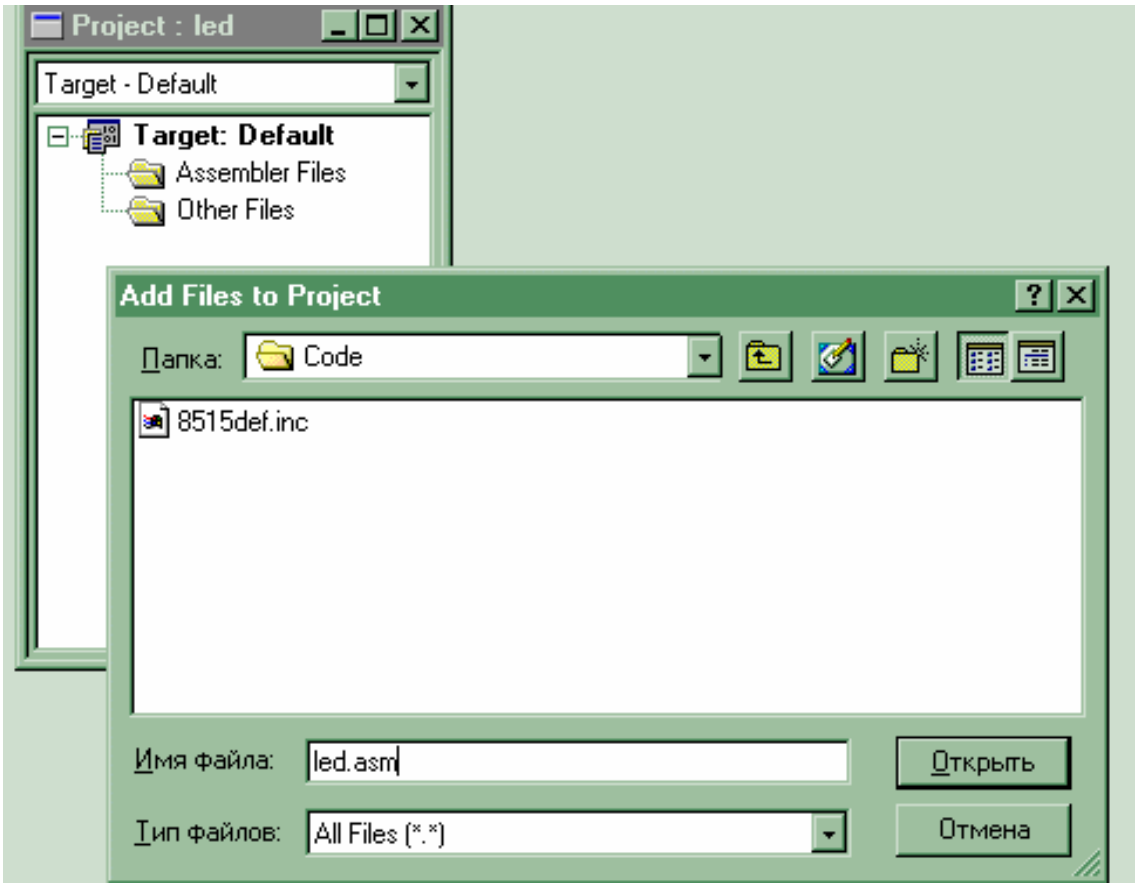


Рис. 5. Добавление файла **`led.asm`** в проект

Шаг 3 – Создание файла с кодом программы на ассемблере

Теперь следует включить в проект файл с ассемблерным кодом. Это можно сделать, редактируя уже существующий файл или создавая новый. Если вы хотите написать новый программный код, AVR Studio автоматически создаст соответствующий файл. Для этого следует в окне проекта выделить строку "**Assembler Files**" и затем в меню **"Project"** выбрать опцию "**Add Files**". Появится стандартное диалоговое окно открытия файла, позволяющее указать путь к файлу и его имя. Обратите внимание, что вы должны явно определить тип файла как ***.asm***. Если вы хотите открыть уже существующий файл, то найдите его в файловой системе и дважды щелкните левой кнопкой мыши на соответствующей пиктограмме этого окна.

В результате после подтверждения "**Открыть**" в окне на рис. 5. будет создан файл ***led.asm*** и его пиктограмма отобразиться в окне проекта как это показано на рис.6. Он будет автоматически отмечен как "Assembler Entry File" (ассемблерный входной код). Это указание определяет, что данный файл служит начальным файлом для ассемблирования программы. Папка "Assembler Files" должна содержать только один файл, отмеченный как входной. Текущий входной файл отмечен в соответствующем окне пиктограммой с красной стрелкой, направленной вправо, другие файлы отмечены голубой стрелкой, направленной вниз.

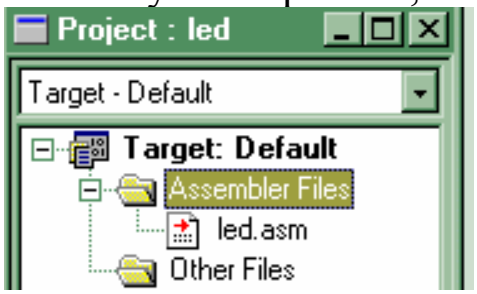


Рис. 6. Окно проекта с указанием на файл с исходным входным ассемблерным кодом программы.

