ВВОДНЫЕ УКАЗАНИЯ

Пособие для выполнения лабораторных работ по дисциплине «Микропроцессорные устройства РЭО» содержит описания трех лабораторных работ, выполняемых с применением компьютерных средств в интегрированной системе программирования «AVR Studio».

Этот цикл работ охватывает следующие разделы дисциплины:

1. Архитектура микропроцессоров.
2. Основы программирования микропроцессоров.
3. Типовые программные процедуры.
4. Алгоритмы цифровой обработки сигналов.
5. Применение микропроцессоров для цифровой обработки сигналов.

В цикле работ данного практикума студенты знакомятся с архитектурой 8-разрядного микроконтроллера AVR ATmega128, изучают систему его команд и методы адресации, осваивают интегрированную систему программирования, получают практические навыки программирования микропроцессорных систем на языке Ассемблер.

Перед выполнением каждой работы студенты опрашиваются преподавателем с целью определения степени подготовки их к работе, которая включает: знание содержания выполняемой работы, основные теоретические сведения по содержанию работы, порядок подготовки программы на Ассемблере для выполнения работы. При обнаружении недостаточной готовности студент не допускается к работе.

После получения допуска к выполнению работы студент получает задание на работы, подготавливает программу на Ассемблере, осуществляет ее компиляцию и пошаговое выполнение в программной среде «AVR Studio»,

При оформлении отчета следует руководствоваться следующим. Первая страница - титульный лист. Далее - содержание, включающее: цель работы, краткие теоретические сведения, задание на работу, текст программы с комментариями. Отчет оформляется на стандартных листах бумаги формата А4, расположенных вертикально. Допускается оформление на тетрадных двойных страницах (в клетку).

Студенты, не защитившие две предыдущие работы, к выполнению последующей не допускаются.

ЛАБОРАТОРНАЯ РАБОТА №1

МЕТОДЫ АДРЕСАЦИИ, КОМАНДЫ ПЕРЕДАЧИ ДАННЫХ И УПРАВЛЕНИЯ

1. ЦЕЛЬ РАБОТЫ

Знакомство с интегрированной средой программирования; изучение методов адресации, команд передачи данных и управления.

2. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

При создании программы для микроконтроллера на языке Ассемблер разработчик оперирует программно доступными ресурсами микропроцессорной системы. У микроконтроллера ATmega128 эти ресурсы включают: программно доступные регистры микроконтроллера, внутреннюю память данных, внешнюю память данных. Основные сведения о микроконтроллере ATmega128 приведены в приложении 1.

Каждая команда языка Ассемблер сообщает процессору выполняемую операцию и методы доступа к операндам. Командная строка Ассемблера включает метку (символический адрес), мнемонику (символическое имя) команды, поле операндов, комментарий. Имя команды однозначно связано с выполняемой ею операцией.

Методы адресации представляют собой набор механизмов доступа к операндам. Одни из них просты и поэтому приводят к компактному формату команды и быстрому доступу к операнду, но объем доступных с их помощью ресурсов ограничен. Другие методы адресации позволяют оперировать со всеми имеющимися в системе ресурсами, но команда получается длинной, на ее ввод и выполнение тратится много времени. Набор методов адресации в каждой системе команд является компромиссным сочетанием известных механизмов адресации, выбранных проектировщиками архитектуры исходя из набора решаемых задач.

Заметим, что при двух операндах и приемнике результата имеет место трехадресная команда и каждый адрес формируется с использованием собственного метода адресации. Если адрес операнда или приемника результата не указан в команде, а подразумевается, то имеет место неявная адресация.

Команды передачи данных и команды переходов приведены в табл. 1.

