Лянторский нефтяной техникум
(филиал) федерального государственного бюджетного образовательного учреждения
высшего образования «Югорский государственный университет»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по выполнению практических работ

по дисциплине ОП.06 Микропроцессорная техника

специальность 15.02.14 Оснащение средствами автоматизации технологических процессов и производств (по отраслям)

Содержание

	СТР.
ПОЯСНИТЕЛЬНАЯ ЗАПИСКА	4
ПРАВИЛА ВЫПОЛНЕНИЯ ПРАКТИЧЕСКИХ РАБОТ	5
ПЕРЕЧЕНЬ ПРАКТИЧЕСКИХ РАБОТ	6
ПРАКТИЧЕСКАЯ РАБОТА № 1	7
ПРАКТИЧЕСКАЯ РАБОТА № 2	9
ПРАКТИЧЕСКАЯ РАБОТА № 3	11
ПРАКТИЧЕСКАЯ РАБОТА № 4	14
ПРАКТИЧЕСКАЯ РАБОТА № 5	17
ПРАКТИЧЕСКАЯ РАБОТА № 6	28
ПРАКТИЧЕСКАЯ РАБОТА № 7	39
ПРАКТИЧЕСКАЯ РАБОТА № 8	41
ПРАКТИЧЕСКАЯ РАБОТА № 9	43
ПРАКТИЧЕСКАЯ РАБОТА № 10	45
ПРАКТИЧЕСКАЯ РАБОТА № 11	48
Список литературы	50

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Методические указания по выполнению практических работ предназначены для упорядочения работы обучающихся, разработаны на основе Федерального государственного образовательного стандарта среднего профессионального образования по специальности 15.02.14 Оснащение средствами автоматизации технологических процессов и производств (по отраслям).

Структура методических указаний определена последовательностью изучения дисциплины Микропроцессорная техника, которая входит в общепрофессиональный цикл специальности.

Программой дисциплины Микропроцессорная техника предусмотрено выполнение практических работ в количестве 22 часов.

Практические занятия служат связующим звеном между теорией и практикой. Они необходимы для закрепления теоретических знаний, полученных на уроках теоретического обучения, а так же для получения практических знаний. Практические задания выполняются студентом самостоятельно, с применением знаний и умений, полученных на уроках, а так же с использованием необходимых пояснений, полученных от преподавателя при выполнении практического задания

В результате освоения дисциплины обучающийся должен уметь:

- выполнять анализ и синтез электронных схем с МПС;
- вести проектирование и расчет электронных устройств с помощью ЭВМ

В результате освоения дисциплины обучающийся должен знать:

- принципы построения электронных устройств на основе современной элементной базы и МПС;
- принципы функционирования электронных устройств на основе современной элементной базы и МПС;
- основные технические параметры, эксплуатационные характеристики области
- применения основных устройств и функциональных узлов электроники и МПС;
- основные принципы проектирования схем на базе МПС.

В результате освоения дисциплины обучающийся должен обладать общими компетенциями, включающими в себя способность:

ОК1. Выбирать способы решения задач профессиональной деятельности, применительно к различным контекстам.

ОК2. Осуществлять поиск, анализ и интерпретацию информации, необходимой для выполнения задач профессиональной деятельности.

ОК.04 Работать в коллективе и команде, эффективно взаимодействовать с коллегами, руководством, клиентами.

ОК.05 Осуществлять устную и письменную коммуникацию на государственном языке с учетом особенностей социального и культурного контекста.

ОК 9. Использовать информационные технологии в профессиональной деятельности.

ОК 10. Пользоваться профессиональной документацией на государственном и иностранном языках.

В результате освоения дисциплины обучающийся должен обладать профессиональными компетенциями, соответствующими видам деятельности:

ПК 1.3. Проводить виртуальное тестирование разработанной модели элементов систем автоматизации для оценки функциональности компонентов.

ПК 2.3. Проводить испытания модели элементов систем автоматизации в реальных условиях с целью подтверждения работоспособности и возможной оптимизации.

ПК 3.3 Разрабатывать инструкции и технологические карты выполнения работ для подчиненного персонала по монтажу, наладке и техническому обслуживанию систем и средств автоматизации.

ПК 4.3. Организовывать работы по устранению неполадок, отказов оборудования и ремонту систем в рамках своей компетенции.

ПРАВИЛА ВЫПОЛНЕНИЯ ПРАКТИЧЕСКИХ РАБОТ

Практические работы проводятся в ходе осуществления учебного процесса и направлены на закрепление теоретического материала. Практические работы оформляются в письменном виде, преподаватель проверяет отчет студента о выполненной практической работе и делает отметку в журнале учебных занятий.

К каждому практическому занятию преподавателями разработаны инструкции по их проведению. В содержании инструкции каждой практической работы даны краткие теоретические сведения или формулы, примеры решения задач, и задания для самостоятельного решения по вариантам. Практические работы необходимо выполнять в тетрадях с указанием номера, темы, целей работы.

Перед выполнением практической работы преподаватель проверяет готовность студентов к ее выполнению. Преподаватель контролирует выполнение практической работы в соответствии с инструкцией по проведению. Неподготовленные обучающиеся к выполнению работы не допускаются.

Изучая теоретическое обоснование, студент должен знать, что основной целью изучения теории является умение применять ее на практике.

После выполнения работы студент должен представить отчет о проделанной работе с полученными результатами и устно ее защитить.

При отсутствии студента по неуважительной причине выполняет работу самостоятельно во внеаудиторное время и защищает на консультации.

Неаккуратно выполненная практическая работа возвращается для доработки.

Показатели оценки практической работы по дисциплине:

- умение обучающегося использовать теоретические знания при выполнении практических заданий;
- уровень освоения обучающимися учебного материала;
- правильность и четкость изложения ответа;
- оформление материала в соответствии с требованиями.

Критерии оценивания практических работ

Отметка «5» ставиться, если:

- работа выполнена полностью;
- в логических рассуждениях и обосновании решения нет пробелов и ошибок;
- в решении нет математических ошибок (возможна одна неточность, описка, не являющаяся следствием незнания или непонимания учебного материала).

Отметка «4» ставится, если:

- выполнено 75-90% заданий;
- либо работа выполнена полностью, но обоснования шагов решения недостаточны;
- допущена одна ошибка или два-три недочета в выкладках, рисунках, чертежах или графиках (если эти виды работы не являются специальным объектом проверки).

Отметка «3» ставиться, если:

- выполнено 51-75% заданий;
- допущены более одной ошибки или более двух-трех недочетов в выкладках, чертежах или графиках, но учащийся владеет обязательными умениями по проверяемой теме.

Отметка «2» ставится, если:

- выполнено менее 50% заданий;
- допущены существенные ошибки, показавшие, что учащийся не владеет обязательными умениями по данной теме в полной мере.

ПЕРЕЧЕНЬ ПРАКТИЧЕСКИХ РАБОТ

№	Наименование темы	Наименование практических работ	Форма контроля	Кол-во часов	Формируемые ОК, ПК
1	**Тема 1.4 Система команд микропроцессора**	Линейное программирование математических операций	оценка выполнения заданий, устный опрос	1	ОК 01, ОК 02, ОК 09, ОК 10, ПК 1.3, ПК 1.4, ПК 2.3, ПК 4.3
2		Ветвления на выбранном языке программирования	оценка выполнения заданий, устный опрос	1	
3		Организация циклов на языке программирования	оценка выполнения заданий, устный опрос	2	
4	**Тема 2.1 Принципы формирования адресного пространства**	Передача данных	оценка выполнения заданий, устный опрос	2	
5	**Тема 2.2 Система адресации**	Изучение приемов работы со стеком	оценка выполнения заданий, устный опрос	4	
6	**Тема 2.7 Поддержка многозадачности**	Последовательная и параллельная передача информации на языке программирования	оценка выполнения заданий, устный опрос	2	
7	**Тема 2.8 Программы-отладчики**	Работа с массивами на языке программирования	оценка выполнения заданий, устный опрос	2	
8		Написание программ с использованием подпрограмм	оценка выполнения заданий, устный опрос	2	
9		Реализация математических операций на языке программирования	оценка выполнения заданий, устный опрос, оценка	2	
10		Создание программного продукта	оценка выполнения заданий, устный опрос, оценка	2	
11		Комплексная работа по программированию на языке программирования	оценка выполнения заданий, устный опрос, оценка	2	

ПРАКТИЧЕСКАЯ РАБОТА №1

ТЕМА: Линейное программирование математических операций

ЦЕЛЬ РАБОТЫ

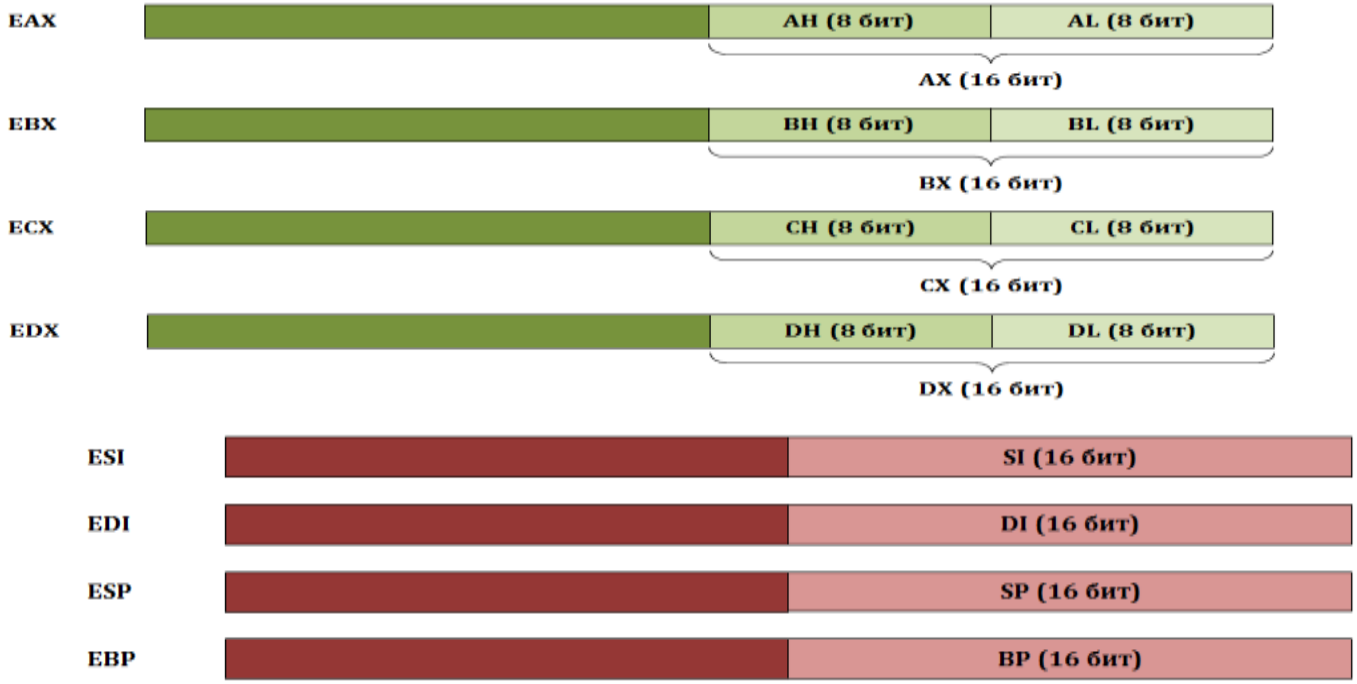
Знакомство с ассемблером, изучение возможностей использования регистров микропроцессора при программировании

ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ.

Регистры – это ячейки памяти, расположенные непосредственно в процессоре. Работа с регистрами выполняется намного быстрее, чем с ячейками оперативной памяти, поэтому регистры активно используются как в программах на языке ассемблера, так и компиляторами языков высокого уровня.

Регистры можно разделить на *регистры общего назначения*, *указатель команд*, *регистр флагов и сегментные регистры*.

К регистрам общего назначения относится группа из 8 регистров, которые можно использовать в программе на языке ассемблера. Все регистры имеют размер 32 бита и могут быть разделены на 2 или более частей.



Названия регистров происходят от их назначения:

- **EAX/AX/AH/AL** (*accumulator register*) – аккумулятор;
- **EBX/BX/BH/BL** (*base register*) –регистрбазы;
- **ECX/CX/CH/CL** (*counter register*) – счётчик;
- **EDX/DX/DH/DL** (*data register*) – регистрданных;
- **ESI/SI** (*source index register*) – индексисточника;
- **EDI/DI** (*destination index register*) – индексприёмника (получателя);
- **ESP/SP** (*stack pointer register*) – региструказателястека;
- **EBP/BP** (*base pointer register*) – региструказателябазыкадрастека.

Несмотря на существующую специализацию, все регистры можно использовать в любых машинных операциях. Однако надо учитывать тот факт, что некоторые команды работают только с определёнными регистрами.

Ещё можно использовать регистры в качестве *базы*, т.е. хранилища адреса оперативной памяти. В качестве регистров базы можно использовать любые регистры, но желательно использовать регистры EBX, ESI, EDI или EBP. В этом случае размер машинной команды обычно бывает меньше.

Регистр EIP (*указатель команд*) содержит смещение следующей подлежащей выполнению команды. Этот регистр непосредственно недоступен программисту.

Флаг – это бит, принимающий значение 1 («флаг установлен»), если выполнено некоторое условие, и значение 0 («флаг сброшен») в противном случае. Процессор имеет *регистр флагов*, содержащий набор флагов, отражающий текущее состояние процессора.

- ZF - признак нуля результата (ZF = 1, если все разряды результата равны 0);
- SF - знак результата (SF = 1, если результат отрицательный);
- OF - признак переполнения (OF = 1, если при выполнении арифметических операций над числами со знаком происходит переполнение разрядной сетки);
- CF - флаг переноса (CF = 1, если выполнение операции сложения приводит к переносу за пределы разрядной сетки), устанавливается также в некоторых других операциях;

Некоторые команды ассемблера:
MOV dst, src – пересылка данных из src в dst
ADD dst, src – содержимое src складывается с содержимым dst, и результат помещается в src.
SUB dst, src – содержимое src вычитается из содержимого dst, и результат помещается в src.
INC dst – прибавляет 1 к содержимому dst.
DEC dst – вычитает 1 из содержимого dst.

ХОД РАБОТЫ.

Задание.

- В регистр AX запишите свой год рождения.
- В регистр BL запишите число = (порядковый номер в журнале+10)
- В регистр DL запишите число $(45)_{10}$
- Сложите содержимое регистров AX и BL
- Результат вычислений сохраните в регистре CX
- Из полученной суммы вычесть содержимое регистра DL
- Из регистра CX вычесть содержимое регистра AX

СОДЕРЖАНИЕ ОТЧЕТА

Отчет должен содержать:

- номер варианта.
- программу по каждому заданию;
- состояние задействованных в вычислениях регистров после выполнения каждой команды (каждого шага программы);
- параллельное вычисление команд вручную.

Подробные переводы чисел можно не писать, просто указать, например, $5_{10}=101_2$

ПРАКТИЧЕСКАЯ РАБОТА №2

ТЕМА: Ветвления на выбранном языке программирования

ЦЕЛЬ РАБОТЫ

Получение практических навыков по программированию ветвлений на языке ассемблера.

ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ.

В языках программирования высокого уровня конструкция выбора известна как оператор IF - THEN. Эта конструкция позволяет выбрать следующее действие из нескольких возможных вариантов в зависимости от выполнения определенного условия. В языке ассемблера механизм выбора реализован посредством команд сравнения, условного и безусловного переходов.

Команда CMP используется для сравнения двух операндов: CMP o1, o2

Эта команда работает подобно команде вычитания: o2 вычитается из o1. Результат нигде не сохраняется, команда меняет регистр признаков.

Примеры:

CMP ax, 4 - сравниваем содержимое регистра с числом

CMP dl, ah - сравниваем содержимое двух регистров

В языке ассемблер используется несколько команд условного перехода. Чтобы запомнить названия команд, надо запомнить несколько английских слов: jump - прыжок ,equal - равно, above - больше, below - ниже, zero- ноль, greater - больше, less - меньше, not - нет.Например, прыжок, если не ноль - JNZ.

Результат сравнения	**o1=o2**	**o1 не равно o2**	**o1>o2**	**o1<o2**	**o1<=o2**	**o1>=o2**
для положительных чисел	JE(JZ)	JNE (JNZ)	JA (JNBE)	JB(JNAE)	JNA (JBE)	JNB (JAE)
	Переход, если равно (переход, если 0)	Переход, если не равно (переход, если не 0)	Переход, если больше (переход, если не меньше или равно)	Переход, если меньше (переход, если не больше или равно)	Переход, если не больше (переход, если меньше или равно)	Переход, если не меньше (переход, если больше или равно)
Для отрицательных чисел	JE(JZ)	JNE (JNZ)	JG (JNLE)	JL(JNGE)	JNG (JLE)	JNL (JGE)

ХОД РАБОТЫ.