Шаг 4 – Редактирование ассемблерного файла с кодом программы

В результате предыдущих действий мы добавили новый пустой файл в проект. На следующем этапе следует заполнить его кодом программы. Откройте файл "***led.asm***" для редактирования, дважды щелкнув левой кнопкой мыши на его пиктограмме. В результате откроется окно встроенного редактора (рис. 7).

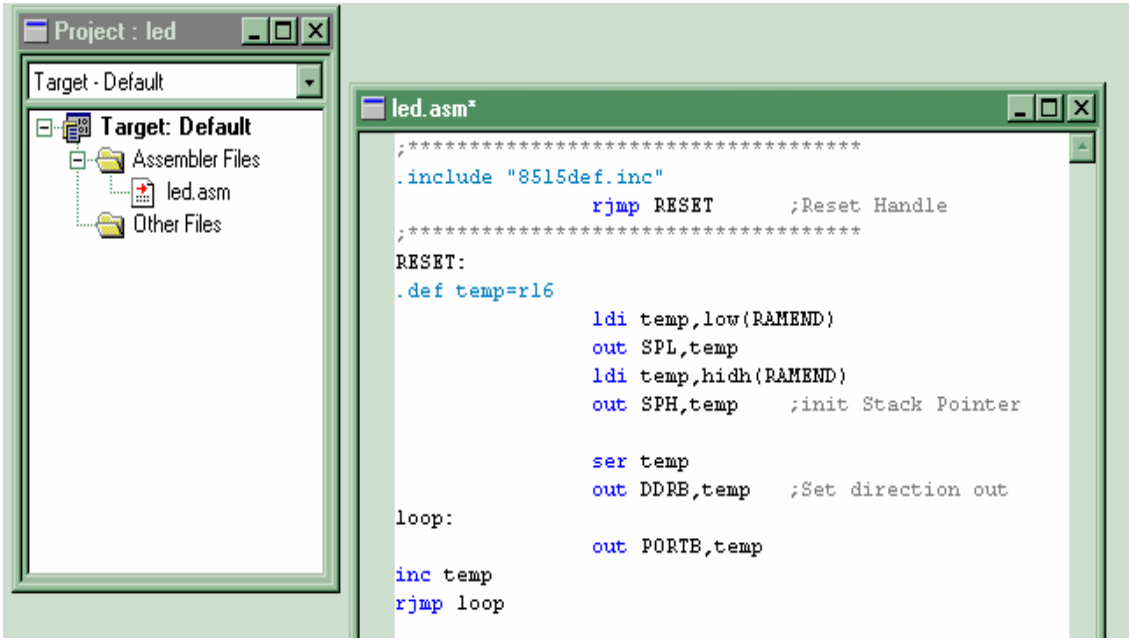


Рис. 7. Окно редактирования кода программы.

Файл вначале пустой и вам следует ввести в него программный код, воспользовавшись обычными приемами редактирования. Напомним, что удобно использовать процедуру копирования и вставки из буфера (кнопки {Ctrl+C} и {Ctrl+V}).

```
;ПРОГРАММА ЦИКЛИЧЕСКОГО СЧЕТЧИКА С ВЫВОДОМ В ПОРТ
;
;.include "8515def.inc"
.def     tmp=r16      ; Директивы Ассемблера,
.def     tmp1=r17 ; присваивющие символические
.def     tmp2=r18 ; имена рабочим регистрам
;
.cseg
.org 0x00     ; Директива установки адреса
rjmp reset    ; последующего программного кода
;
; ПОДПРОГРАММА ВРЕМЕННОЙ ЗАДЕРЖКИ
delay:
ldi  tmp1, 0xff
d0:  ldi  tmp2, 0xff
d2:  dec  tmp2
brne d2
dec  tmp1
     brne d0
     ret
```

```
;
; ГОЛОВНАЯ ПРОГРАММА
reset:
ldi  r16, high(RAMEND); Установка указателя стека
out  SPH, r16         ; на конец памяти программ
ldi  r16, low(RAMEND)
out  SPL, r16
;
ser  tmp            ; установка битов регистра
out  DDRB, tmp      ; Установка порта B на вывод
loop:               ; Цикл
out  portB, tmp     ; Вывод содержимиго tmp в порт B
inc  tmp            ; Прибавление единицы
rcall    delay      ; Задержка
rcall    delay
rjmp loop           ; Возврат на метку цикла
```

Шаг 5 – Ассемблирование исходного кода программы

Следующий шаг заключается в создании машинного кода из исходного текста программы. Для выполнения этой процедуры следует выбрать опцию "Assemble" в меню "Projects" или нажать клавишу {F7} (см. рис. 8).