Табл. 1. Команды пересылки данных и переходов

Мнемо-ника	Операн-ды	Описание	Операция	Флаги	Кол-во циклов
1	2	3	4	5	6
ELPM		Расширенная загрузка из памяти программ в регистр R0	R0 ← (Z+RAMPZ)	Нет	3
MOV	Rd,Rr 0≤d≤31 0≤r≤31	Копировать регистр	Rd ← Rr	Нет	1
LDI	Rd,K 16≤d≤31 0≤k≤255	Загрузить непосредственное значение	Rd ← K	Нет	1
LDS	Rd,k 0≤d≤31 0≤k≤65535	Загрузить из ОЗУ	Rd ← (k)	Нет	3
LD	Rd,X 0≤d≤31	Загрузить косвенно	Rd ← (X)	Нет	2
LD	Rd,X+ 0≤d≤31	Загрузить косвенно с постинкрементом	Rd ← (X), X ← X+1	Нет	2
LD	Rd,-X 0≤d≤31	Загрузить косвенно с преддекрементом	X ← X - 1, Rd ← (X)	Нет	2
LD	Rd,Y 0≤d≤31	Загрузить косвенно	Rd ← (Y),	Нет	2
LD	Rd,Y+ 0≤d≤31	Загрузить косвенно с постинкрементом	Rd ← (Y), Y ← Y+1	Нет	2
LD	Rd,-Y 0≤d≤31	Загрузить косвенно с преддекрементом	Y ← Y - 1, Rd ← (Y)	Нет	2
LDD	Rd,Y+q 0≤d≤31 0≤q≤63	Загрузить косвенно со смещением	Rd ← (Y+q)	Нет	2
LD	Rd,Z 0≤d≤31	Загрузить косвенно	Rd ← (Z)	Нет	2
LD	Rd,Z+ 0≤d≤31	Загрузить косвенно с постинкрементом	Rd ← (Z), Z ← Z+1	Нет	2
LD	Rd,-Z 0≤d≤31	Загрузить косвенно с преддекрементом	Z ← Z - 1, Rd ← (Z)	Нет	2

Продолжение табл. 1

1	2	3	4	5	6
LDD	Rd,Z+q 0≤d≤31 0≤q≤31	Загрузить косвенно со смещением	Rd ← (Z+q)	Нет	2
STS	k,Rr 0≤r≤31 0≤k≤65535	Загрузить непосредственно в ОЗУ	(k) ← Rr	Нет	3
ST	X,Rr 0≤r≤31	Записать косвенно	(X) ← Rr	Нет	2
ST	X+,Rr 0≤r≤31	Записать косвенно с постинкрементом	(X) ← Rr, X ← X + 1	Нет	2
ST	-X,Rr 0≤r≤31	Записать косвенно с преддекрементом	X ← X - 1, (X) ← Rr	Нет	2
ST	Y,Rr 0≤r≤31	Записать косвенно	(Y) ← Rr	Нет	2
ST	Y+,Rr 0≤r≤31	Записать косвенно с постинкрементом	(Y) ← Rr, Y ← Y + 1	Нет	2
ST	-Y,Rr 0≤r≤31	Записать косвенно с преддекрементом	Y ← Y - 1, (Y) ← Rr	Нет	2
STD	Y+q,Rr 0≤r≤31 0≤q≤63	Записать косвенно со смещением	(Y+q) ← Rr	Нет	2
ST	Z,Rr 0≤r≤31	Записать косвенно	(Z) ← Rr	Нет	2
ST	Z+,Rr 0≤r≤31	Записать косвенно с постинкрементом	(Z) ← Rr, Z ← Z + 1	Нет	2
ST	-Z,Rr 0≤r≤31	Записать косвенно с преддекрементом	Z ← Z - 1, (Z) ← Rr	Нет	2
STD	Z+q,Rr 0≤r≤31 0≤q≤63	Записать косвенно со смещением	(Z+q) ← Rr	Нет	2
LPM		Загрузить байт из памяти программ	R0 ← (Z)	Нет	3
IN	Rd,P 0≤d≤31 0≤P≤63	Загрузить данные из порта I/O в регистр	Rd ← P	Нет	1

Продолжение табл. 1

1	2	3	4	5	6
OUT	P,Rr 0≤r≤31 0≤P≤63	Записать данные из регистра в порт I/O	P ← Rr	Нет	1
PUSH	Rr 0≤r≤31	Сохранить регистр в стеке	STACK ← Rr	Нет	2
POP	Rd 0≤d≤31	Выгрузить регистр из стека	Rd ← STACK	Нет	2
RJMP	k - 2K≤k≤2K	Перейти относительно	PC ← PC + k + 1	Нет	2
LJMP		Перейти косвенно	PC ← Z	Нет	2
JMP	k 0≤k≤4M	Перейти	PC ← k	Нет	3
RCALL	K - 2K≤k≤2K	Вызвать подпрограмму относительно	PC ← PC + k + 1	Нет	3
ICALL		Вызвать подпрограмму косвенно	PC ← Z	Нет	3
CALL	K 0≤k≤64K	Выполнить длинный вызов подпрограммы	PC ← k	Нет	4
RET		Вернуться из подпрограммы	PC ← STACK	Нет	4
RETI		Вернуться из прерывания	PC ← STACK	1	4
CPSE	Rd,Rr 0≤d≤31, 0≤r≤31	Сравнить и пропустить, если равно	if Rd=Rr then PC ← PC + 2 (or 3)	Нет	1/2/3