Задание:

в регистр запишите число

разделите его на 3_{10}

если число разделилось без остатка, то снова разделите результат на 3_{10}

и так до тех пор, пока не появится остаток

Запишите программу с адресами строк и результаты пошагового выполнения. Обязательно напишите какую команду процессор выполняет на каждом шаге.

СОДЕРЖАНИЕ ОТЧЕТА

Отчет должен содержать:

- номер варианта.
- программу по каждому заданию;

- состояние задействованных в вычислениях регистров после выполнения каждой команды (каждого шага программы);
- параллельное вычисление команд вручную.

Подробные переводы чисел можно не писать, просто указать, например, $5_{10}=101_2$

ПРАКТИЧЕСКАЯ РАБОТА №3

ТЕМА Организация циклов на языке программирования

ЦЕЛЬ РАБОТЫ изучить особенности программирования на языке ассемблера и особенности выполнения группы команд пересылок, приобрести навыки отладки программ

ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ.

При выполнении программы команды МП выполняются последовательно одна за другой. Однако во многих случаях необходимо изменять порядок следования команд, например при выполнении циклических вычислений, анализе различных признаков, влияющих на процесс выполнения программы. Такой переход может быть осуществлен безусловно или только при выполнении определенных условий. В качестве условий выступают значения признаков в регистре признаков МП. Анализируя значения признаков, команды не изменяют их.

Мнемокод	Операнд	Комментарий
JMP	метка	Производится безусловный переход на метку
JC	метка	Производится условный переход на метку, если C=1.
JNC	метка	Производится условный переход на метку, если C=0.
JZ	метка	Производится условный переход на метку, если Z=1.
JNZ	метка	Производится условный переход на метку, если Z=0.
JS	метка	Производится условный переход на метку, если S=1.
JNS	метка	Производится условный переход на метку, если S=0
JP	метка	Производится условный переход на метку, если P=1,
JNP	метка	Производится условный переход на метку, если P=0
JO	метка	Производится условный переход на метку, если O=1
JNO	метка	Производится условный переход на метку, если O=0
JCXZ	метка	Производится условный переход на метку, если CX=0
LOOP	метка	Производится •декремент регистра cx; •сравнения регистра cx с нулем: •если (cx) > 0, то управление передается на метку •если (cx) = 0, то управление передается на следующую после loop команду.
LOOPE (LOOPZ)	метка	Производится •декремент регистра cx; •сравнения регистра cx с нулем; •анализа состояния флага нуля zf: •если (cx) > 0 и zf = 1, управление передается на метку ; •если (cx) = 0 или zf = 0, управление передается на следующую после loop команду.
LOOPNE (LOOPNZ)	метка	Производится •декремента регистра cx; •сравнения регистра cx с нулем; •анализа состояния флага нуля zf: •если (cx) > 0 и zf = 0, управление передается на метку; •если (cx)=0 или zf=1, управление передается на следующую после loop команду.

Команды loope/loopz и loopne/loopnz по принципу своей работы являются взаимообратными. Они расширяют действие команды loop тем, что дополнительно

анализируют флаг zf, что дает возможность организовать досрочный выход из цикла, используя этот флаг в качестве индикатора.

ХОД РАБОТЫ

Составить алгоритм и программу заполнения последовательностью констант, начиная с 1H, области памяти длиной FH байт с начальным адресом ds:0000H, если число в аккумуляторе - нечетное. В противном случае эту область заполнить константой F5H.

Вариант 2

Занести в область памяти ds:0008H - ds:0027H последовательность констант 0H - 19H и найти среди них третье по счету четное число, записать это число в регистр CL.

Вариант 3

Составить алгоритм и программу заполнения области памяти ds:0008H - ds:0057H последовательность чисел от 0H до 49H, если число в аккумуляторе не отрицательное. Вывести в ячейку памяти ds:0058H число, находящееся в аккумуляторе.

Вариант 4

Составить алгоритм и программу заполнения области памяти ds:0018H - ds:001FH константой F3H если число в аккумуляторе - четное, в противном случае константой 11H. Вывести в ячейку памяти ds:0020H число, находящееся в аккумуляторе.

Вариант 5

Составить алгоритм и программу заполнения области памяти ds:0020H - ds:0027H константой E7H если число в аккумуляторе отрицательное, в противном случае константой CEH. Вывести в ячейку памяти ds:0028H число, находящееся в аккумуляторе.

Вариант 6

Составить алгоритм и программу заполнения области памяти длиной 17H байт с начальным адресом ds:0008H константой D1H если в аккумуляторе ноль, в противном случае в эту область памяти внести последовательность чисел начиная с 1H.

Вариант 7

Занести в область памяти ds:0000H - ds:0017H последовательность констант от 70H до 87H и найти среди них второе по счету отрицательное число, записать это число в регистр BL.

Вариант 8

Составить алгоритм и программу заполнения области памяти длиной 25H байт с начальным адресом ds:0000H константу E5H если значение аккумулятора нулевое, в противном случае в эту область памяти внести последовательность чисел начиная с 1H. Вывести в ячейку памяти ds:0026H число, находящееся в аккумуляторе.

Вариант 9

Составить алгоритм и программу заполнения области памяти длиной 20H байт с начальным адресом ds:0000H последовательность констант от 0H до 20H а в область памяти ds:0020H - ds:0040H последовательность констант от 20H до 0H.

Вариант 10

Составить алгоритм и программу занесения в область памяти ds:0008H - ds:0027H последовательность констант от 70H до 8FH и найти среди них четвертое по счету отрицательное число, записать это число в ячейку памяти ds:0018H.

Вариант 11

Составить алгоритм и программу заполнения последовательностью констант 16H, 15H, 14, … области памяти длиной 8H байт с начальным адресом ds:0008H, если число в аккумуляторе - четное. В противном случае эту область заполнить константой EEH.

Вариант 12

Занести в область памяти ds:0018H - ds:0027H последовательность констант 20H - 11H и найти среди них второе по счету четное число, записать это число в регистр BH.

Вариант 13

Составить алгоритм и программу заполнения области памяти ds:0008H – ds:000FH последовательностью чисел от 0H до 7H, если число в аккумуляторе отрицательное в

противном случае в эту область заполнить последовательностью чисел от 7H до 0H. Вывести в ячейку памяти ds:0058H число, находящееся в аккумуляторе.

Вариант 14

Составить алгоритм и программу заполнения области памяти ds:0018H - ds:001FH константой F3H если число в аккумуляторе - четное, в противном случае константой 11H. Вывести в ячейку памяти ds:0020H число, находящееся в аккумуляторе.

Вариант 15

Составить алгоритм и программу заполнения области памяти ds:0020H – ds:003FH константой BBH если число в аккумуляторе не отрицательное, в противном случае константой ABH. Вывести в ячейку памяти ds:0028H число, находящееся в аккумуляторе.

СОДЕРЖАНИЕ ОТЧЕТА

Отчет должен содержать:

- программу по каждому заданию;
- состояние задействованных в вычислениях регистров после выполнения каждой команды (каждого шага программы);

ПРАКТИЧЕСКАЯ РАБОТА №4

ТЕМА Передача данных

ЦЕЛЬ РАБОТЫ Изучить особенности выполнения арифметических команд, разработка алгоритма, составление и отладка программы с использованием этих команд

.

ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ.

Задача №1:

Ввод строки с клавиатуры осуществляется с помощью 10-ой функции прерывания 21h операционной системы MS DOS.

В регистре DX задаётся адрес буфера вводимой строки. Ограничитель количества вводимых с клавиатуры символов строки вносится в первую ячейку буфера.

Количество вводимых с клавиатуры символов на 1 меньше значения ограничителя.

В регистре AH задаётся номер функции прерывания 21h: 10-ая функция.

Ниже представлен листинг программы (ассемблерного модуля), предназначенной для ввода строки с клавиатуры.

```
model small
gr GROUP c_sgm, d_sgm
ASSUME cs: gr, ds: gr

d_sgm SEGMENT
      space  DB 12 DUP(0)
d_sgm ENDS

c_sgm SEGMENT
strt:
      mov  ax,   gr
      mov  ds,   ax
      mov  bp,   offset space
      mov  al,   8
      mov  [bp], al
      mov  dx,   bp
      mov  ah,   10
      int  21h
      mov  ah,   4ch
      int  21h
c_sgm ENDS
END strt
```

Задача №2:

Запросить имя. Ввести это имя. Вывести именное приветствие.

Ниже представлен листинг программы (ассемблерного модуля), предназначенного для решения поставленной задачи.

```
model small
gr GROUP c_sgm, d_sgm
ASSUME cs: gr, ds: gr

d_sgm SEGMENT
      y_name DB "What's your name?",13,10,"$"
      space  DB 8, 12 DUP(0)
      hello  DB 13,10,"Hello, ",10 DUP(32)
d_sgm ENDS
```

```
c_sgm SEGMENT
strt:
      mov  ax,  gr
      mov  ds,  ax
      mov  dx,  offset y_name
      mov  ah,  9
      int  21h
      mov  dx,  offset space
      mov  ah,  10
      int  21h
      mov  si,  offset space
      mov  cl,  [si +01]
      mov  ch,  0
      mov  di,  offset hello
mlp:   mov  ah,  [si+02]
      mov  [di+09], ah
      inc  si
      inc  di
      loop mlp
      mov  ax,2421h
      mov  [di+09], ax
      mov  dx,  offset hello
      mov  ah, 09
      int  21h
      mov  ah,  4ch
      int  21h
c_sgm ENDS

END strt
```

Комментарий к представленному выше коду:
y_name – буфер вывода приветствия и последующего перевода курсора на следующую строку
space – буфер ввода строки с клавиатуры (с ограничителем = 8 в начальной ячейке буфера)
hello – буфер именного приветствия (заранее заготовлен шаблон приветствия: «Hello, ...»).

Имя копируется в этот буфер после ввода с клавиатуры. С учетом количества символов слова «Hello, ...» при копировании выбрано смещение = 09.

Сначала выводится приветствие “What's your name?”
с автоматическим переводом курсора
на следующую строку (,13,10,”$”)

Далее выполняется ввод строки в буфер space.
В этот буфер заранее введено значение ограничителя = 8
(максимальное количество символов вводимой строки = 7).

Далее из ячейки ([si +01]) в регистр cl
копируется количество символов введённой строки.

После этого в цикле выполняется копирование символов
введённой строки из буфера space в буфер hello
со смещением = 09 (чтобы не затереть слово «Hello, ...»).

По завершении копирования в ячейки,
расположенные непосредственно за скопированной строкой,
добавляются символы 2421h («!$»),
завершающие формирование строки именного приветствия.

После проведённой работы
остаётся вывести именное приветствие.
Стоит обратить внимание на то,

что именное приветствие начинается кодами (13,10,),
которые обеспечивают перевод курсора на следующую строку.

ХОД РАБОТЫ.

Задания:

1. Изучить содержание теоретической части.
2. Повторить выполнение рассмотренных примеров .
3. Составить ассемблерные модули для решения следующих задач:
4. Ввести строку символов и вывести на экран первые 5 символов введённой строки;
5. Ввести строку символов и вывести на экран последние 5 символов введённой строки;
6. Ввести строку символов и вывести на экран все символы строки, кроме 3-х первых символов и 3-х последних.

СОДЕРЖАНИЕ ОТЧЕТА

Отчет должен содержать:

- программу по каждому заданию;
- состояние задействованных в вычислениях регистров после выполнения каждой команды (каждого шага программы);

ПРАКТИЧЕСКАЯ РАБОТА №5

ТЕМА Изучение приемов работы со стеком

- **ЦЕЛЬ РАБОТЫ**

Получение практических навыков по идентификации и установке процессора.

- **ОБЕСПЕЧЕННОСТЬ ЗАНЯТИЯ**

Реализация учебной дисциплины требует наличия кабинета-лаборатории архитектуры ЭВМ и вычислительных систем и сборки, монтажа и эксплуатации средств ВТ.

Оборудование:

- ✓ посадочные места по количеству обучающихся;
- ✓ рабочее место преподавателя;
- ✓ плакаты;

Технические средства обучения:

- ✓ компьютер с лицензионным программным обеспечением: ОС Windows, пакет MS Office, AVR Studio
- ✓ мультимедиапроектор;

- **ЛИТЕРАТУРА** Таненбаум Э. Архитектура компьютера/Э.Таненбаум.-Питер, 2012.-699 с

- **ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ.**

Фирма Atmel – разработчик микроконтроллеров AVR, позаботилась о сопровождении своей продукции. Для написания программ, их отладки, трансляции и прошивки в память микроконтроллера фирма разработала и бесплатно распространяет специализированный программный инструмент разработчика под названием «AVR Studio». Установочный пакет этой инструментальной программы можно скачать с сайта фирмы http://www.atmel.com. Программная среда «AVR Studio» — это современный программный продукт, позволяющий производить все этапы разработки программ для многих микроконтроллеров серии AVR. Пакет включает в себя специализированный текстовый редактор для написания программ, программный отладчик. Кроме того, «AVR Studio» позволяет управлять целым рядом подключаемых к компьютеру внешних устройств, позволяющих выполнять аппаратную отладку, а также программирование («прошивку») микросхем AVR. Программная среда «AVR Studio» работает не просто с программами, а с проектами. Проект в терминах AVR Studio – это совокупность файлов исходных текстов программ(ы) и служебных файлов AVR Studio. Исходные тексты разработчик создает сам, реализуя логику работы микроконтроллера в рамках поставленой задачи. Служебные файлы проекта создаются самим AVR Studio для хранения в них информации о проектных файлах и о настройках среды. Это текстовые файлы с расширениями aps и aws. Для каждого проекта должен быть отведен свой отдельный каталог на жестком диске. В AVR Studio одновременно может быть загружен только один проект. При загрузке нового проекта предыдущий проект автоматически выгружается. Головным файлом проекта является файл с расширением aps. Он содержит сведения о типе процессора, используемого в проекте, частоте тактового генератора и т. д. Он также содержит описание всех остальных файлов, входящих в проект. Все эти сведения используются при отладке и трансляции программы. Кроме файла aps, проект должен содержать хотя бы один файл с текстом программы. Такой файл имеет расширение asm. Недостаточно просто поместить asm-файл в директорию проекта. Его нужно еще включить в проект. Как это делается, будет показано позже. Проект может содержать несколько asm-файлов. При этом один из них является главным (он указывается при создании проекта). Остальные могут подключаться главным файлом при помощи директивы препроцессора .include. На этом заканчивается список файлов проекта, которые создаются при участии разработчика. Если программа не содержит синтаксических ошибок и процесс трансляции прошел успешно, то в директории проекта автоматически появляются следующие файлы: - файл, содержащий результирующий код в hex формате; - файл, содержащий все символьные имена транслируемой программы со своими значениями; - листинг-трансляции (lst) и другие вспомогательные файлы. Однако будет важен выходной

hex-файл (файл с расширением hex). Именно он будет служить источником данных при прошивке программы в программную память микроконтроллера.

ВНЕШНИЙ ВИД ПРОГРАММЫ «AVR STUDIO»

На рисунке 1 показано, как внешне выглядит «AVR Studio». На самом деле «AVR Studio» имеет очень гибкий интерфейс, и внешний вид (рисунок 1) может сильно отличаться от варианта, показанного на рисунке. Но далее будет рассмотрен случай, когда выбраны установки по умолчанию.

Рисунок 1 – Внешний вид AVR Studio

Главная панель программы AVR Studio разделена на три основных окна. На рисунке 1 они обозначены цифрами 1, 2, 3 и 6.Окно 6 предоставляет информацию о текущем проекте.

Окно 6 «Project» содержит информацию по текущему загруженному проекту. Информация представлена в виде дерева. Разные ветви этого дерева описывают все исходные и результирующие файлы проекта, все метки, процедуры и присоединяемые файлы. Окно 1 отображает архитектурный состав компонентов выбранного для проекра микроконтроллера: регистры управления и состояния устройств (таймеры, порты, АЦП и т.д.), их прерывания и значения их регистрой данных. В этом окне во время отладки программы в режиме симулятора можно будет наблюдать за изменением состояний устройств, прерываний и регистров с которыми работает отлаживаемая программа. В пошаговом пежиме можно будет также «на лету» подменять значения нужных регистров и отдельно взятых управляющих битов с целью более оптимальной проверки работы алгоритмов программы. Элементы, появляющиеся в результате раскрытия ветви, в свою очередь также могут быть раскрыты, если они имеют свое содержимое. На рисунке 2 в увеличенном виде показано дерево ресурсов микроконтроллера ATiny2313. На рисунке несколько ветвей специально раскрыты, чтобы можно было увидеть их состав. Если какая-либо ветвь может быть раскрыта, то в своем основании она имеет квадратик с плюсиком внутри. Двойной щелчок на этом плюсике раскрывает ветвь. В раскрытой ветви плюсик превращается в минус. Повторный двойной щелчок по квадратику закрывает раскрытую ветвь. На рисунке 2 для наглядности раскрыты ветви портов ввода/вывода и можно видеть: - полный состав управляющих регистров для каждого из устройств; - их названия и адреса; - состав и название каждого бита (если биты имеют свои названия).