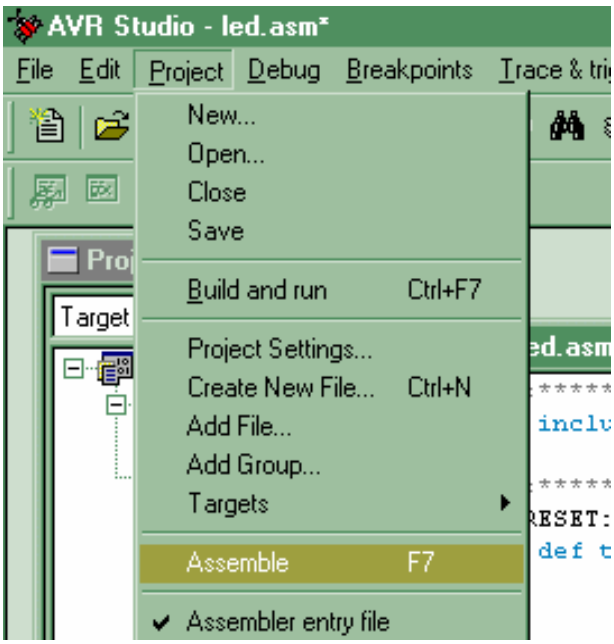


Рис. 8. Пункты горизонтального меню"Project", задающие функции работы над проектом.

После выполнения программы ассемблирования в окне "Project Output" будет выведена информация о результатах. На

рис. 9 в окне вывода результатов компиляции показано, что обнаружена ошибка. Как видно из рисунка, если в этом окне выделить строку с указанием ошибки щелкнуть на ней мышью, то автоматически соответствующая данной ошибке строка кода будет выделена в окне редактора.

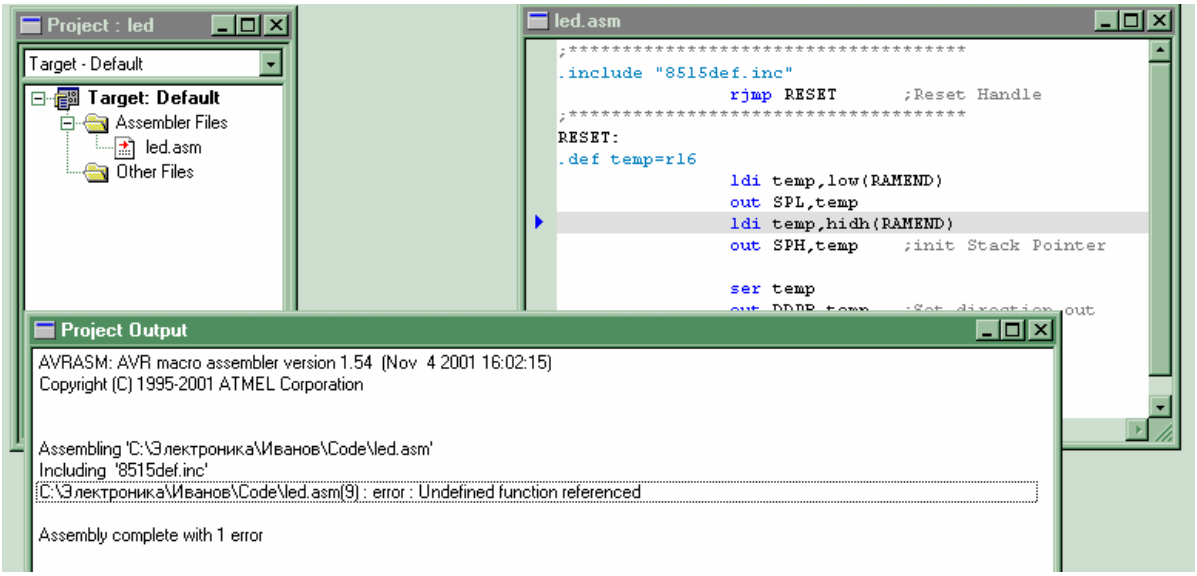


Рис. 9. Окна результатов компиляции и редактирования с указанием на ошибку.

После исправления ошибки следует снова запустить ассемблер. В новом окне результатов (рис. 10), указано, что код содержит 10 слов (т.е. 20 байт) и программа-ассемблер выполнена без ошибок. В результате все подготовлено к тому, чтобы перейти к следующему этапу проектирования, на котором проводится отладка программы в режиме симулятора.

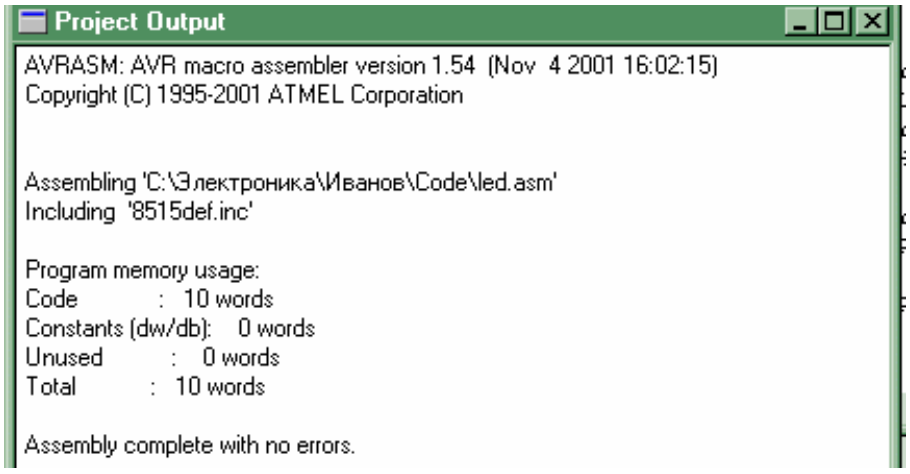


Рис. 10. Окно с результатами работы Ассемблера.