Команды ELPM и LPM выполняют загрузку памяти программ. Команда MOV копирует содержимое одного регистра общего назначения в другой. Команда LDI загружает регистр общего назначения непосредственно константой (только R16 – R32), а команды LD выполняют загрузку регистра из ячейки ОЗУ при косвенной адресации, используя в качестве источника адреса регистры X, Y, Z, а также варианты с постинкрементом и преддекрементом источника. Команда LDD выполняет эту же операцию при помощи косвенной

адресации со смещением. Команда LDS загружает в регистр общего назначения содержимое прямо адресуемой ячейки памяти данных.

Команда ST выполняют обратную операцию относительно LD – заносит в косвенно адресуемую ячейку памяти данных содержимое регистра общего назначения, используя также варианты с постинкрементом и преддекрементом адреса. Команда STD выполняет эту же операцию при помощи косвенной адресации со смещением, а команда STS прямо адресует ячейку ОЗУ.

Команда IN заносит содержимое регистра ввода-вывода в регистр общего назначения, а команда OUT выполняет обратную операцию.

Команда PUSH сохраняет содержимое регистра в стеке, а команда POP выполняет обратную операцию. При выполнении этих команд кроме программного счетчика изменяется и значение указателя стека SP.

3. МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ РАБОТЫ

3.1. Получить допуск к выполнению лабораторной работы и вариант задания пересылки данных в микроконтроллере ATmega 128 от преподавателя.

3.2. Подготовить программу на Ассемблере, соответствующую полученному заданию.

3.3. Запустить на компьютере систему «AVR Studio» (см. приложение 2), создать новый проект (имя проекта – фамилия студента) и ввести текст программы.

3.4. Осуществить компиляцию проекта с помощью команды Project/Build.

3.5. С помощью команды Debug/Start Debugging запустить симулятор.

3.6. Выполнить программу по шагам, выполняя команду Debug/Step Intro(F11). После выполнения текущей команды курсор в окне редактора текста указывает на следующую команду.

3.7. Проверить правильность пересылки данных и оформить отчет.

4. СОДЕРЖАНИЕ ОТЧЕТА

Отчет по лабораторной работе должен содержать:

- титульный лист;
- цель работы;
- текст задания;
- текст программы с подробными комментариями;
- выводы по работе.

5. РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. Горбунов А.Л. Микропроцессорные устройства РЭС: Конспект лекций. – М.:МГТУ ГА, 1999.

2. Калабеков Б.А. Микропроцессоры и их применение в системах передачи и обработки сигналов. – М.:Радио и связь, 1988.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

6.1. Для чего используется адресация в микропроцессорах?

6.2. Назовите основные виды адресации.

6.3. Что такое условные переходы?

6.4. Что такое безусловные переходы?

6.5. Какие регистры в микроконтроллере ATmega 128 используются для адресации данных?

6.6. Что такое подпрограмма?

6.7. Где сохраняется адрес команды к которой надо перейти после выполнения подпрограммы?

6.8. Сколько раз может осуществляться вызов одной и той же подрограммы?

ЛАБОРАТОРНАЯ РАБОТА №2

КОМАНДЫ ОБРАБОТКИ ДАННЫХ

1. ЦЕЛЬ РАБОТЫ

Изучение команд арифметических и логических операций, практическое освоение приемов программирования на ассемблере.

2. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Для обработки данных микроконтроллер ATmega128 использует группу команд, реализующих арифметические и логические операции, сдвиги и операции над отдельными битами. Арифметические операции являются операциями над 8-разрядными целыми числами. Для выполнения целочисленных операций над длинными словами служат команды сложения и вычитания с учетом флага переноса

Арифметические и логические команды приведены в табл. 2.