Рисунок 2 – Окно ресурсов микроконтроллера

В процессе отладки в этом окне можно увидеть не только название и состав всех ресурсов, но и их содержимое. Содержимое будет отображаться как в шестнадцатиричном виде, так и путем затемнения квадратиков, отображающих отдельные биты конкретных регистров. Затемненный квадратик означает, что бит равен единице. Светлый квадратик говорит о том, что бит равен нулю. Вы также можете оперативно менять это содержимое прямо в этом окне. Для изменения значения бита достаточно двойного щелчка мышки в соответствующем квадратике. Существуют и другие способы изменения содержимого различных регистров и ячеек памяти в процессе отладки. В нижней части главной панели находится второе вспомогательное окно (окно 2 на рисунке 1). Это окно служит, в основном, для вывода различных сообщений. Оно также содержит ряд вкладок. По умолчанию их четыре. Первая вкладка называется «Build». На вкладке «Build» отражается процесс трансляции. На эту вкладку выводятся сообщения об различных этапах трансляции, сообщения о синтаксических ошибках и различные предупреждения (Warnings). Если трансляция заканчивается нормально (отсутствуют ошибки компиляции), то сюда же выводятся статистические данные о полученном результирующем коде. Эти данные показывают размеры и процент использования всех видов памяти микроконтроллера:

Сообщение означает, что в программном сегменте исполняемый код занимает 72 ячейки. Константы в памяти программ занимают 257 16-битных слов. Размер программной памяти для этого микроконтроллера составляет 513 слов (16-битных ячеек). Последняя строка содержит сообщения об ошибках. В данном случае сообщение переводится так: «Ассемблирование прошло без ошибок». Следующая вкладка второго окна называется «Message». Здесь выводятся разные системные сообщения о загрузке модулей программы и т. п. Третья вкладка второго окна называется «Find in Files» (поиск в файлах). В этом окне отражаются результаты выполнения команды «Поиск в Файлах». Эта команда позволяет производить поиск заданной последовательности символов сразу во всех файлах проекта. По окончании поиска во вкладке «Find in Files» отражаются все найденные вхождения с указанием имени файла и строки, где найдена искомая последовательность. Последняя вкладка называется «Breakpoints and Tracepoints» (Точки останова и точки трассировки). Эти точки проставляются в тексте программы перед началом процесса отладки и дублируются в данном окне. Как проставлять точки останова, беднт описано позже.

Точки останова (breakpoints) используются для того, чтобы приостановить выполнение программы в том или ином месте программы для того, чтобы убедиться, что программа выполняется правильно. Точки останова ставятся/убираются в окне текста программы нажатием <F9> на строке предполагаемого останова программы.

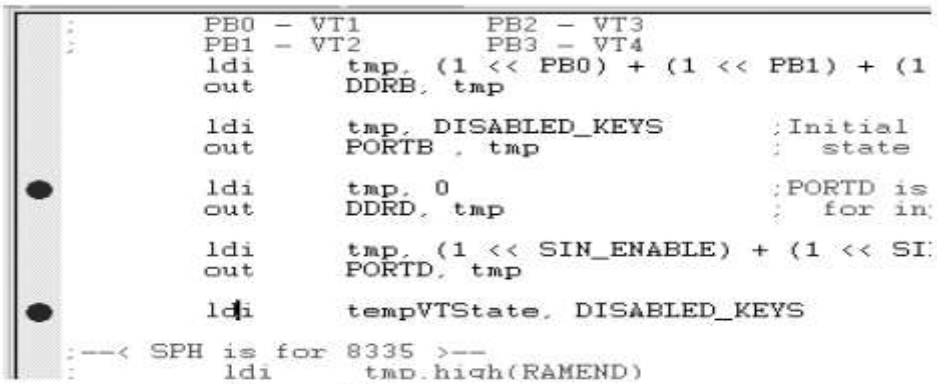


Рисунок 3 – точки останова в тексте

При установке точки останова в тексте программы строка помечается маркером в форме точки большого диаметра, а информация о ней автоматически появляется во вкладке «Breakpoints and Tracepoints» (рисунок 4):

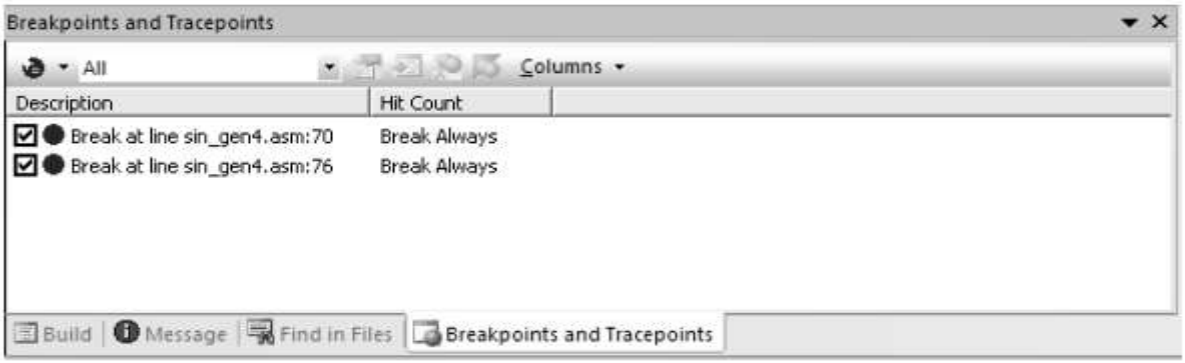


Рисунок 4 – Break points and tracepoints

Вкладка позволяет увидеть все точки останова программы в одном списке. Кроме того, на вкладке против каждой записи, описывающей точку останова, автоматически появляется «Check box» (поле выбора), при помощи которого можно в любой момент временно отключить любую точку останова. В окне 3 можно открывать не только все тексты ассемблерных программ текущего проекта, но и тексты программ других проектов, а также тексты программ, написанных на других языках программирования. Такой прием очень удобен, если нужно переделать программу, написанную для старого микроконтроллера на старой версии Ассемблера на новый лад. Все открытые текстовые окна запоминаются и затем открываются автоматически при открытии проекта.

Любое текстовое окно имеет подсветку синтаксиса. Разные части помещенного туда текста программы подсвечиваются разными цветами. Так, все операторы Ассемблера высвечиваются голубым цветом. Комментарии выделяются зеленым. Остальной текст (параметры команд, псевдооператоры, метки, переменные и константы) остается черным. Это очень удобно. Если написанный вами оператор окрасился в голубой цвет, то это значит, что вы не ошиблись в синтаксисе. Если вы написали комментарий, но перед текстом комментария забыли поставить точку с запятой, то этот комментарий не окрасится в зеленый цвет. Таким образом, многие ошибки видны уже в процессе написания программы. Кроме двух вспомогательных и одного основного окна, главная панель программы имеет строку меню (отмечена цифрой 4 на рисунке 1), а также несколько инструментальных панелей (отмечены цифрой 5). Как и в любой другой программе под Windows, при помощи меню вызываются все функции программы AVR Studio и переключаются все ее режимы. Панели инструментов дублируют часто используемые функции меню.

СОЗДАНИЕ ПРОЕКТА НА АССЕМБЛЕРЕ После запуска программы открывается мастер нового проекта, который также можно вызвать через меню ―Project –> New project‖ (Рисунок 5).

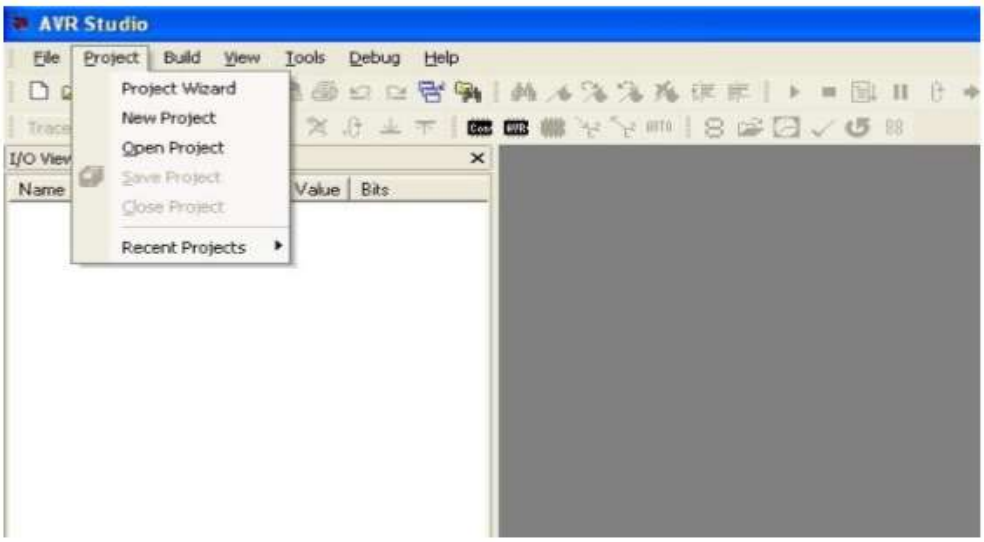


Рисунок 5 – Создание проекта

В открывшемся окне надо выбрать тип проекта - Atmel AVR Assembler (рисунок 4):

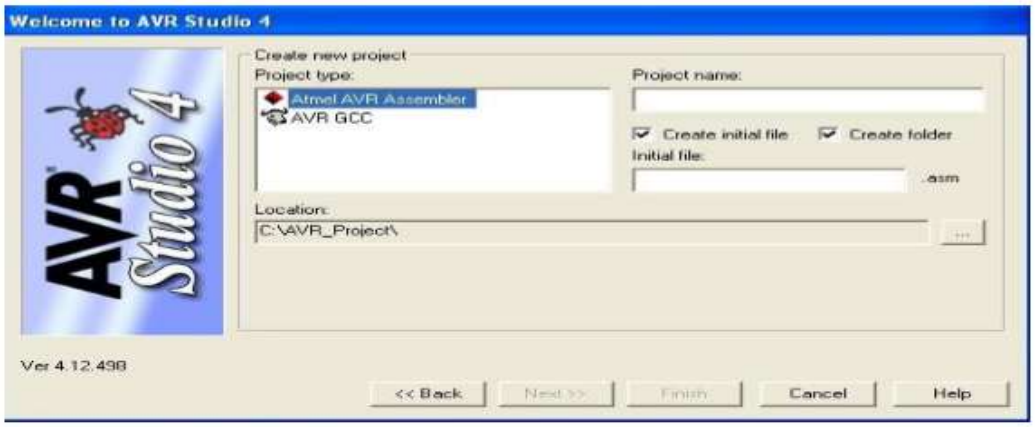


Рисунок 6 – Выбор тира проекта

Первый чек-бокс (Create initialize file) определяет, нужно ли автоматически создавать главный программный файл. Если поставить галочку «Create initial file» - будет создан пустой файл для исходного текста или откроется уже существующий в случае, если он уже существует по указанному пути. Введите имя начального файла Второй чек-бокс (Create folder) определяет, нужно ли автоматически создавать отдельный каталог для данного проекта. Если вы заранее уже создали нужный каталог средствами Windows, снимите пометку. Если нет, оставьте. Следующее поле называется «Initial file». Оно должно содержать имя файла, куда будет записываться текст программы. По умолчанию имя файла вписывается в это поле и, оно идентично введенному имени проекта, к которому добавляется расширение asm. Еще одно поле, требующее нашего вмешательства, — это поле «Location». Здесь вы должны указать путь к тому месту на вашем жестком диске, где будет храниться проект . Путь нельзя ввести непосредственно с клавиатуры. Для изменения пути нужно нажать кнопку справа, на которой в качестве названия поставлено многоточие («...»).

. Назначте рабочую папку - место размещения проекта. Затем перейти далее (нажать ктопку «Next>>») на форму где необходимо определится с типом используемой отладочной платы.

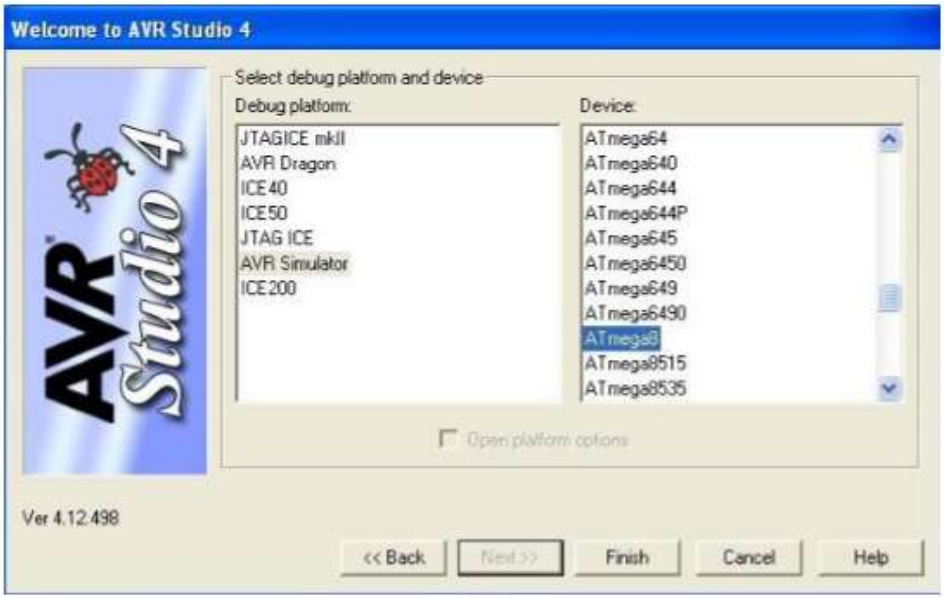


Рисунок 7 – Выбор модели контроллера

Если предполагается воспользоватся встроенным симулятором следует выбрать пункт AVR Simulator и выбрать используемый тип микроконтроллера. После произведенных действий откроется окно програмного кода.

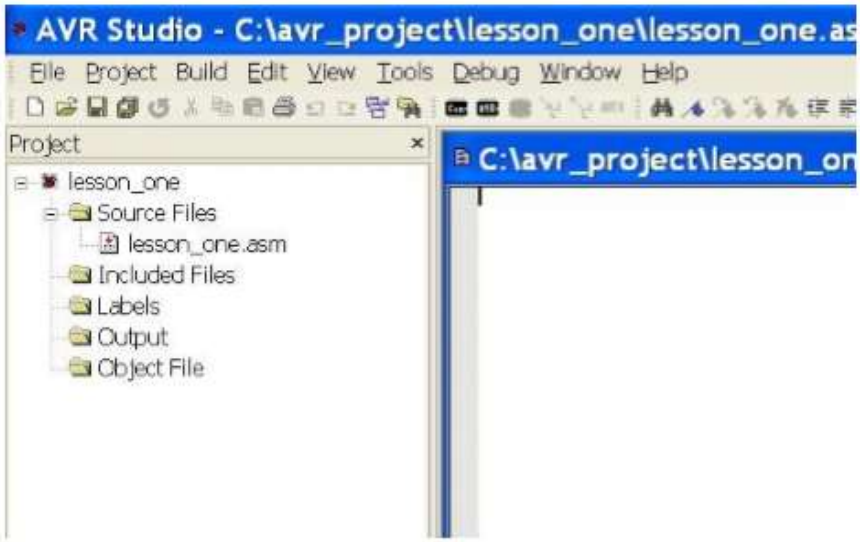


Рисунок 8 – Вид после создания проекта

В этом окне набирается и корректируется текст программы. Помимо окна с кодом программы, в AVR Studio существует еще несколько полезных окон: Project, I/O View и Info, служащих для отображения состава проекта, состояния регистров и портов микроконтроллера с возможностью управления состоянием регистров и портов и дополнительной справочной информации о микроконтроллере, но об этом чуть позднее...

Сразу после создания новый проект состоит всего из двух файлов: - собственно файл проекта Prog1. aps; - файл, куда будет помещен текст программы на Ассемблере Prog1.asm.

Файл текста программы автоматически открывается в окне 3. Причем он пока абсолютно пустой. Теперь вы можете приступать к набору этого текста. Если речь идет о программе Prog1, то просто наберите текст, приведенный в листинге 4.1. При наборе текста вы можете пользоваться всеми возможностями, какие обычно предоставляет любой современный текстовый редактор. Встроенный текстовый редактор программы AVR Studio поддерживает все необходимые сервисные функции: - выделение текстовых фрагментов; - вырезание; - копирование; - вставку; - перетаскивание мышью; - поиск и замену и многое другое. Для управления всеми этими возможностями используется стандартный интерфейс, знакомый вам по многим текстовым редакторам, в частности, по популярному редактору

Microsoft Word. Набранный тест программы не забудьте записать на диск при помощи команды «Save» меню «File» или при помощи соответствующей кнопки на панели инструментов. Кнопка позволяет записать сразу все открытые текстовые файлы.