Отладка кода программы

Если файл с программным кодом подготовлен, то его можно запустить в режиме симулятора, выбрав в меню "Debug" пункт "trace into" или нажав клавишу {F11}. О переходе в этот режим свидетельствует выделение первой строки кода в окне редактирования программы (см. рис. 11). Желтая стрелка слева указывает на инструкцию, которая будет выполняться на следующем шаге в соответствии с содержимым счетчика команд PC.

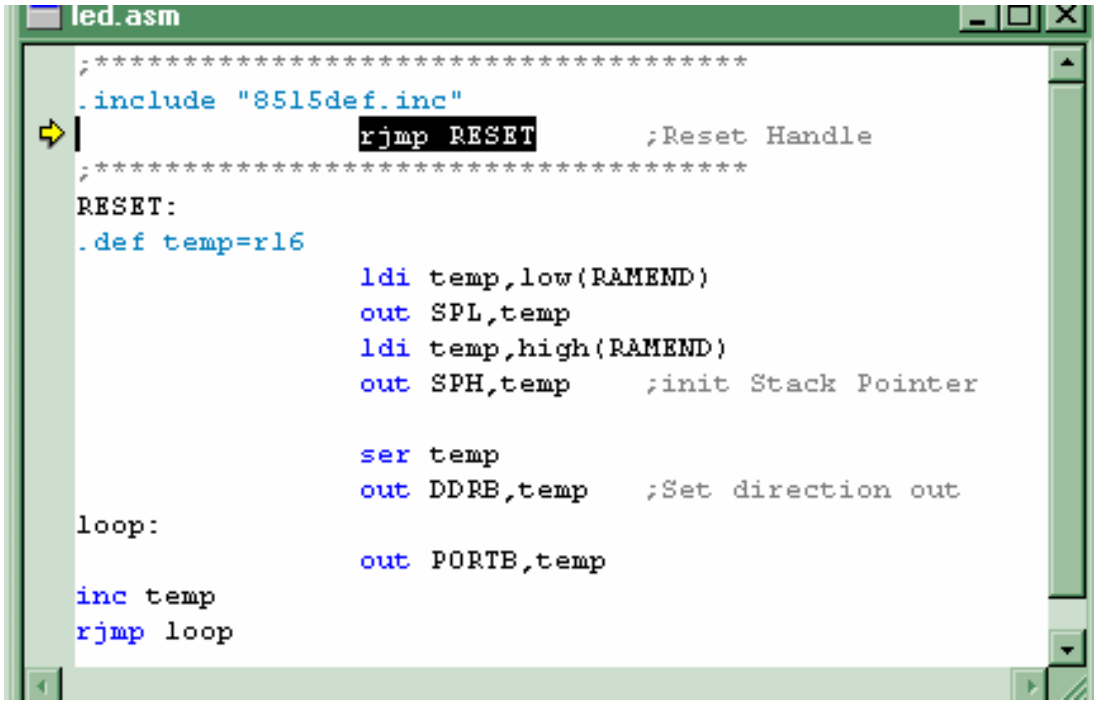


Рис. 11. Указатель положения счетчика команд в окне с кодом программы.

Процесс отладки заключается в отслеживании и управлении хода выполнения программы с помощью окна редактирования и специальных окон, показывающих состояния рабочих регистров (Registers), регистров ввода/вывода (I/O), окна наблюдения за переменными (Watch), регистров процессора и содержимого памяти. Эти окна могут быть выведены на экран из соответствующего пункта меню "**View**".

Выбор устройства

В режиме симулятора доступны подпункты меню "**Option**". В окне "Simulator Options" (рис. 12), следует назначить устройство, для которого предназначен программный код. Пусть

в данном случае это будет AT90S8515. Оставьте значение рабочей частоты и щелкните "OK". Так как выбрано стандартное устройство, то параметры памяти и архитектура не изменяются. Эти функции применяются в режиме настройки параметров устройства "Custom" (Специальный). В режим симулятора можно также войти, выбрав "Build and run" в меню "Project".