Табл. 2. Основные арифметические и логические операции

Мнемо-ника	Операн-ды	Описание	Операция	Флаги	Кол-во циклов
ADD	Rd,Rr 0≤d≤31 0≤r≤31	Сложить без переноса	Rd ← Rd + Rr	Z, C, N, V, H	1
ADC	Rd,Rr 0≤d≤31 0≤r≤31	Сложить с переносом	Rd ← Rd + Rr + C	Z, C, N, V, H	1
SUB	Rd,Rr 0≤d≤31 0≤r≤31	Вычесть без заема	Rd ← Rd - Rr	Z, C, N, V, H	1
SUBI	Rd, K 16≤d≤31 0≤K≤255	Вычесть непосредственное значение	Rd ← Rd - K	Z, C, N, V, H	1
SBC	Rd, Rr 0≤d≤31 0≤r≤31	Вычесть с заемом	Rd ← Rd - Rr - C	Z, C, N, V, H	1
SBCI	Rd, K 16≤d≤31 0≤K≤255	Вычесть непосредственное значение с заемом	Rd ← Rd - K - C	Z, C, N, V, H	1
AND	Rd, Rr 0≤d≤31 0≤r≤31	Выполнить логическое AND	Rd ← Rd • Rr	Z, N, V	1
OR	Rd, Rr 0≤d≤31 0≤r≤31	Выполнить логическое OR	Rd ← Rd ∨ Rr	Z, N, V	1
EOR	Rd, Rr 0≤d≤31 0≤r≤31	Выполнить исключающее OR	Rd ← Rd ⊕ Rr	Z, N, V	1
INC	Rd 0≤d≤31	Инкрементировать	Rd ← Rd + 1	Z, N, V	1
DEC	Rd 0≤d≤31	Декрементировать	Rd ← Rd - 1	Z, N, V	1

3. МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ РАБОТЫ

3.1. Получить допуск к выполнению лабораторной работы и один из приведенных ниже вариантов задания.

ВАРИАНТ 1

Сложить четыре числа 17, 25, 93, 77 (без учета единица переноса). Для записи чисел использовать РОНЫ R20…R23, результат записать в ячейку ОЗУ 0040.

Загрузить ячейки ОЗУ 0080 и 0081 числами 4A и 18 соответственно, используя косвенную адресацию (регистровые пары X и Y соответственно).

Вычесть (без заема) из содержимого ячейки ОЗУ 0040 число 10. Результат записать в ячейку ОЗУ 0041, используя косвенную адресацию (регистровая пара Z).

Записать в регистр R16 число 50.

Поменять местами содержимое ячеек ОЗУ 0040 и 0041.

Осуществить вызов подпрограммы ROOT, после чего «зациклить» программу.

Написать подпрограмму ROOT в которой:

Сохранить указатели X, Y, Z и содержимое регистра R16 а стеке. Начальный адрес стека – 10FF.

Записать в регистр R16 содержимое ячейки ОЗУ 0081 и сравнить с содержимым ячейки ОЗУ 0040, в случае неравенства увеличить содержимое регистра R16 на единицу.

Извлечь сохраненные данные из стека, поменяв содержимое указательных регистров X и Y местами.

ВАРИАНТ 2

Сложить четыре числа 27, 15, 13, 87 (без учета единица переноса). Для записи чисел использовать РОНЫ R17…R20, результат записать в ячейку ОЗУ 0040.

Загрузить ячейки ОЗУ 0080 и 0081 числами 45 и 15 соответственно, используя косвенную адресацию (регистровые пары X и Y соответственно).

Вычесть (без заема) из содержимого ячейки ОЗУ 0040 число 80. Результат записать в ячейку ОЗУ 0041, используя косвенную адресацию (регистровая пара Z).

Записать в регистр R16 число 40.

Поменять местами содержимое ячеек ОЗУ 0040 и 0041.

Осуществить вызов подпрограммы ROOT, после чего «зациклить» программу.

Написать подпрограмму ROOT в которой:

Сохранить указатели X, Y, Z и содержимое регистра R16 а стеке. Начальный адрес стека – 10FF.

Записать в регистр R16 содержимое ячейки ОЗУ 0081 и сравнить с содержимым ячейки ОЗУ 0040, в случае неравенства уменьшить содержимое регистра R16 на единицу.

Извлечь сохраненные данные из стека, поменяв содержимое указательных регистров Z и Y местами.

ВАРИАНТ 3

Сложить четыре числа 37, 25, 17, 97 (без учета единица переноса). Для записи чисел использовать РОНЫ R19…R22, результат записать в ячейку ОЗУ 0040.

Загрузить ячейки ОЗУ 0080 и 0081 числами 49 и 33 соответственно, используя косвенную адресацию (регистровые пары X и Y соответственно).