ТРАНСЛЯЦИЯ ПРОГРАММЫ

После того, как текст программы набран и записан на жесткий диск, необходимо произвести трансляцию программы. В процессе трансляции создается результирующий файл, который представляет собой ту же программу, но в машинных кодах, предназначенную для записи в программную память микроконтроллера. Результирующий файл имеет расширение hex. Кроме hex-файла транслятор создает еще несколько вспомогательных файлов. И главное, файл с расширением eep. Этот файл имеет точно такую же внутреннюю структуру, как файл hex. А содержит он информацию, предназначенную для записи в EEPROM. Такая информация появляется в том случае, когда в тексте программы переменным, размещенным в сегменте eeprom, присвоены начальные значение. В наших примерах мы этого не делали. Поэтому файл с расширением eep во всех проектах будет пустой (содержать лишь завершающую строку). Формат файлов hex и eep. В обоих случаях применяется так называемый HEXформат, который практически является стандартом для записи результатов транслирования различных программ. Он поддерживается практически всеми трансляторами с любого языка программирования. В принципе, разработчику не обязательно знать структуру этого формата. Достаточно понимать, что в hex-файле определенным способом закодирована программа в машинных кодах. Именно этот файл используется программатором для «прошивки» программной памяти микроконтроллера. Любой программатор поддерживает hex-формат и распознает записанные туда коды автоматически. Процедура трансляции Для того, чтобы запустить процесс трансляции текущего проекта, нужно выбрать в меню «Build» пункт, который тоже называется «Build», или нажать кнопку <F7>. Длительность процесса трансляции зависит от размеров программы- Сразу же после начала процесса вкладка «Build» в окне 2 выходит на передний план. В процессе трансляции сюда выводятся служебные сообщения. К таким сообщениям относятся: сообщения о завершении различных этапов трансляции, сообщения об ошибках (Error), а также предупреждения (Warning). В готовой отлаженной программе ошибок и предупреждений быть не должно. Если программа обнаружит ошибку (Error), то процесс трансляции будет остановлен, и результирующие файлы созданы не будут. В этом случае необходимо устранить ошибки и повторить трансляцию. Естественно, транслятор не в состоянии найти все виды ошибок. Он находит только явные ошибки, которые можно найти автоматически. К таким ошибкам относятся: - ошибки синтаксиса (неправильное написание имени команды); - неверное количество параметров у оператора; - попытка использования неописанных переменных и т. п. Например, сообщение «Unknown instruction or macro» означает, что найдена «Неизвестная инструкция или макрокоманда». Предупреждения (Warnings) — это сообщения о найденых местах в программе, в которых может быть скрыта ошибка, связанная например с использованием разных имен для одного адреса, переопределением констант новыми значениями и т.п. Транслятор не считает подобные факта ошибками, т.к. с точки зрения синтаксиса все нормально. Однако он обращает на эти места внимание, т.к. иногда бывает, что из-за пересечения или переопределения имен возникают логические ошибки в ходе выполнения самой программы. Это не означает, что в подобных местах обязательно может быть ошибка. Главное – обратить на сообщения внимание и удостовериться, что все сделано правильно или исправить ошибку, если она действительно закралась в указанном месте. Все сообщения во вкладке «Build» появляются по мере их поступления. Для наглядности каждое сообщение помечено цветным кружочком в начале строки: - сообщения об ошибках помечаются кружочком красного цвета; - предупреждения – желтым кружочком; - сообщения об успешном выполнении каждого очередного этапа трансляции помечаются зеленым кружочком. В случае успешного завершения процесса трансляции в качестве последнего сообщения выводится статистическая информация. Каждое сообщение об ошибке во вкладке «Build» содержит точное указание места в программе, где произошла эта ошибка. При этом указывается - имя файла; - номер строки;

- фрагмент текста программы, содержащий ошибку; - ее расшифровка. Для того, чтобы быстро перейти к фрагменту программы, содержащему эту ошибку, достаточно двойного щелчка по сообщению об ошибке. Окно с текстом программы выйдет на передний план, и в этом окне автоматически отобразится нужный участок текста. На левой границе окна напротив строки, содержащей ошибку, вы увидите синюю стрелочку — указатель ошибки. Иногда программа неверно определяет место, где возникла ошибка. Это происходит из-за несовершенства анализатора синтаксиса. Дело в том, что очень сложно разработать идеальный алгоритм анализа ошибок. Если в какой-либо строке транслятор показывает ошибку, а вы ошибок не наблюдаете, посмотрите на предыдущие строки. Возможно, ошибка где-то там. Если исправлены все ошибки, то это значит, что можно записывать ее в программную память микроконтроллера и пробовать ее работу «в железе». Однако в большинстве случаев отсутствие синтаксических ошибок еще не означает отсутствие логических ошибок, допущенных на стадии разработки алгиритма или ввода исходного текста программы. Можно написать команду правильно, да не ту. Но самая главная неприятность — ошибки алгоритма или его реализация-Программист может упустить какой-либо шаг или неправильно поставить условие. Всех возможных ошибок алгоритма не перечесть. Но в результате программа может работать неправильно либо совсем не работать. По этой причине перед тем, как записывать программу в программную память микроконтроллера, необходимо попытаться выявить все эти ошибки. Вообще, процесс написания программы процентов на 60—70 состоит из поиска и устранения ошибок. И основное количество ошибок выявляется при отладке программы.

ОТЛАДКА ПРОГРАММЫ НА АССЕМБЛЕРЕ ВЫПОЛНЕНИЕ ПРОГРАММ В ПОШАГОВОМ РЕЖИМЕ ПРОСМОТР РЕГИСТРОВ

С юмором подошли к этому вопросу англичане. По-английски процесс отладки называется Debug (Дебаг). Слово «Bug» — означает блоха, а «Debug» — это процесс избавления от ошибок или процесс ловли блох. Именно этим вам и придется заняться. Если после успешного асссемблирования запустить программу на выполнение,

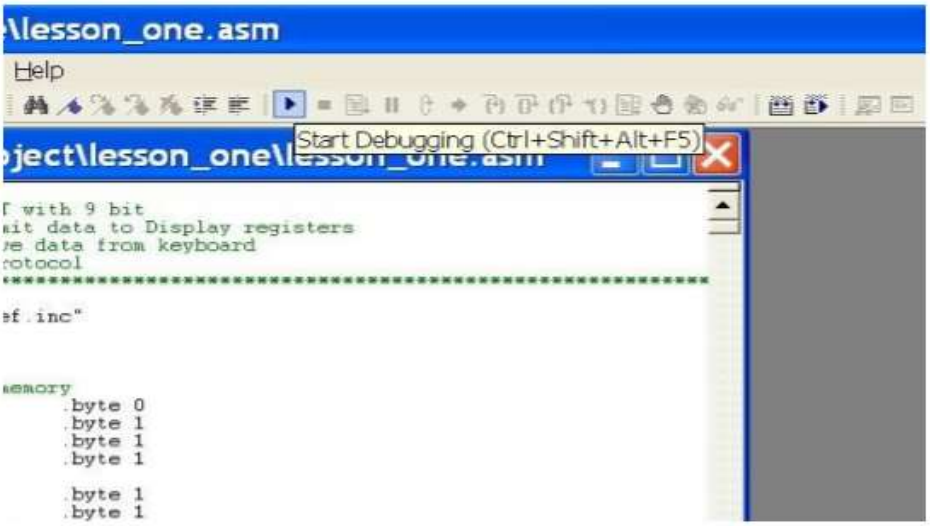


Рисунок 9 – Запуск симулятора

то в окне исходного кода программы появится маркер в виде стрелки желтого цвета, который находится на строке, которая будет выполнена в следующем шаге:

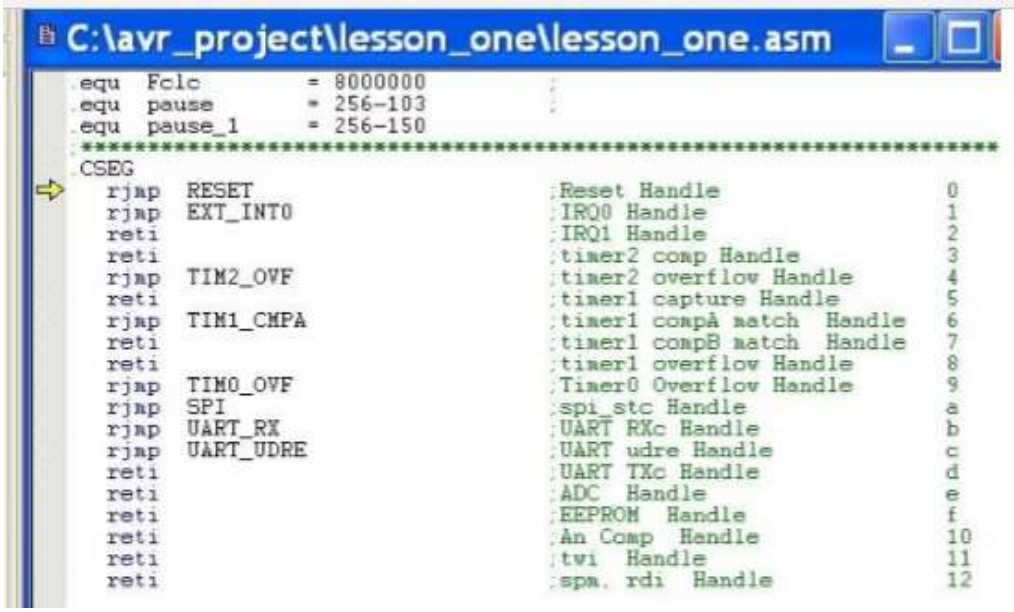


Рисунок 10 – Окно текста программы в режиме отладки

С помощью средств управления ходом выполнения программы, можно выполнять программу в пошаговом режиме, или выполнить программу до места, где стоит курсор.

Рисунок 11 – Панель инструментов отладки

Можно определять точки останова, при достижении которых выполнение программы приостановится. Для ус

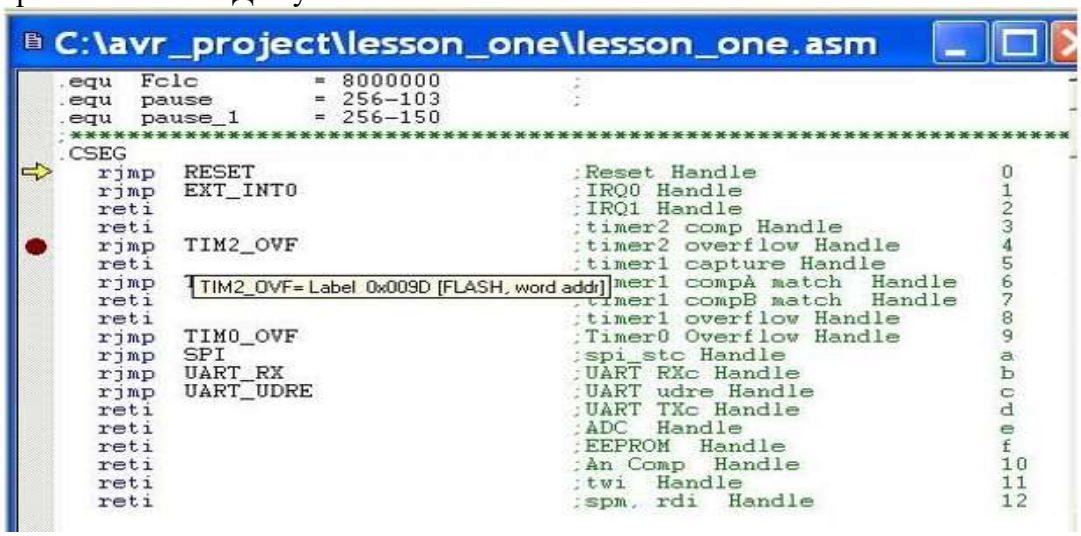


Рисунок 12 – Панель инструментов отладки

Точки останова сохраняются между сессиями работы. Если во время выполнения программы в окне исходного кода поместить курсор на имя регистра то появится подсказка о значении этого регистра, его расположении в памяти.

Для облегчения отладки существуют окна:
- Watch window;
- Register window;
- Memory window;

Watch window: Окно показывает значения и адреса заданных переменных (рисунок 13).

Рисунок 13 – Окно Watch

Переменные в окно Watch можно добавлять вручную, либо при помощи меню правой кнопки мыши (рисунок 14):

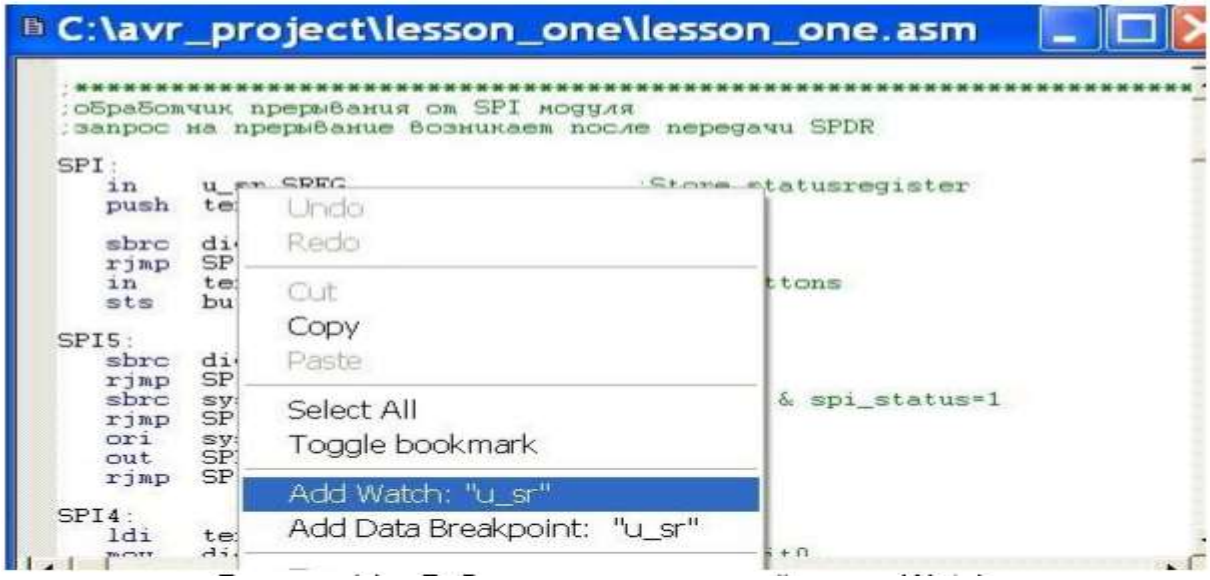


Рисунок 14 – Добавление переменной в окно Watch

Значения переменных (регистров общего назначения) можно добавлять и изменять во время остановки работы программы.
Register window:

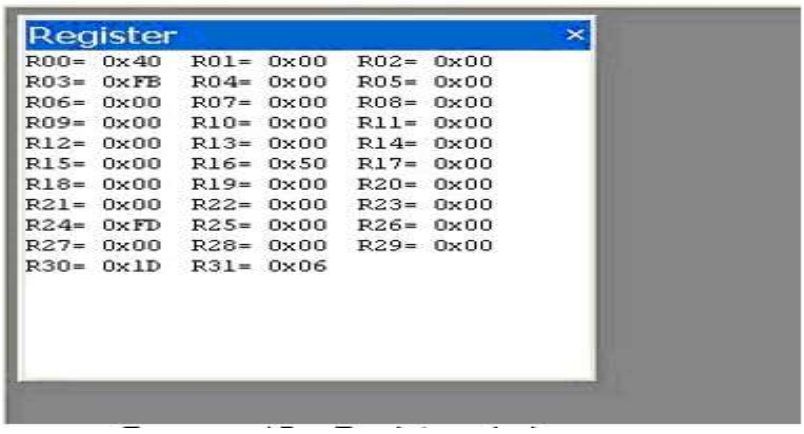


Рисунок 15 – Registerwindow

Окно показывает содержимое регистров. Регистры можно изменять во время остановки программы.