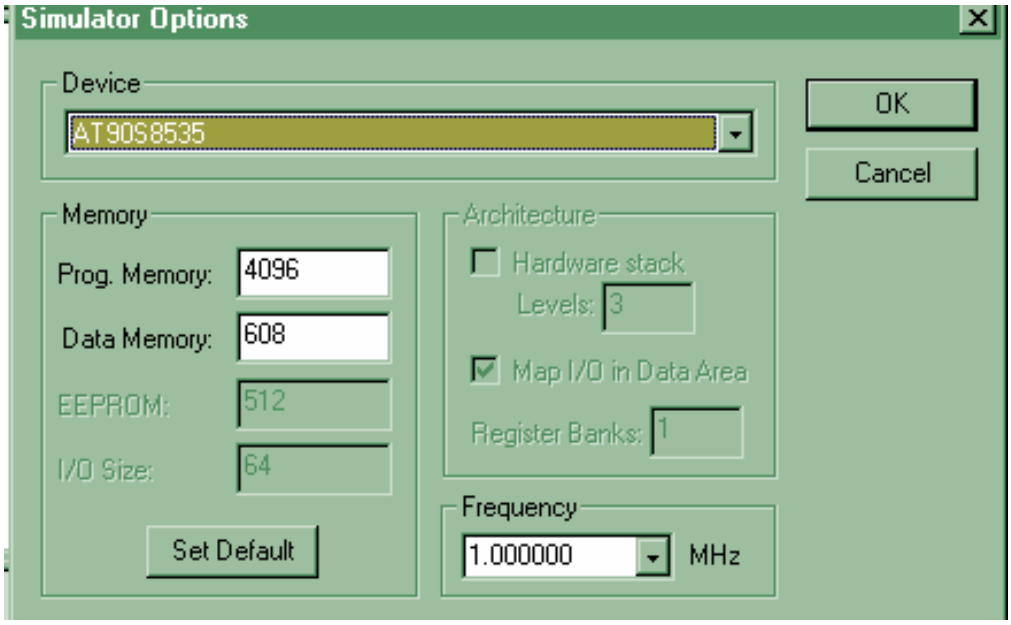


Рис. 12. Окно назначения параметров устройства.

Настройка окна ввода-вывода

Откройте окно "IO", щелкнув на пиктограмме "IO Window"(или соответствующий пункт меню "View"). Поскольку симулятор уже настроен для работы с типом микроконтроллера AT90S8515, то соответствующие элементы его архитектуры отобразятся в этом окне автоматически (см. рис. 13).

Щелчок на значке + у строчки с надписью PORTB открывает связанные с портом ввода-вывода B (см. рис.14): регистр данных порта B (PORT B DATA), регистр управления направлением передачи порта Data Direction (DDRB) и регистр драйверов порта Input Pins (PINB). Состояние каждого бита этих регистров отображается в соответствующей позиции окна: логическому нулю ("0") соответствует пустая позиция, а логическая единица ("1") отображается галочкой. Эти позиции окна могут изменяться в ходе выполнения программы, показывая текущее состояние каждого бита. В ходе выполнения программы

вы также можете сами установить или сбросить эти биты, щелкая левой кнопкой мыши на соответствующем поле.

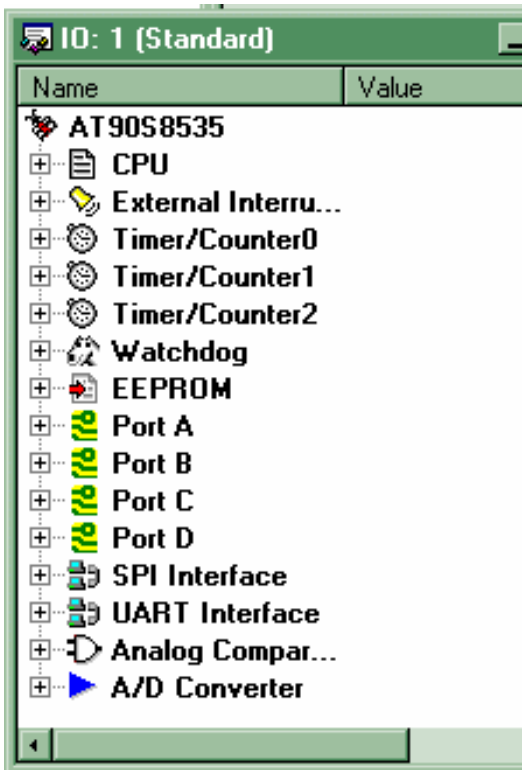


Рис. 13. Окно устройств ввода/вывода микроконтроллера AT90S8535.

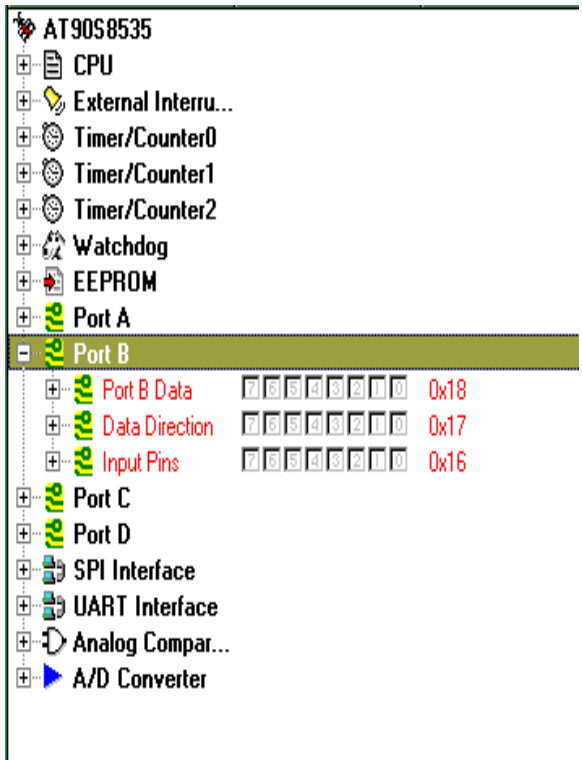


Рис. 14. Содержимое регистров порта B.

Пошаговое выполнение программы

Имеются две команды, управляющие пошаговым выполнением программы: "Step Over" {F10} и "Trace Into" {F11}. Различие между этими командами состоит в том, что активизация {F10} позволяет не отслеживать ход выполнения программы внутри подпрограмм. В рассматриваемом примере нет подпрограмм и, следовательно, нет различия при выборе одной из этих клавиш.

Теперь перейдите к последней строке кода, нажимая несколько раз клавишу {F11} или выбрав опцию "Trace Into" из меню "Debug". Обратите внимание, что при изменении состояний регистров их цвета изменяются с черного на красный (см. рис. 15). Продолжая нажимать на кнопку {F11}, наблюдайте, как увеличивается значение данных в порту B.

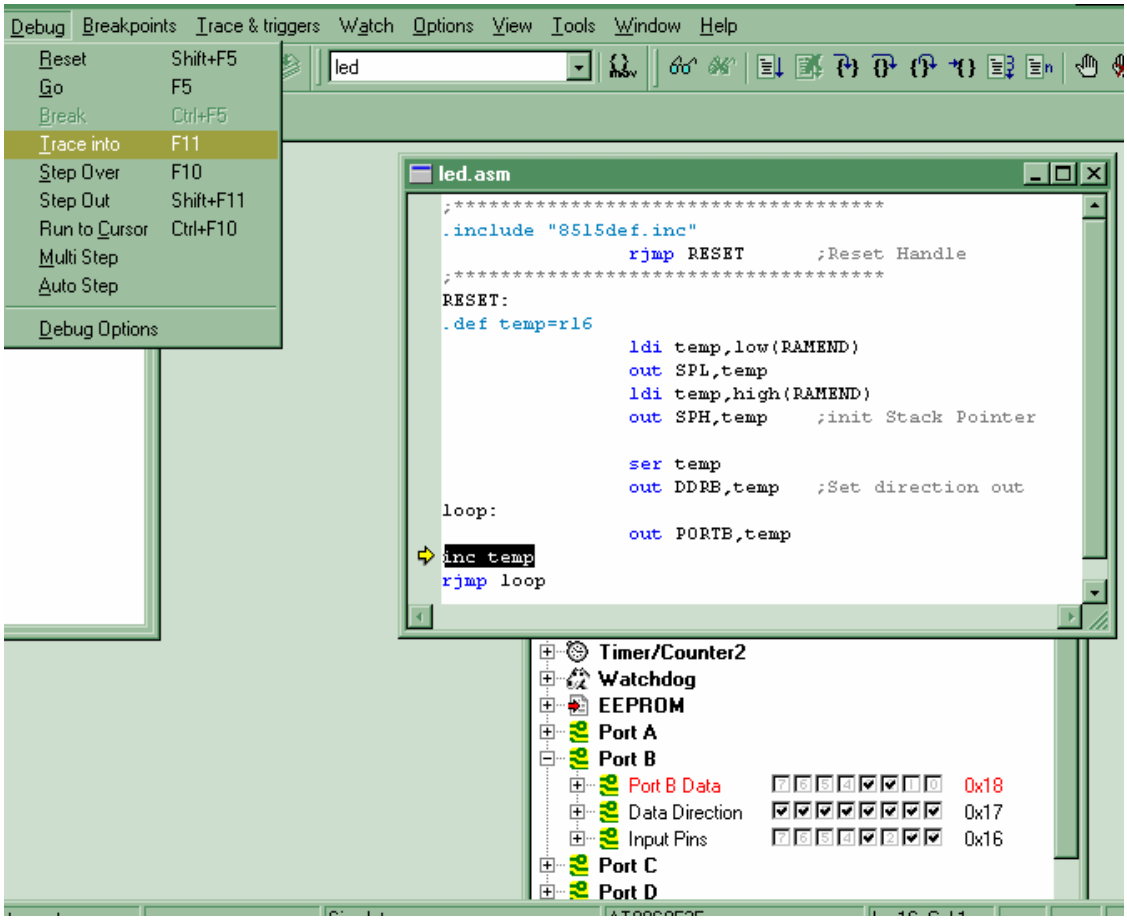


Рис. 15. Просмотр содержимого регистров PORTB в режиме пошагового выполнения программы.

Назначение точек остановки

Точки остановки позволяют остановить ход выполнения программы. Указывая точки остановки в ассемблерном коде программы, вы имеете возможность остановить ее выполнение в соответствующем месте (см. рис. 16).

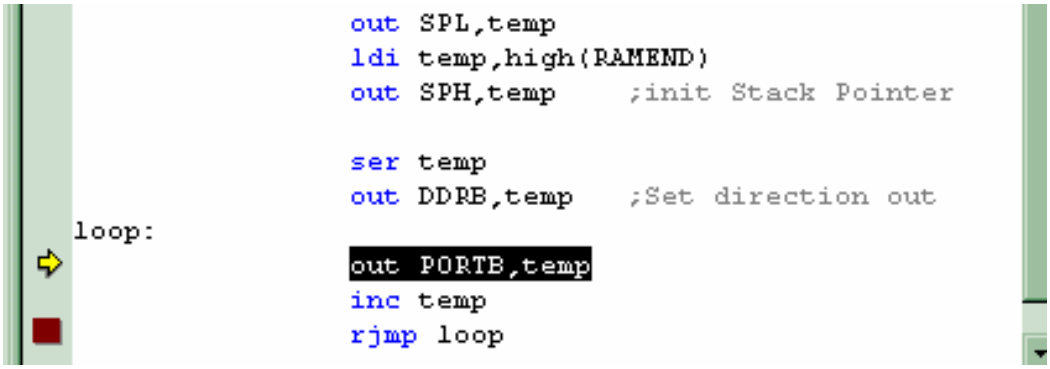


Рис. 16. Индикатор точки останова программы.