Memory windows:

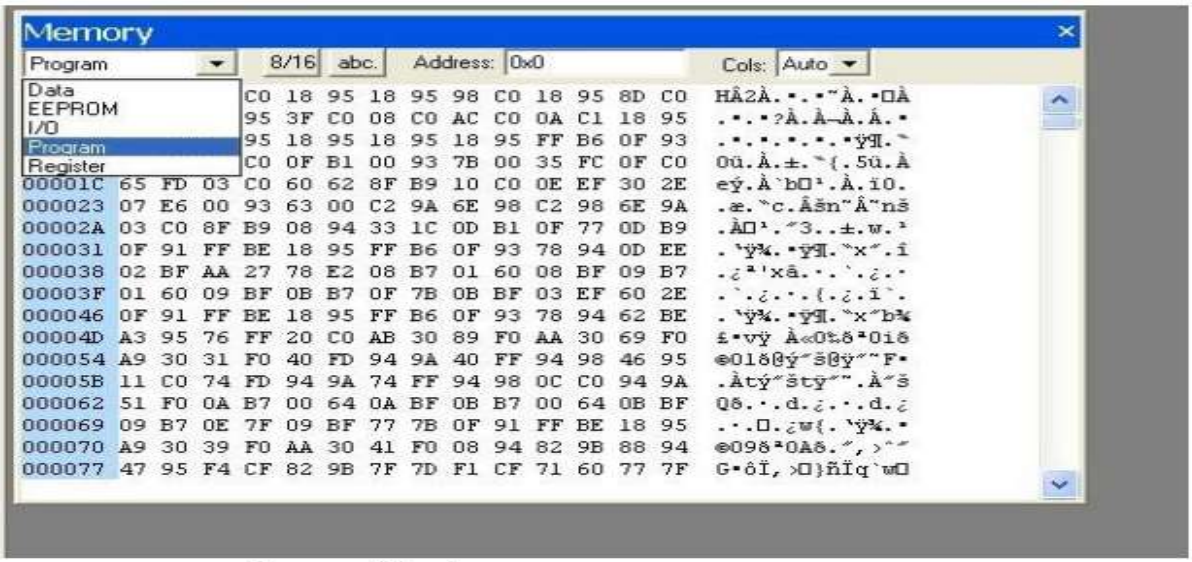


Рисунок 16 – Окно просмотра состояния памяти

Окна показывают содержимое памяти программ, данных, портов ввода/вывода и энергонезависимого ПЗУ. Память можно просматривать в HEX, двоичном или десятичном форматах. Содержимое памяти можно изменять во время остановки программы. Показывает содержимое различных регистров ввода/вывода: EEPROM, USART, таймеры и др. При первом запуске требуется настроить окна для управления и вывода необходимой информации. Во время следующей сессии работы настройки автоматически восстанавливаются.

ХОД РАБОТЫ.

Изучить теоретическую часть.
Ознакомиться с интерфейсом AVR Stidio
Выполнить проект по заданию
Ответить на контрольные вопросы

СОДЕРЖАНИЕ ОТЧЕТА

Отчет должен содержать задание на работу, блок-схему реализованного алгоритма, словесное описание хода рассуждений при решении поставленной задачи (желательно приведение таблиц, расчетов, диаграмм, и т.п.), краткое словесное описание работы программы по отдельным функциональным узлам, исходный текст программы. Исходный текст программы должен быть выполнен моноширинным шрифтом (courier, courier new).

КОНТРОЛЬНЫЕ ВОПРОСЫ:

1. Назначение пакета AVR Studio.
2. Какие основные окна AVR Studio используются в работе?
3. Что такое проект в терминах AVR Studio?
4. Как создать проект в AVR Studio?
5. Что такое трансляция (компиляция) программы? Зачем она нужна?
6. Как производить поиск ошибок компиляции?
7. Что такое симулятор (эмулятор)?
8. Как вы понимаете процесс отладки программы? Зачем он нужен?
9. Что такое точки останова (breakpoints)? Для чего они нужны
10. Как можно в процессе отладки программы убедиться в том, что значения определенных переменных, ячеек пямяти или регистров в нужные моменты принимают верные значения в процессе исполнения.? Какие средства AVR Studio для этого имеются?

ПРАКТИЧЕСКАЯ РАБОТА №6

ТЕМА Последовательная и параллельная передача информации на языке программирования

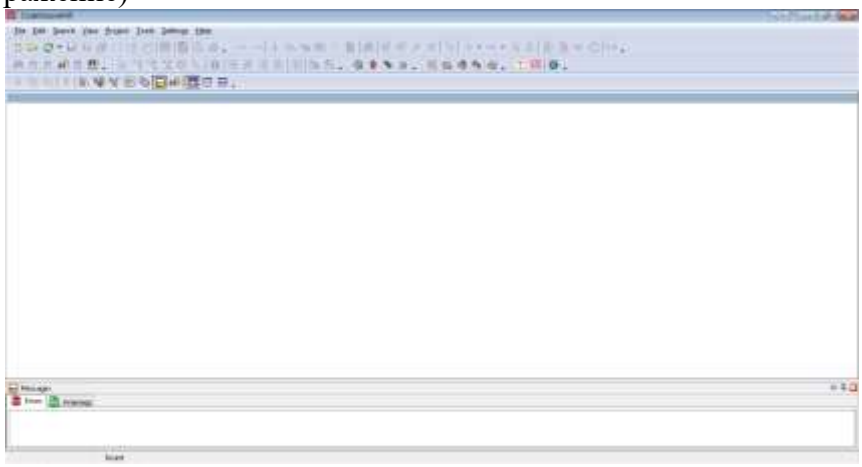
ЦЕЛЬ РАБОТЫ

Получение практических навыков по идентификации и установке процессора.

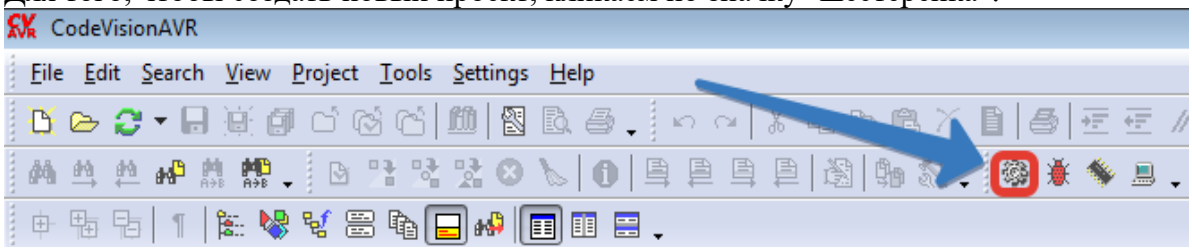
ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ.

CodeVisionAVR - это среда разработки, которая поддерживает все операции с МК AVR. В ней мы можем написать заготовку кода, скомпилировать ее и с помощью программатора "залить" в нужный нам МК. И не надо отдельно качать программку-прошиватель, компилятор и другие ненужные программы. А зачем, если все это есть в КодВижне? Также КодВижн поддерживает почти все известные программаторы для AVR, будь это китайские клоны или оригинальные программаторы. Большой плюс КодВижена в том, что он сам создает готовый шаблон программы, из которого нам нужно удалить все лишнее, что является несомненным плюсом для новичков.

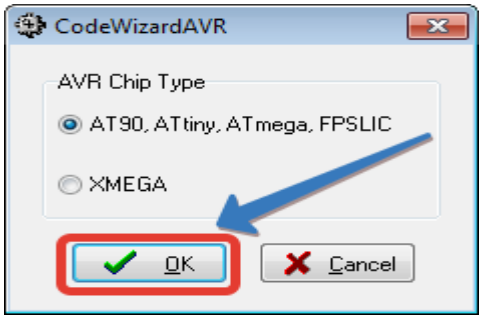
Вот так выглядит программа после ее установки (кликните по картинке, чтобы увеличить изображение)



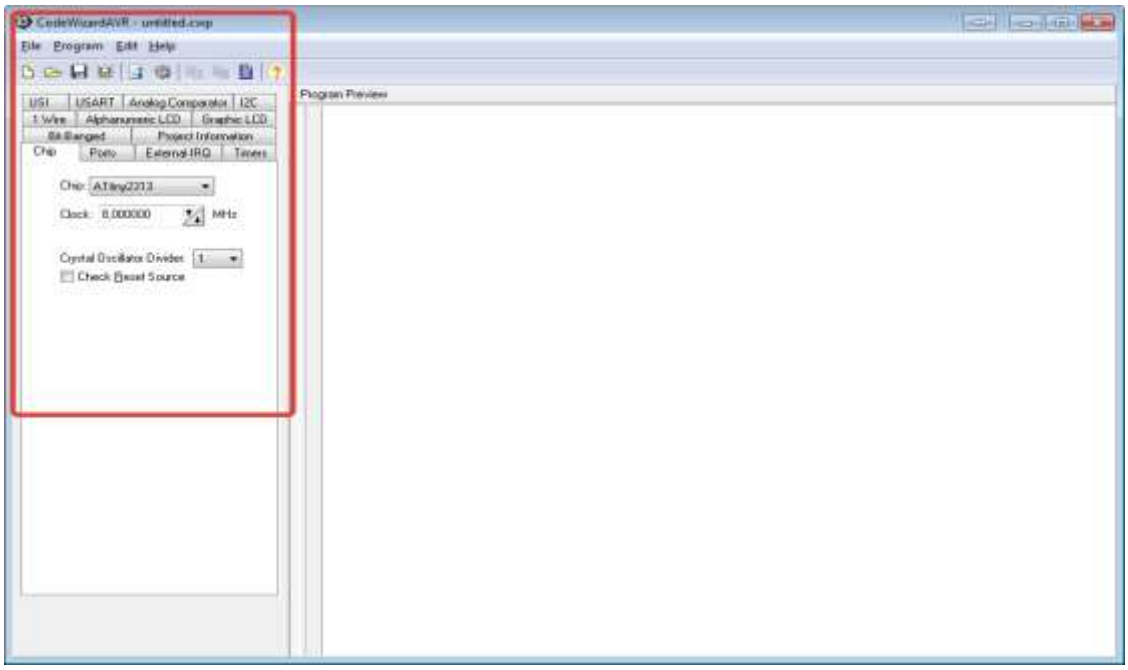
Для того, чтобы создать новый проект, кликаем по значку "шестеренка":



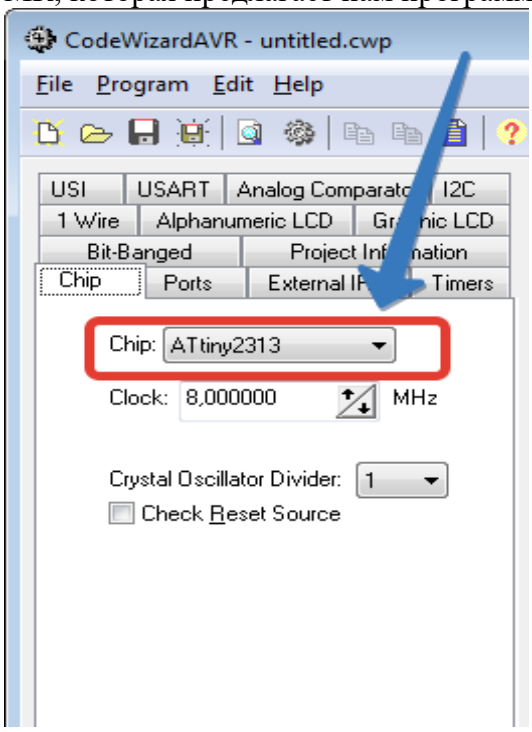
После появится вот такое окошко. Ставим маркер-точку на AT90, Atiny, FPSLIC и нажимаем "ОК".



Потом вот такое окошко. В красном прямоугольнике я показал, на какую часть окошка надо обратить внимание:

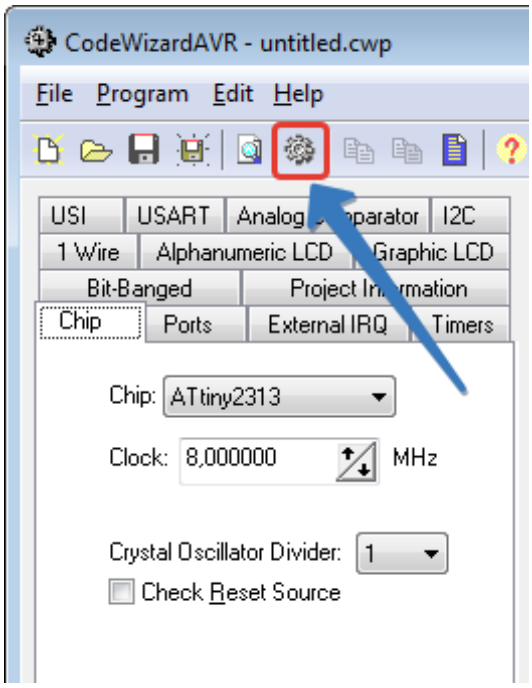


Выбираем наш МК. В данном случае я использую МК Atiny2313, поэтому из всего списка МК, которая предлагает нам программа, выбираю именно его.

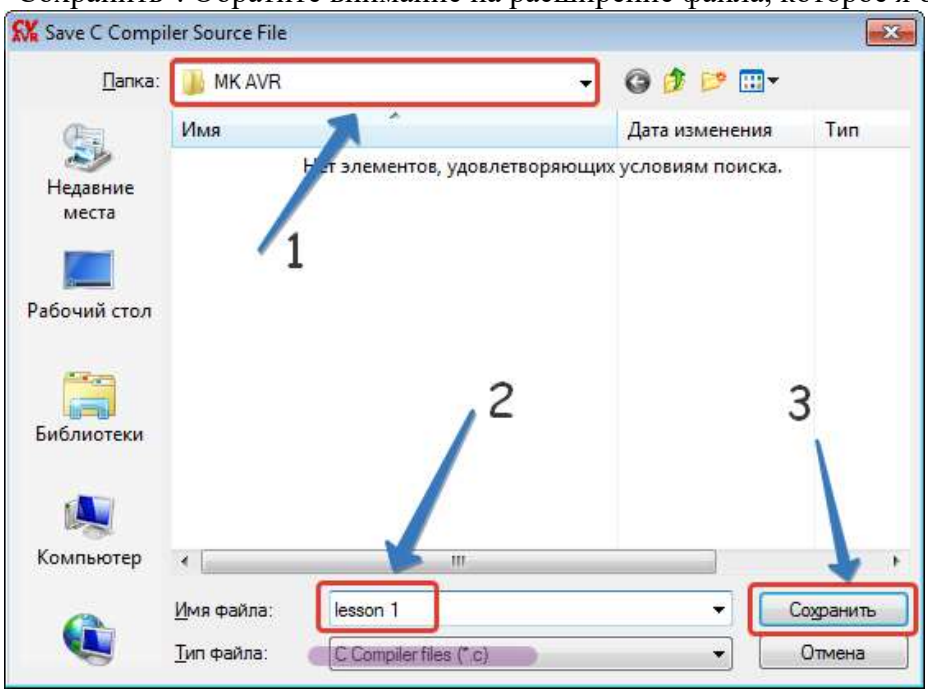


Также можно поменять частоту "Clock". По умолчанию она устанавливается на 8 МегаГерц.

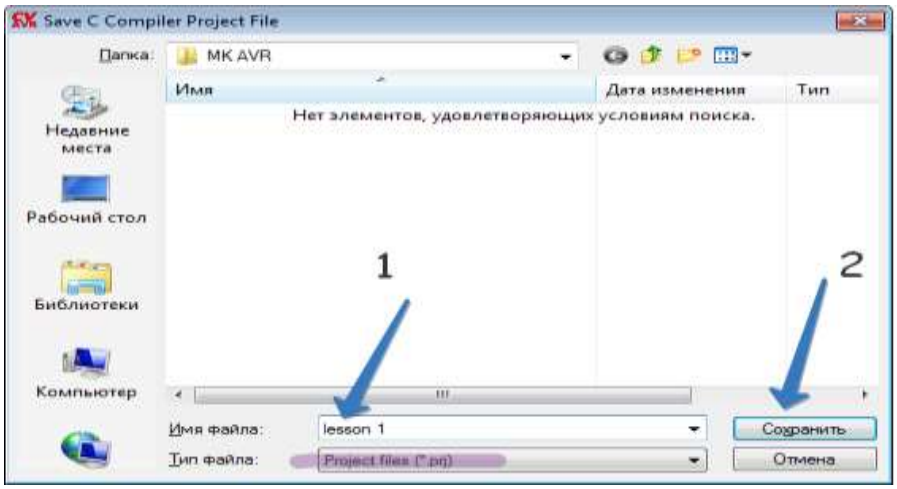
Далее нажимаем в этом же окошке на значок "шестеренки":



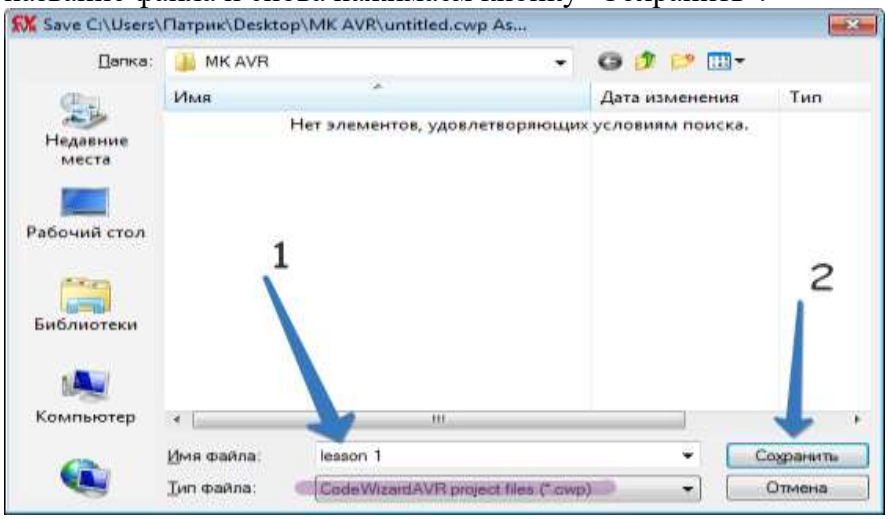
Вышло вот такое окошко. Первым делом мы выбираем папку, в которую будем сохранять наш проект. Я создал папку на рабочем столе и назвал ее MK AVR. Потом написал название нашего проекта "lesson 1", ну типа "первый урок"). Ну и потом нажимаем кнопку "Сохранить". Обратите внимание на расширение файла, которое я обвел фиолетовым цветом:



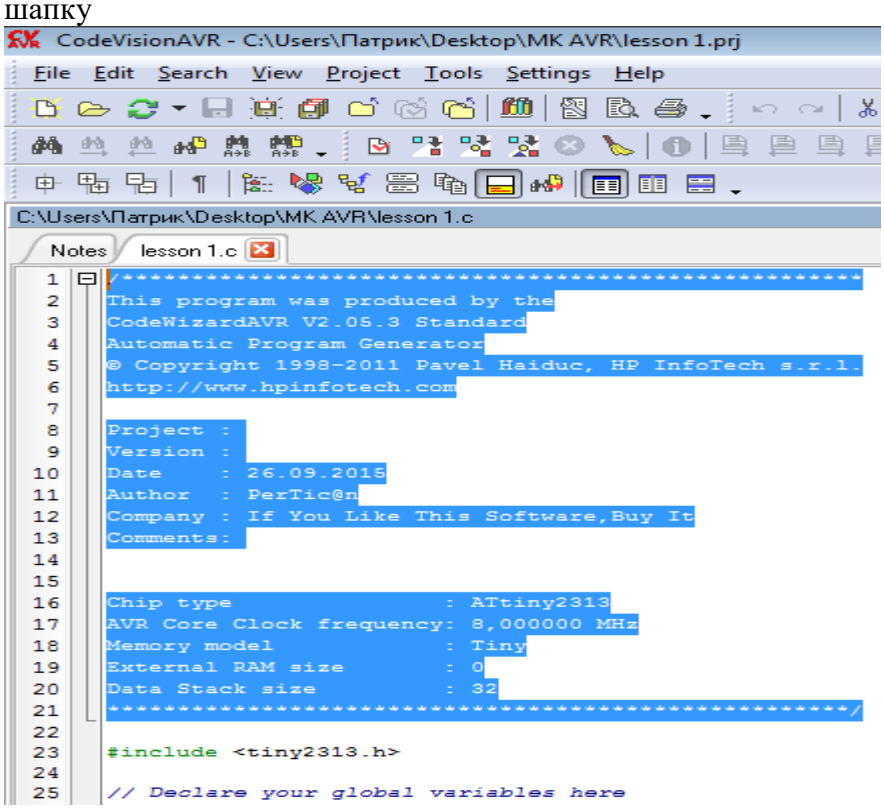
Потом выйдет еще одно такое же окошко, только расширение файла будет другим. Не теряемся, прописываем также название файла, в данном случае lesson 1, и нажимаем кнопку "Сохранить"



И потом выйдет точно такое же завершающее окошко ;-). Также не теряемся, прописываем название файла и снова нажимаем кнопку "Сохранить".

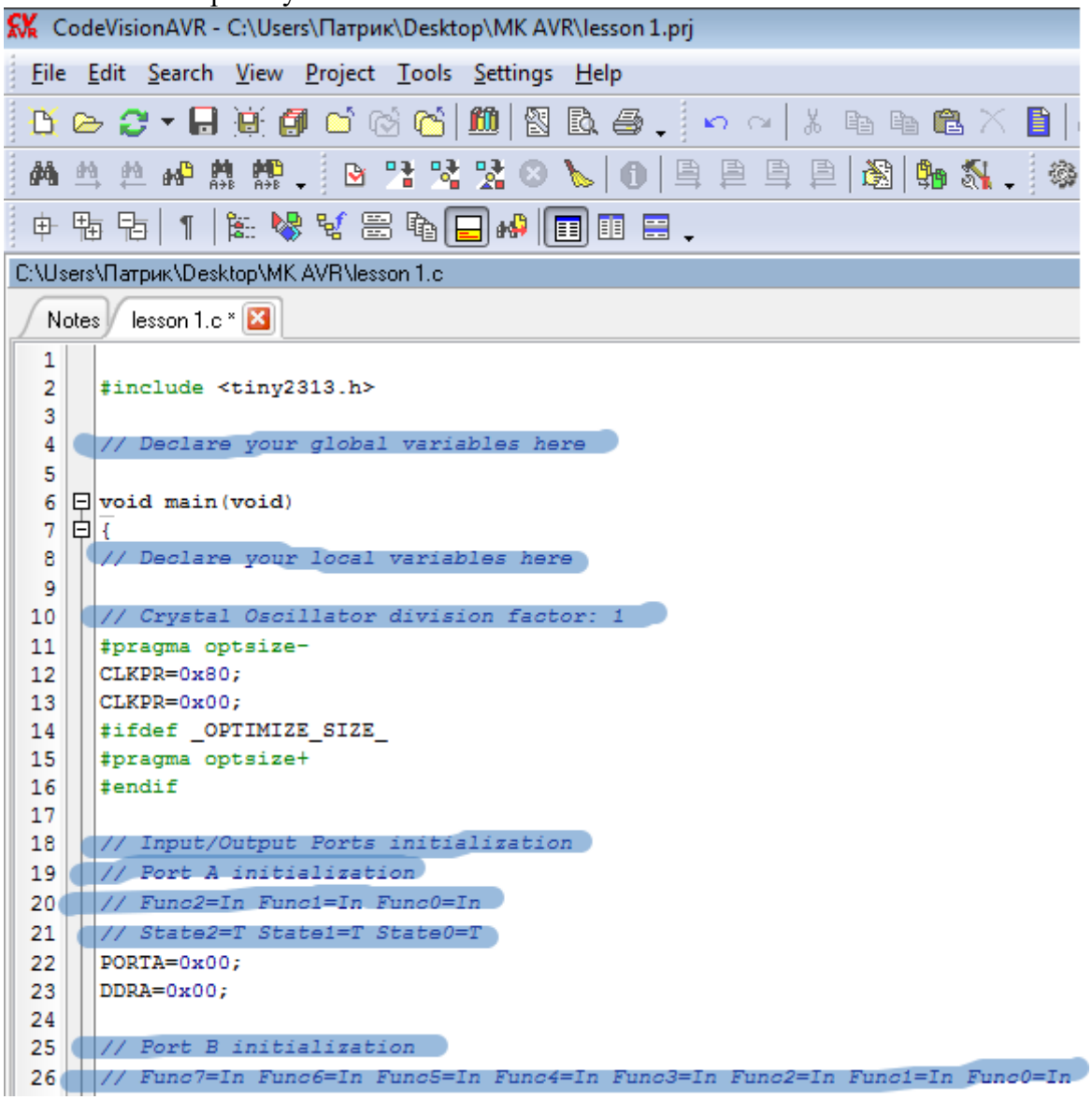


КодВижен для нас приготовил шаблон. Здесь явно много лишнего. Первым делом убираем шапку



Выделяем, нажимаем Del.

Текст, который находится после двух косых черточек "*// любой текст* " или */* любой текст */* называется *комментарием*. Они нужны для удобного восприятия и никак не сказываются на работу МК.



Проект: Наша программа будет поочередно зажигать каждый из трех светодиодов и потом в таком же порядке их тушить. У нас будет вот такой алгоритм, то есть порядок действий:

1) При подаче питания на МК зажечь первый светодиод.

2) Через 1 секунду зажечь второй светодиод, но первый светодиод при этом продолжает гореть.

3) Через 1 секунду зажечь третий светодиод, но два предыдущих также горят , то есть у нас горят сразу 3 светодиода.

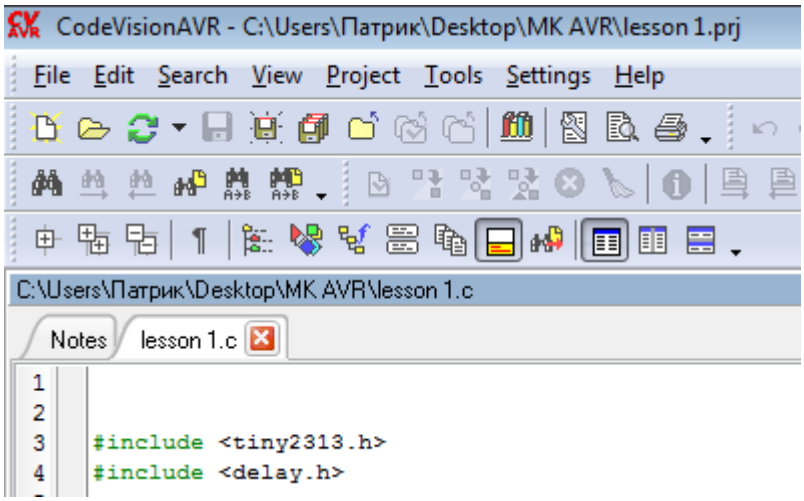
4) Через 2 секунды тухнет третий светодиод.

5) Через секунду тухнет второй светодиод.

6) Еще через секунду тухнет первый светодиод.

7) Через 2 секунды весь этот цикл повторяется с пункта 1.

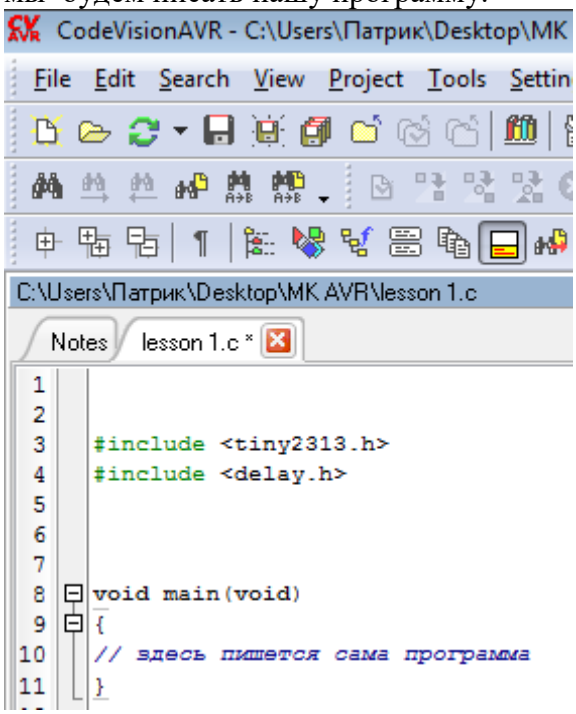
Итак, первые две строчки кода - это подключение библиотек. То есть эти файлы у нас уже есть в программе КодВижн. Нам осталось их только подключить.



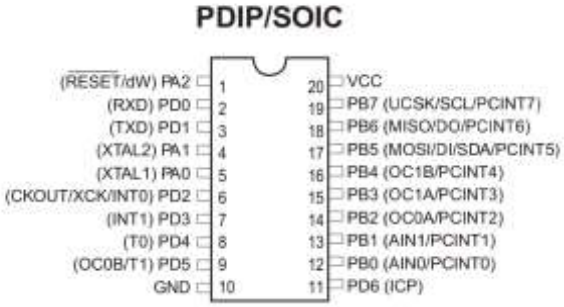
Include *с англ.* - включать, содержать, подключать. Include еще называется оператором присоединения внешних файлов.

То есть мы с вами присоединяем к нашей программе библиотеку *"tiny 2313.h"* для МК Atiny 2313, так как именно на нем будет работать наша схема. А также подключаем библиотеку *"delay.h"*. Delay - *с англ.* откладывать, отсрочивать.

Сама программа пишется после void main (void) между фигурными скобками. Там мы будем писать нашу программу.



Pinout ATtiny2313

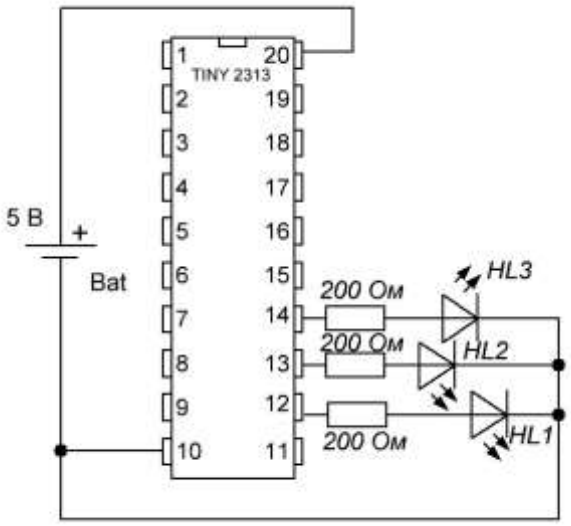


Давайте внимательно глянем на наш МК:

Здесь мы видим основное название ножек, а в скобках их двойное предназначение. Двойное предназначение нас пока не интересует. Для нас в данный момент интересны обозначения *PA, PB, PD*. Их еще называют портами Ввода-Вывода. То есть на них можно что-то вводить и с них можно что-то выводить). В данном случае я хочу использовать ножки *PB0, PB1, PB2* для того, чтобы управлять светодиодами. На каждую из этих ножек я повешу по одному

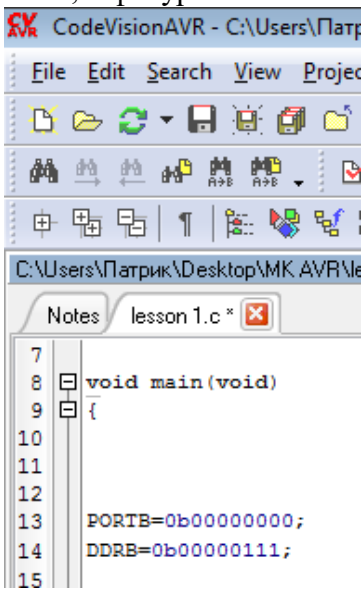
светодиоду, итого получается три. Остальные ножки нас не интересуют, так как мы не будем их использовать.

Наша схема будет выглядеть вот таким образом:



Следовательно, я должен настроить эти ножки для управления светодиодами.

Итак, в фигурных скобках пишем:



PORTB=0b00000000

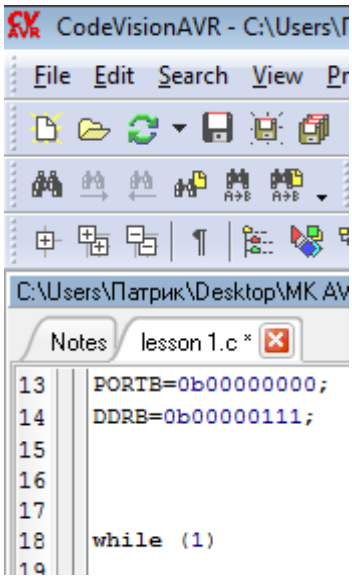
0b - это бинарная система счисления. Настраивать в бинарной системе удобно тем, что каждый бит - это одна ножка МК. Поэтому счет идет **справа-налево**. Присмотритесь к рисунку выше. Здесь мы видим PORTB и нули при двоичной ситеме исчисления. Что это означает? А означает это то, что эти выводы при неиспользовании их в программе будут притянуты к нулю, то есть на них будет низкий потенциал или ноль Вольт. В случае, если мы видим единички установленные на каком-либо из битов, то это значит, что на этом порту при неипользовании в программе у нас будет высокий потенциал или плюс 5 вольт, или, говоря другими словами, этот вывод у нас будет принят к плюсу питания.

DDRB=0b00000111

Это означает, что *PB0, PB1, PB2* настроены на выход. Считаем справа-налево, если вы не забыли. А если ноль, то значит настроены на вход.

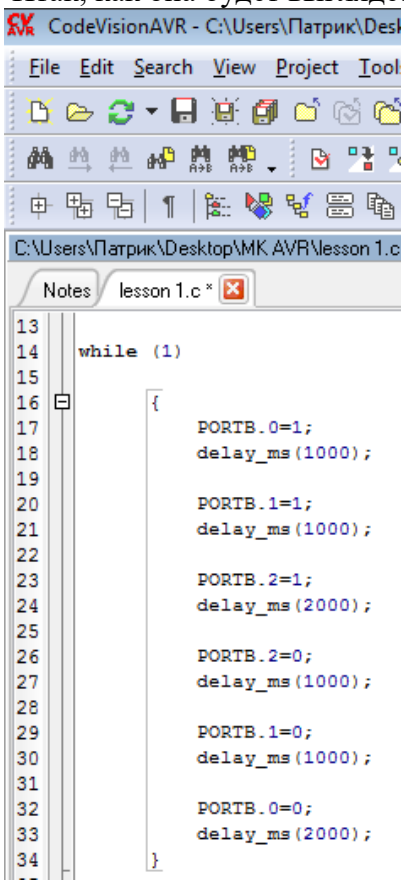
После каждой строчки не забывайте ставить знак *" ; "* , иначе при компиляции (превращения вашей программы в код, который понимает МК), будут ошибки.

Теперь рассмотрим *while (1)*. Он означает, что кусочек программы, который будет написан после него, будет зацикливаться, то есть повторяться бесконечное количество раз.



Ну и собственно можно начать писать саму программу...

Итак, как она будет выглядеть?