Требуется три раза нажать на кнопку {F11}, чтобы выйти из цикла. При желании вы можете ускорить ход выполнения программы при отладке. Для этого установите точку останова напротив инструкции "`rjmp`", используя клавишу {F9} (или пункт "Toggle Breakpoint" меню "Breakpoint"). Красный квадрат, появившийся в полях слева от кода отмечает установленное вами место остановки. После нажатия клавиши {F5} или выбора пункта "Go" в меню "Debug", программа начнет выполняться и остановится перед выполнением отмеченной вами команды.

Изменение программного кода

Пусть требуется изменить программу, например, сделать так, чтобы значения в выходном порту PORTB декрементировались. Чтобы отредактировать соответствующим образом первоначальный код программы, поместите курсор в нужное место кода и измените инструкцию "`inc`" на "`dec`". Если теперь нажать {F5}(Go), то появится диалоговое окно, которое указывает, что один из исходных файлов проекта был изменен, и требуется новая компиляция и перекомпоновка проекта. Выберите кнопку "**Yes**". Проект будет снова скомпонован и программный счетчик установится напротив первой строки кода. При этом положение точки остановки программы сохраняется.

Открытие окна просмотра

Окно просмотра значений переменных открывается выбором соответствующего пункта меню "**Veiw**"- "**Watch**" или пиктограммы.

Переменные, которые были определены в программе (с помощью директивы `.def`) могут быть выбраны для отражения их содержания в окне просмотра значений. В рассматриваемом примере такая переменная только одна – "`temp`". Если дважды щелкнуть кнопкой на ней в окне кода, то эта переменная будет выделена. Ее можно перенести методом "держи и тащи" в окно для просмотра значений. Значение этой переменной будет отображаться в этом окне по ходу выполнения программы (см. рис. 17).

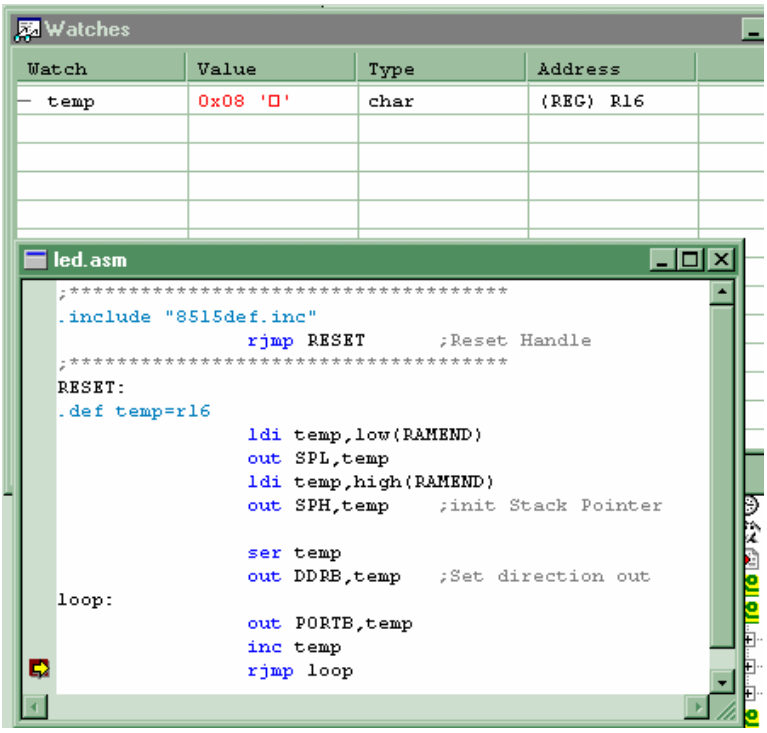


Рис. 17. Окно просмотра значений переменных, определенных в программе.

Установка окна просмотра состояния процессора

На рис. 18 изображено окно просмотра состояния процессора.

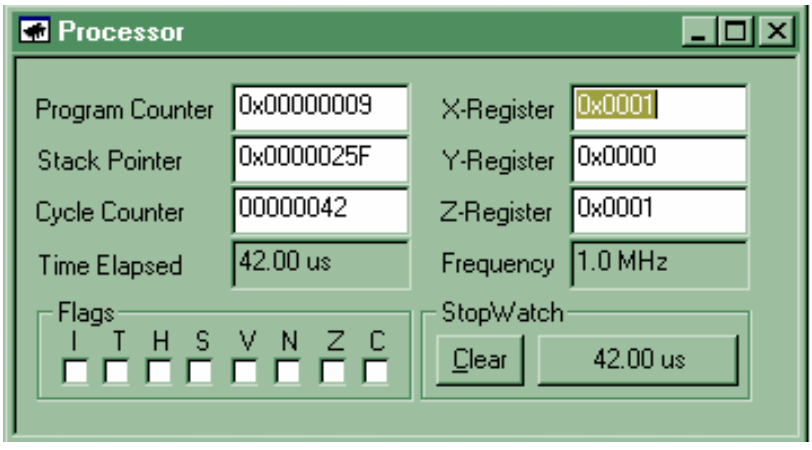


Рис. 18. Окно "Processor", показывающее состояние регистров процессора в ходе выполнения программы.

Его можно открыть, выбрав соответствующую пиктограмму (или "View"- "Processor") Это окно отображает состояние регистра флагов "Flag" и текущие значения различных

счетчиков. Дополнительно в нем можно наблюдать состояние счетчика циклов (Cycle Counter) и сторожевого таймера (StopWatch). Этим окном удобно пользоваться, если требуется определить длительность выполнения цикла или время выполнения подпрограммы. В данном примере нет необходимости использовать это окно.

Аналогично можно по ходу выполнения программы просматривать содержимое регистров, памяти данных и других элементов микроконтроллера.

Сохранение проекта

Перед окончанием работы AVR Studio предложит сохранить ваш проект. AVR Studio запомнит все открытые вами окна и их настройки и воспроизведет их при следующем открытии проекта. Сохранение проекта осуществляется выбором пункта "Save" в меню "Project".

Загрузка программы в ПЗУ

После того как прикладная программа отлажена на симуляторе, ее можно загрузить в память микроконтроллера. Для этой цели служит программа и устройство - программатор. Для загрузки предварительно должен быть сформирован файл загрузки (***.hex**-файл).

Плата контроллера содержит элементы, обеспечивающие внутрисистемное программирование контроллера через последовательный порт RS-232 персонального компьютера. Для этого следует воспользоваться одной из специальных программ, например, программой прошивки, включенной в комплект Astudio. Переход в режим программирования производится активизацией строки **AVRPrg** пункта меню "**Tools**".

При правильном соединении компьютера и контроллера после включения питания на экране появляется диалоговое окно программы прошивки. В элементах этого окна следует указать тип программируемого МК, адрес файла с шестнадцатеричным кодом (***.hex**) загружаемой программы, после чего выбрать кнопку **Programming** в группе элементов управления **Flash**. Программа, осуществляющая загрузку памяти МК (прошивку) последовательно производит операции стирания информации

(запись единицы во все биты памятей программ и данных)запись значений из hex-файла и проверку соответствия записанной в память программы и исходной. Ход процесса прошивки отображается в строке состояния группы элементов **Flash.** Успешное завершение процесса отмечается появлением сообщения "OK" в строке состояния.

После завершения процесса программирования процессор готов к работе. Окно программатора может оставаться активным во время работы ассемблера/симулятора, что дает возможность оперативно модифицировать программу.

Пример кода программы, реализующей вывод на светодиоды результата (n^2+1), где n – число, закодированное положениями переключателя:

```
.include "8515def.inc"
.def temp=r16
.def a=r20
.def b=r21
.def i=r22

.org 0
     rjmp start

start:
     ldi  temp,LOW(RAMEND)    ;инициализация указателя стека
для AVR,
     out  spl,temp       ;имеющих програмный стек, ОБЯЗАТЕЛЬНО
выполняется
     ldi  temp,HIGH(RAMEND)   ;в начале программы
     out  sph,temp

     rcall     port_init      ;инициализация портов

;основной цикл программы
loop:
     in   temp,pina      ;значение (pinA)->temp
     rjmp umnogenie      ;temp=(temp)^2
follow:   inc  temp           ;temp=temp+1
     com  temp           ;инвертируем ответ,
     out  portb,temp          ;чтобы "1" отображалась
светящимся диодом
     rjmp loop           ;бесконечный цикл

;подпрограмма инициалиции портов
port_init:
     ldi  temp,0xff
```

```
      out  ddrb,temp
      out  ddra,temp
      ldi  temp,0x00
      out  portb,temp
      ret

;подпрограмма умножения axb, где b=a; aєN; bєN
umnogenie:
      mov  a,temp
      mov  b,temp              ;т.к. в данном примере b=a
      push b
      tst  b
      brne go
      rjmp exit                ;если a=0
go:   ldi  i,0x01
      push b
      sub  b,i
      brne next
      rjmp exit
next:     pop  b               ;если a=1
      add  temp,a
      inc  i
      push b
      sub  b,i
      brne next           ;если a>1
exit:
      rjmp follow
```

Библиографический список

[1.] Изделия и компоненты, предлагаемые фирмой "КТЦ-МК". Микроконтроллеры фирмы "ATMEL" семейства AVR. Справочник. 2-е изд. – М.: КТЦ-МК, 1999. – 299 с.

[2.] Бродин В.Б., Калинин А.В. Системы на микроконтроллерах и БИС программируемой логики. - М.: Издательство ЭКОМ, 2002 – 400 с.

[3.] Предко М. Руководство по микроконтроллерам. В 2 т. - М.: ЗАО "Предприятие Постмаркет", 2000. –2 т.

[4.] Евстифеев А.В. Микроконтроллеры семейства Classic фирмы "ATMEL". М.: Издательский дом "Додэка-XXI", 2002. – 288 с.

[5.] Фрунзе А.В. Микроконтроллеры? Это же просто! В 2 т. – М.: ООО "ИД СКИМЕН", 2002. – 2 т.

[6.] Гребнев В.В. Микроконтроллеры семейства AVR фирмы ATMEL. М.: ИП РадиоСофт, 2002. – 176 с.

[7.] AVR. Atmel Corporation 8-bit RISC Microcontrollers Data Book. August 1999. Atmel Corporation, San Jose, CA 95131, USA. 1999.