Смотрите, программа находится между фигурными скобками. Эти скобки принадлежат *while (1)*. То есть этот кусок программы будет повторяться бесконечное число раз.

Давайте разберем саму программу...

PORTB.0=1; *Это означает, что при подачи питания на МК, у нас ножка PB0 будет иметь логическую единицу , то есть на выходе этой ножки будет 5 Вольт, которые зажгут светодиод.*

delay_ms(1000); Ждем 1000 миллисекунд, то есть 1 секунду.

PORTB.1=1; *Через одну секунду на ножке PB1 появляется логическая единица, то есть 5 вольт, которые и зажгут второй светодиод*

delay_ms(1000); *Ждем еще одну секунду.*

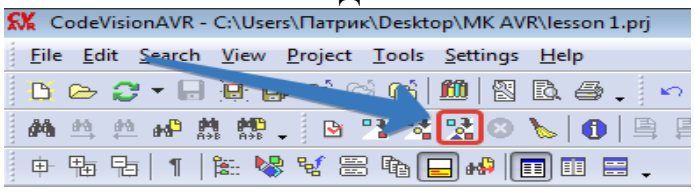
PORTB.2=1; *Как только прошла секунда, у нас на ножке PB2 появляется логическая единичка, и, следовательно, зажигается третий светодиод.*

delay_ms(2000); *Ждем 2 секунды...Итого у нас горят все разом три светодиода в течение двух секунд.*

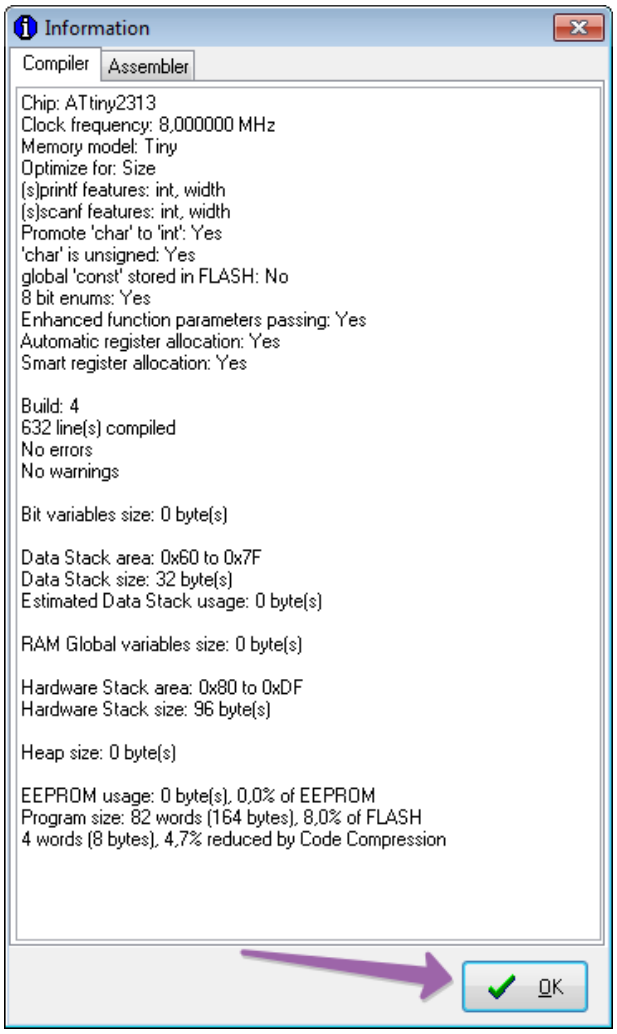
PORTB.2=0; *На ножке PB2 появляется логический ноль, то есть напряжение исчезает. Третий светодиод тухнет.*
delay_ms(1000); *Ждем секунду*
PORTB.1=0; *Исчезает напряжение на ножке PB1. Светодиод второй тоже перестает источать свет.*
delay_ms(1000); *Ждем секунду.*
PORTB.0=0; *И на ножке PB0 напряжение тоже стает равно нулю. Первый светодиод тухнет тоже.*
delay_ms(2000); *Ждем две секунды и возвращаемся на начало программы, то есть к*
PORTB.0=1;
delay_ms(1000);
Вот так выглядит полностью сама программа. Ее можно даже скопировать и скомпилировать.

```
#include <tiny2313.h>
#include <delay.h>

void main(void)
{
PORTB=0b00000000;
DDRB=0b00000111;

while (1)
    {
      PORTB.0=1;
      delay_ms(1000);
             PORTB.1=1;
      delay_ms(1000);
              PORTB.2=1;
      delay_ms(2000);
               PORTB.2=0;
      delay_ms(1000);
            PORTB.1=0;
      delay_ms(1000);
                  PORTB.0=0;
      delay_ms(2000);
    }
}
```

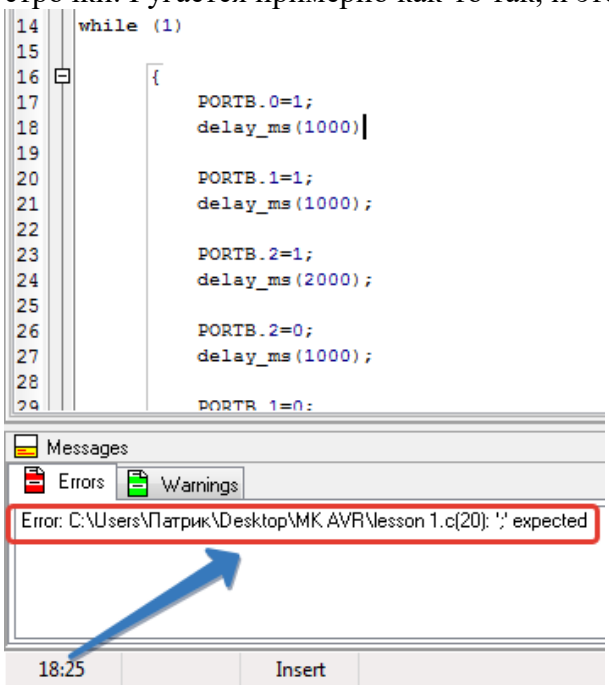
Ну все, программа написана. Осталось дело за малым: преобразовать ее в машинный код, чтобы его понимал МК. Для этого мы нажимаем на кнопочку "создать все файлы проекта"



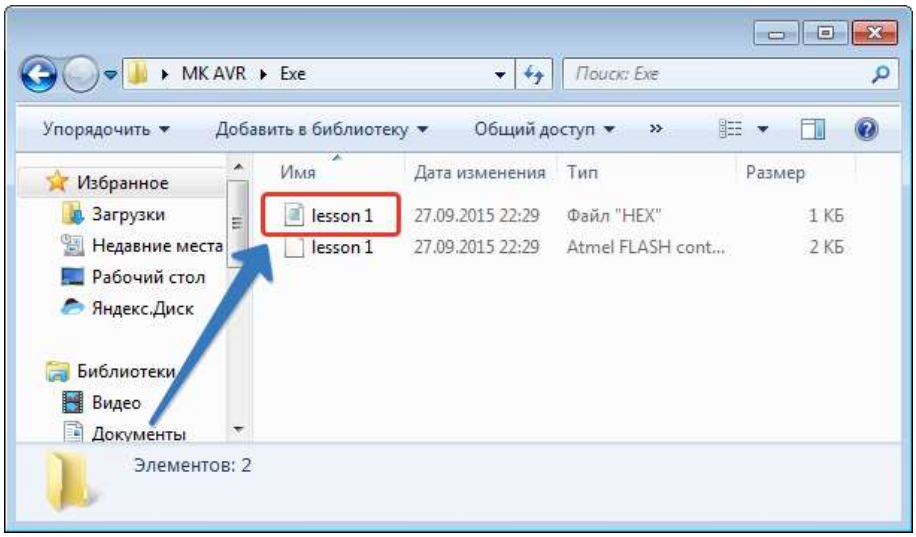
После нажатия на эту кнопку, у нас выскакивает окошко, в котором говорится о том, успешно ли прошла компиляция и тд. Если все нормально, то нажимаем ОК.



Если что-то не так, то отобразятся ошибки. Я специально убрал знак " ; " после одной строчки. Ругается примерно как-то так, и это не есть хорошо:



Если все ОК, то у вас в папке "Exe", там где у вас все проекты будет лежать готовый файлик, в данном случае "lesson 1. hex".



Теперь этот файлик с удовольствием скушает наш МК AVR и будет выполнять программку, которая находится в этом файле.

ХОД РАБОТЫ.

Изучить теоретическую часть.
Ознакомиться с интерфейсом AVRStidio
Выполнить проект по заданию
Ответить на контрольные вопросы

СОДЕРЖАНИЕ ОТЧЕТА

Отчет должен содержать задание на работу, блок-схему реализованного алгоритма, словесное описание хода рассуждений при решении поставленной задачи (желательно приведение таблиц, расчетов, диаграмм, и т.п.), краткое словесное описание работы программы по отдельным функциональным узлам, исходный текст программы. Исходный текст программы должен быть выполнен моноширинным шрифтом (courier, courier new).

ПРАКТИЧЕСКАЯ РАБОТА №7

ТЕМА Работа с массивами на языке программирования

ЦЕЛЬ РАБОТЫ

Получение практических навыков по идентификации и установке процессора.

.

ХОД РАБОТЫ.

1. Убедитесь в том, что компьютерная система обесточена (при необходимости, отключите систему от сети).
2. Разверните системный блок задней стенкой к себе.
3. По наличию или отсутствию разъемов USB установите форм-фактор материнской платы (при наличии разъемов USB - форм-фактор ATX, при их отсутствии -AT).
4. Установите местоположение и снимите характеристики следующих разъемов:
   - питания системного блока;
   - питания монитора;
   - сигнального кабеля монитора;
   - клавиатуры;
   - последовательных портов (два разъема);
   - параллельного порта;
   - других разъемов.
5. Убедитесь в том, что все разъемы, выведенные на заднюю стенку системного блока, не взаимозаменяемы, то есть каждое базовое устройство подключается одним единственным способом.
6. Изучите способ подключения мыши.
   Мышь может подключаться к разъему последовательного порта или к специальному порту PS/2, имеющему разъем круглой формы. Последний способ является более современным и удобным. В этом случае мышь имеет собственный выделенный порт, что исключает возможность ее конфликта с другими устройствами, подключаемыми к последовательным портам. Последние модели могут подключаться к клавиатуре через разъем интерфейса USB.

7. Заполните таблицу:

Разъем	Тип разъема	Количество контактов	Примечания

8. Определить наличие основных устройств персонального компьютера.
9. Установите местоположение блока питания, выясните мощность блока питания (указана на ярлыке).
10. Установите местоположение материнской платы.
11. Установите характер подключения материнской платы к блоку питания.
    Для материнских плат в форм-факторе AT подключение питания выполняется двумя разъемами. Обратите внимание на расположение проводников черного цвета - оно важно для правильной стыковки разъемов.
12. Установите местоположение жесткого диска.
    Установите местоположение его разъема питания. Проследите направление шлейфа проводников, связывающего жесткий диск с материнской платой. Обратите внимание на местоположение проводника, окрашенного в красный цвет (на жестком диске он должен быть расположен рядом с разъемом питания).
13. Установите местоположения дисководов гибких дисков и дисковода CD-ROM.

Проследите направление их шлейфов проводников и обратите внимание на положение проводника, окрашенного в красный цвет, относительно разъема питания.

14. Установите местоположение платы видеоадаптера.
    Определите тип интерфейса платы видеоадаптера.
15. При наличии прочих дополнительных устройств выявите их назначение, опишите характерные особенности данных устройств (типы разъемов, тип интерфейса и др.).
16. Заполните таблицу:

Устройство	**Характерные особенности**	**Куда и при помощи чего подключается**

СОДЕРЖАНИЕ ОТЧЕТА

Отчет должен содержать результаты всех исследований в рамках практического занятия.

ПРАКТИЧЕСКАЯ РАБОТА №8

ТЕМА: Написание программ с использованием подпрограмм
ЦЕЛЬ "Изучить особенности выполнения команд вызова подпрограмм и возвращения из подпрограмм. Разработка алгоритма, составление и отладка программы с использованием этих команд".

ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

При разработке сложных программ очень часто приходится выполнять однотипные действия, но с разными значениями. Например, обрабатывать показания датчиков, преобразовывать числа из двоичной системы исчисления в десятичную и т.д. Наиболее просто это сделать с использованием подпрограмм.

Мнемокод	Операнд	Комментарий
CALL	ADR (метка)	Производится безусловный переход к подпрограмме по адресу ADR или на метку ADR → (IP), (IP) → (SP)
RET		Производится безусловный возврат из подпрограммы (SP) → (IP)

Задание на лабораторную работу № 5

Вариант 1

Заполнить две области памяти ds:2100H - ds:2150H и ds:2200H - ds:2250H последовательностью констант от 0H до 50H, если число, находящееся в регистре BL четное, в противном случае эти области заполнить последовательностью констант от 50H до 0H.

Вариант 2

Если $7^{ой}$ бит аккумулятора равен 1, то заполнить константой 45H две области памяти длиной 65 байт начиная с адресов ds:2100H и ds:2200H, если $7^{ой}$ бит равен 0, то эти две области заполнить константой 55H.

Вариант 3

Если в аккумуляторе находится число 58H, заполнить константой B1H две области памяти длиной 75 байт, начиная с адресов ds:2100H и ds:2200H, в противном случае эти две области заполнить константой B2H.

Вариант 4

Заполнить две области памяти длиной 45 байт начиная с адресов ds:2100H, ds:2200H последовательностью констант начиная с 1H, если число находящееся в ячейке памяти ds:2250H отрицательное, в противном случае эти области заполнить константой A5H.

Вариант 5

Если нулевой бит аккумулятора равен 1, то заполнить константой F1H две области памяти длиной 120 байт начиная с адресов ds:2100H и ds:2200H, если нулевой бит равен 0, то эти две области заполнить константой F2H.

Вариант 6

Если число в аккумуляторе четное, заполнить константой 30H две области памяти длиной 110 байт, начиная с адресов ds:2200H и ds:2300H, в противном случае эти две области заполнить константой 90H.

Вариант 7

Если второй бит аккумулятора равен 1, то заполнить константой 55H две области памяти длиной 65 байт начиная с адресов ds:2200H и ds:2300H, если второй бит равен 0, то эти две области заполнить константой 33H.

Вариант 8

Если число в аккумуляторе четное, заполнить константой C9H две области памяти длиной 72 байт, начиная с адресов ds:2100H и ds:2200H, в противном случае эти две области заполнить последовательностью констант, начиная с 1H.

Вариант 9

Если в аккумуляторе находится число 20H, заполнить константой E5H две области памяти длиной 80 байт, начиная с адресов ds:2100H и ds:2200H, в противном случае эти две области заполнить константой D6H.

Вариант 10

Заполнить две области памяти длиной 85 байт, начиная с адресов ds:2100H, ds:2200H последовательностью констант начиная с 0H, если $5^{ый}$ и $2^{ой}$ бит аккумулятора равны 0, в противном случае заполнить эти области константой 50H.

Вариант 11

Если число в аккумуляторе отрицательное заполнить две области памяти длиной 15 байт начиная с адресов ds:3000H, ds:3010H последовательностью констант 01H - 0FH , в противном случае заполнить эти области памяти заполнить последовательностью констант 0FH - 01H. Вывести число, находящееся в аккумуляторе в ячейку памяти ds:3020H.

Вариант 12

Если число в регистре AL равно числу записанному в регистре BL заполнить две области памяти длиной 120 байт начиная с адресов ds:3000H, ds:3100H константой AAH, если число в регистре AL меньше числа в регистре B заполнить эти две области памяти константой ABH , если число в регистре AL больше числа в регистре B заполнить эти две области памяти константой BAH. Вывести числа находящееся в регистре AL и в регистре BL в ячейки памяти ds:3200H и ds:3201H.

Вариант 13

Если число в аккумуляторе 40H заполнить две области памяти длиной 20 байт, начиная с адресов ds:3000H, ds:3020H последовательностью констант начиная с 1H, в противном случае заполнить эти области памяти заполнить константой A5H. Вывести число находящееся в аккумуляторе в ячейку памяти ds:3040H.

Вариант 14

Если $3^{ий}$ и $6^{ой}$ бит аккумулятора равны 0, заполнить две области памяти длиной 20 байт, начиная с адресов ds:3000H, ds:3020H последовательностью констант начиная с 1H, в противном случае заполнить эти области памяти заполнить константой 40H. Вывести число, находящееся в аккумуляторе в ячейку памяти ds:3040H.

Вариант 15

Заполнить две области памяти ds:0008H – ds:0017H и ds:0020H – ds:002FH последовательностью констант от 0H до FH, если число, находящееся в регистре CL четное, в противном случае эти области заполнить последовательностью констант от FH до 0H.

ХОД РАБОТЫ.

Изучите теоретический материал
Произведите настройку ПК

СОДЕРЖАНИЕ ОТЧЕТА

В отчете напишите алгоритм настройки BIOS

ПРАКТИЧЕСКАЯ РАБОТА №9

ТЕМА: Реализация математических операций на языке программирования

ЦЕЛЬ РАБОТЫ

Получение практических навыков по идентификации и установке процессора.

ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ.

Процесс тестирования можно разделить на отдельные части, называемые элементарными проверками. Элементарная проверка состоит в подаче на объект тестового воздействия и в измерении (оценке) ответа объекта на это воздействие.

Алгоритм тестирования определяется как совокупность и последовательность элементарных проверок вместе с определенными правилами анализа результатов последних с целью отыскания места в объекте, параметры которого не отвечают заданным значениям.

Таким образом, диагностика — это тоже контроль, но контроль последовательный, направленный на отыскание неисправного места (элемента) в диагностируемом объекте.

Обычно тестирование начинается по сигналу ошибки, выработанному схемами контроля ПК. Диагностическое программное обеспечение чрезвычайно необходимо в том случае, если система начинает сбоить или если осуществляется модернизация системы, добавляя новые устройства.

Диагностические программы можно разделить на три уровня:

- Тестовые средства ПК (тест POST)
- Системные средства (средства ОС)
- Дополнительные программы, которые либо поставляются вместе с компьютером, либо приобретаются у его изготовителя.

Дополнительные программы можно разделить на:

о Информационные программы — Которые тестируют компьютер или отдельные компоненты, и выдают подробную информацию о его состоянии, функциональности, и возможных программных и физических неполадках.

о Тестовые программы. — Которые работают по принципу максимальной загрузки различными операциями, эмулирующими работу пользователя за компьютером, и замеряют общую производительность системы или производительность отдельных компонентов на основе сравнения, с уже имеющейся базой данных. Выполняя тестирование отдельных элементов или системы в целом.

1. **Atomic Cpu Test** – Утилита для проверки производительности вашего процессора и отдельных его составляющих (кэш-память, арифметическо-логическое устройство).
2. **CPU-Z 1.59 –** Очень полезная программа, определяющая информацию о процессоре, чипсете материнской платы и памяти.
3. **AIDA64 (EVEREST)** является мощнейшим средством для анализа начинки компьютера (железо, софт, сеть), тестирования производительности и мониторинга состояния ключевых узлов системы.
4. **RightMark Memory Analyzer 3.5** — новая версия тестового пакета с новым тестом стабильности функционирования подсистемы памяти
5. **MemTest** — утилита предназначена для тестирования надежности работы оперативной памяти. При тестировании оценивается способность памяти записывать и считывать данные. Есть возможность задавать количество мегабайт для тестирования.
6. **TestVideoRAM** предназначен для тестирования видеопамяти на картах от nVidia! Полное описание работы можно найти на страничке программы.
7. **Victoria**,– предназначенна для глубокого тестирования состояния жесткого диска.
8. **MHDD–** предназначенна для тестирования жесткого диска.
9. **System Information for Windows** — программа для предоставления детальной информации о компьютере. Показывает информацию о материнской плате, BIOS, процессоре, жестких дисках, установленных устройствах.

10. **SiSoftware SANDRA–** нформационная и диагностическая программа, которая предоставляет подробнейшую информацию об аппаратном и программном обеспечении компьютера. В процессе работы Sandra тестирует компьютер и сравнивает полученные результаты с эталонными данными.
11. **HDDSpeed v2.3.2** – Тестирование реальной скорости жестких дисков
12. **HDDScan –** Программа предназначена для проверки носителей информации на наличие сбойных блоков, просмотра S.M.A.R.T. атрибутов, изменения специальных настроек, таких как: управление питанием, старт/стоп шпинделя, регулировка акустического режима и др.

ХОД РАБОТЫ.

1 Знакомство с ПО или утилитой согласно варианту, дать описание описание.
2 Опишите основные функции работы утилиты с пояснениями и скриншотами.
3 Выполните проверку устройства с пояснениями и скриншотами.
4 Опишите принцип работы заданной утилиты с пояснениями и скриншотами.

СОДЕРЖАНИЕ ОТЧЕТА

В отчете должны присутствовать:
1. Краткое описание ПО.
2. Принципы настройки, установки.
3. Примеры работы со скриншотами.

ПРАКТИЧЕСКАЯ РАБОТА №10

ТЕМА Создание программного продукта

ЦЕЛЬ: Изучить особенности программирования с помощью команд обработки строк символов .

ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

Мнемокод	Операнд	Комментарий
movs	адр.пр.,адр.ист.	Команда копирует байт, слово или двойное слово из цепочки, адресуемой операндом адрес_ист, в цепочку, адресуемую операндом адрес_пр.
movsb	адр.пр.,адр.ист.	переслать цепочку байт;
movsw	адр.пр.,адр.ист.	переслать цепочку слов
movsd	адр.пр.,адр.ист.	переслать цепочку двойных слов
cld		очистить флаг направления. Команда сбрасывает флаг направления df в 0.
std		установить флаг направления. Команда устанавливает флаг направления df в 1.
cmps	адр.пр.,адр.ист.	Сравнение цепочек
cmpsb	адр.пр.,адр.ист.	Сравнить строку байт
cmpsw	адр.пр.,адр.ист.	Сравнить строку слов
cmpsd	адр.пр.,адр.ист.	Сравнить строку двойных слов
scas	адр_приемн.	Сканирование цепочек байт
scasb	адр_приемн.	Сканирование цепочек байт
scasw	адр_приемн.	Сканирование цепочек слов
scasd	адр_приемн.	Сканирование цепочек двойных слов
lods	адр_ист.	загрузка элемента из цепочки
lodsb	адр_ист.	загрузить байт из цепочки в AL
lodsw	адр_ист.	загрузить слово из цепочки в AX
lodsd	адр_ист.	загрузить слово из цепочки в EAX
stosb	адр_приемн.	Сохранение из регистра AL в цепочке
stosw	адр_приемн.	сохранение из регистра AX в цепочке
stosd	адр_приемн.	сохранение из регистра EAX в цепочке
rep		Префикс повторения используется с командами movs и stos. Префикс rep заставляет данные команды выполняться, пока содержимое в ecx/cx не станет равным 0. При этом цепочечная команда, перед которой стоит префикс, автоматически уменьшает содержимое ecx/cx на единицу. Та же команда, но без префикса, этого не делает
repe или **repz**		Цепочная команда выполняется до тех пор, пока содержимое ecx/cx не равно нулю или флаг zf равен 1, иначе управление передается следующей команде программы. Наиболее эффективно эти префиксы можно использовать с командами cmps и scas для поиска отличающихся элементов цепочек.
Repne или **repnz**		Цепочная команда выполняется до тех пор, пока содержимое ecx/cx не равно нулю или флаг zf равен 0, иначе управление передается следующей команде программы. Данные префиксы можно использовать с командами cmps и scas для поиска совпадающих элементов цепочек.

адрес_источника — пара ds:esi/si;
адрес_приемника — пара es:edi/di.

Вариант 1

Сравнить две строки длиной 15 байт и поместить первый не совпавший байт в памяти в яч. памяти 0020H, второй в следующую и т.д. Если первый байт > второго поместить его в ячейку памяти 0041H. Вывести все данные и результаты на экран.

Вариант 2

Сравнить две строки длиной 20H байт и поместить первый не совпавший байт в памяти поместить в яч. памяти 0150H, второй в следующую и т.д. Если первый байт < второго поместить его в ячейку памяти 0149H. Вывести все данные и результаты на экран.

Вариант 3

Сравнить две строки длиной 10 байт и поместить первый совпавший байт в AL. Если первый байт четный вывести его на экран. В противном случае поменять содержимое строк местами. Вывести данные и результаты на экран.

Вариант 4

Сравнить две строки длиной 13 байт и поместить не совпавшие байты местами, если первый не совпавшие байт = 45H. Вывести все данные и результаты на экран.

Вариант 5

Сравнить две строки длиной 13 байт и поместить не совпавшие байты поместить в строку 3, если первый не совпавший байт нечетный. В случае полного совпадения переслать строку 2 в строку 3 Вывести все данные и результаты на экран.

Вариант 6

Сравнить две строки длиной 8 байт и поместить второй не совпавший байт в памяти в яч. памяти в AL. Если этот байт нечетный поменять строку 1 и строку 2 местами. Вывести все данные и результаты на экран.

Вариант 7

Сравнить две строки длиной 6 байт и поместить первый не совпавший байт в памяти поместить в яч. памяти 0150H, второй в следующую и т.д. Если первый байт > второго поместить его в ячейку памяти 0149H. Вывести все данные и результаты на экран.

Вариант 8

Сравнить две строки длиной 14 байт и поместить первый совпавший байт в AL. Если первый совпавший байт четный вывести его на экран. В противном случае поменять содержимое строк местами. Вывести данные и результаты на экран.

Вариант 9

Сравнить две строки длиной 11 байт и поместить не совпавшие байты местами, если первый не совпавшие байт четный. Вывести все данные и результаты на экран.

Вариант 10

Сравнить две строки длиной 10 байт и поместить не совпавшие байты в строку 3, если первый не совпавший байт нечетный. В случае полного совпадения обменять первые половины строк 1 и 2. Вывести все данные и результаты на экран.

Вариант 11

Сравнить две строки длиной 12 байт и поместить совпавшие байты в строку 3, если первый совпавший байт четный. В случае полного совпадения обменять вторые половины строк 1 и 2. Вывести все данные и результаты на экран.

ХОД РАБОТЫ.

1 Знакомство с ПО или утилитой согласно варианту, дать описание описание.
2 Опишите основные функции работы утилиты с пояснениями и скриншотами.
3 Выполните проверку устройства с пояснениями и скриншотами.
4 Опишите принцип работы заданной утилиты с пояснениями и скриншотами.

СОДЕРЖАНИЕ ОТЧЕТА

В отчете должны присутствовать:

1. Краткое описание ПО.
2. Принципы настройки, установки.
3. Примеры работы со скриншотами.

ПРАКТИЧЕСКАЯ РАБОТА №10

ТЕМА Комплексная работа по программированию на языке программирования

ЦЕЛЬ: Изучить особенности выполнения арифметических команд, разработка алгоритма, составление и отладка программы с использованием этих команд.

ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

Часть 1

Мнемокод	Операнд	Комментарий
ADD	Op1,Op2	Сложение значений Op1 и Op2, результат помещается в Op1. (Op1)+(Op2) → (Op1)
ADC	Op1,Op2	Сложение значений Op1 и Op2 с учетом флага переноса CF, результат помещается в Op1. (Op1)+(Op2)+(CF) → (Op1)
SUB	Op1,Op2	Вычитание значений Op1 и Op2, результат помещается в Op1. (Op1)-(Op2) → (Op1)
SBB	Op1,Op2	Вычитание значений Op1 и Op2 с учетом заема, результат помещается в Op1. (Op1)-(Op2)-(CF) → (Op1)
INC	Op	Инкремент операнда Op . Содержимое Op увеличивается на1. (Op) + 1→ (Op)
DEC	Op	Декремент операнда Op . Содержимое Op уменьшается на 1. (Op) - 1→ (Op)
DAA		Коррекция результата сложения для представления в десятичном виде. AL
DAS		Коррекция результата вычитания для представления в десятичном виде. AL

Примечание: Операции с многобайтными числами производятся по байтам, начиная с младших. При сложении (вычитание) многобайтных чисел младшие байтных чисел младшие байты складываются (вычитаются) командой ADD (SUB) все последующие старшие байты командой ADC (SBB R).

МП может работать с двоичными и с двоично-десятичными числами. Но т.к. двоично-десятичные числа складываются (вычитаются) на двоичном сумматоре, то требуется коррекция результата, для этого используется команда DAA(DAS) - десятичная коррекция.

Часть 2

Мнемокод	Операнд	Комментарий
MUL	Op	Целочисленное умножение без учета знака (AL)*(Op8) → (AX) (AX)*(Op16) → (DX):(AX)
DIV	Op	**Беззнаковое деление** (AX)/(Op8) → частное a (AL), остаток в (AH) (DX)(AX)/(Op16) → частное a (AX), остаток в (DX)
AAM		Коррекция после умножения двух неупакованных BCD чисел. Преобразование двоичного числа меньшего 63h (99_{10}) в его неупакованный BCD-эквивалент.
AAD		Коррекция перед делением двух неупакованных BCD чисел. Преобразование двузначного неупакованного BCD-числа меньшего 63h (99_{10}) в двоичное представление.

Команда MUL выполняет умножение двух операндов без учета знаков. Алгоритм зависит от формата операнда команды и требует явного указания местоположения только одного сомножителя, который может быть расположен в памяти или в регистре. Местоположение второго сомножителя фиксировано и зависит от размера первого сомножителя.

Для команды DIV необходимо задание двух операндов — делимого и делителя. Делимое задается неявно и размер его зависит от размера делителя, который указывается в команде. При выполнении операции деления возможно возникновение исключительной ситуации: 0 — ошибка деления. Эта ситуация возникает в одном из двух случаев: делитель равен 0 или частное слишком велико для его размещения в регистре ax/al.

ХОД РАБОТЫ

Вариант 1

Вычислить выражение X + Y - Z, где X = 2860; Y = 88; Z =56, X,Y,Z - десятичные числа. Результат разместить в ячейках памяти, начиная с адреса ds:0005H.

Вариант 2

Найти сумму двух трехбайтных чисел A5C865H и 74D6CAH. Результат разместить в ячейках памяти, начиная с адреса ds:0010H.

Вариант 3

Найти сумму двух десятичных чисел 2357 и 9599. Результат разместить в ячейках памяти, начиная с адреса ds:0013H.

Вариант 4

Вычислить выражение X + Y - Z, где X = 4792; Y = 6888; Z =1014; X,Y,Z - десятичные числа. Результат разместить в ячейках памяти, начиная с адреса ds:0011H.

Вариант 5

Найти разность двух трехбайтных чисел D48B15H и 81A7E4H. Результат разместить в ячейках памяти, начиная с адреса ds:0003H.

Вариант 6

Найти разность двух десятичных чисел 18723 и 9569. Результат разместить в ячейках памяти, начиная с адреса ds:0004H.

Вариант 7

Найти сумму двух десятичных чисел 9092 и 2624. Результат разместить в ячейках памяти, начиная с адреса ds:0009H.

Вариант 8

Вычислить выражение X + Y - Z, где X = 6453; Y =2369; Z =38; X,Y,Z - десятичные числа, Результат разместить в ячейках памяти, начиная с адреса ds:0008H.

Вариант 9

Найти разность двух шестнадцатиричных чисел 732112H и 354634H. Результат разместить в ячейках памяти, начиная с адреса ds:0007H.

Вариант 10

Найти разность двух десятичных чисел 576623 и 8769. Результат разместить в ячейках памяти, начиная с адреса ds:0003H

Вариант 11

Найти разность двух шестнадцатиричных чисел D4568BH и 4857E4H. Результат разместить в ячейках памяти, начиная с адреса ds:0015H.

Вариант 12

Найти сумму двух шестнадцатиричных чисел F546D4H и 72D781H. Результат разместить в ячейках памяти, начиная с адреса ds:0010H.

Вариант 13

Найти сумму двух десятичных чисел 549092 и 958261. Результат разместить в ячейках памяти, начиная с адреса ds:0013H.

Вариант 14

Найти разность двух десятичных чисел 651521 и 18145. Результат разместить в ячейках памяти, начиная с адреса ds:0023H

Вариант 15

Вычислить выражение X + Y - Z, где X = 86453; Y = 12369; Z =238; X,Y,Z - десятичные числа. Результат разместить в ячейках памяти, начиная с адреса ds:0008H.

Список литературы

Основные источники:
1.Гуров, В. В. Микропроцессорные системы : учебник / В. В. Гуров. - Москва : ИНФРА-М, 2020.— 336 с. - (Среднее профессиональное образование). - ISBN 978-5-16-107848-8. – URL : https://new.znanium.com/catalog/document?id=357994 (дата обращения: 04.02.2020). - Текст : электронный.
2.Информационные системы управления качеством в автоматизированных и автоматических производствах : учебное пособие / А. Л. Галиновский, С.В. Бочкарев, И.Н. Кравченко [и др.] ; под ред. А.Л. Галиновского. — Москва : ИНФРА-М, 2020. — 284 с. — (Среднее профессиональное образование). - ISBN . - URL : https://new.znanium.com/catalog/document?id=353085 (дата обращения: 04.02.2020). Текст : электронный.
3.Кушнер, Д.А. Основы автоматики и микропроцессорной техники : учебное пособие / Д.А. Кушнер, А. В. Дробов, Ю.Л. Петроченко. - Минск : РИПО, 2019. - 245 с. - ISBN 978-985-503-853-6. – URL : https://new.znanium.com/catalog/document?id=347010 (дата обращения: 04.02.2020). - Текст : электронный.

Дополнительные источники :
1.Гальперин, М. В. Автоматическое управление : учебник / М. В. Гальперин. — Москва : ИД «ФОРУМ» : ИНФРА-М, 2019. - 224 с. — (Среднее профессиональное образование). - ISBN 978-5-16-103363-0. - URL : https://new.znanium.com/catalog/document?id=338850 (дата обращения: 04.02.2020). - Текст : электронный.