Вычесть (без заема) из содержимого ячейки ОЗУ 0040 число 30. Результат записать в ячейку ОЗУ 0041, используя косвенную адресацию (регистровая пара Z).

Записать в регистр R16 число 70.

Поменять местами содержимое ячеек ОЗУ 0040 и 0041.

Осуществить вызов подпрограммы ROOT, после чего «зациклить» программу.

Написать подпрограмму ROOT в которой:

Сохранить указатели X, Y, Z и содержимое регистра R16 а стеке. Начальный адрес стека – 10FF.

Записать в регистр R16 содержимое ячейки ОЗУ 0081 и сравнить с содержимым ячейки ОЗУ 0040, в случае неравенства уменьшить содержимое регистра R16 на единицу.

Извлечь сохраненные данные из стека, поменяв содержимое указательных регистров Z и X местами.

3.2. Подготовить программу на Ассемблере, соответствующую полученному варианту задания.

3.3. Запустить на компьютере систему «AVR Studio» (см. приложение 2), создать новый проект (имя проекта – фамилия студента) и ввести текст программы.

3.4. Осуществить компиляцию проекта с помощью команды Project/Build.

3.5. В случае успешной компиляции с помощью команды Debug/Start Debugging запустить симулятор.

3.6. Выполнить программу по шагам, выполняя команду Debug/Step Intro(F11). После выполнения текущей команды курсор в окне редактора текста указывает на следующую команду.

3.7. Проверить правильность выполнения команд и оформить отчет.

4. СОДЕРЖАНИЕ ОТЧЕТА

Отчет по лабораторной работе должен содержать:
- титульный лист;
- цель работы;
- текст задания;
- текст программы с подробными комментариями;
- выводы по работе.

5. РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. Горбунов А.Л. Микропроцессорные устройства РЭС: Конспект лекций. – М.:МГТУ ГА, 1999.

2. Калабеков Б.А. Микропроцессоры и их применение в системах передачи и обработки сигналов. – М.:Радио и связь, 1988.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

6.1. Перечислите основные логические и арифметические операции, осуществляемые в микроконтроллере.

6.2. Что такое стек?

6.3. Назначение указателя стека.

6.4. Как осуществляется запись данных в стек.

6.5. Как осуществляется извлечение данных из стека.

6.6. Формат регистра флагов.

6.7. Что такое регистры общего назначения?

ЛАБОРАТОРНАЯ РАБОТА №3

РЕАЛИЗАЦИЯ АЛГОРИТМОВ ЦИФРОВОЙ ОБРАБОТКИ СИГНАЛОВ НА МИКРОКОНТРОЛЛЕРЕ ATMEGA 128

1. ЦЕЛЬ РАБОТЫ

Получение практических навыков применения микроконтроллера для решения задач цифровой обработки сигналов (ЦОС).

2. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

ЦОС охватывает широкий спектр применений. К ним относятся цифровая фильтрация, кодирование и декодирование информации, распознавание речи, обработка изображений, спектральный анализ, управление системами. В РЭС типовыми задачами ЦОС являются реализации различных видов ЦФ, перенос спектра сигнала из одной частотной области в другую, вычисление энергетического спектра, корреляционных функций. В большинстве случаев эти задачи решаются с применением алгоритма быстрого преобразования Фурье (БПФ).

БПФ - специальная версия дискретного Фурье - преобразования (ДПФ), позволяющего представить сигналы в виде суперпозиции гармонических функций. ДПФ, описывается парой соотношений (прямым и обратным):

$$G(f_k) = \frac{1}{N}\sum_{n=0}^{N-1} X(t_n)\exp\left(-\frac{j2\pi kn}{N}\right); \quad (1)$$

$$X(t_n) = \frac{1}{N}\sum_{k=0}^{N-1} G(f_k)\exp\left(\frac{j2\pi kn}{N}\right), \quad (2)$$

где $t_n = n\Delta t$, $f_k = k\Delta f = k/T$ выполняет блочный анализ временного сигнала.

Блок состоит из N отсчётов входного процесса, что соответствует длине реализации T. При этом в ДПФ подразумевается, что анализируемый блок является одним из периодов бесконечной периодической функции. Соответственно, получаемый частотный спектр (т.е. зависимость амплитуд и фаз гармонических составляющих сигнала от частоты) будет линейчатым, т.е. состоящим из отдельно

расположенных вдоль частотной оси с интервалом линий, определяющих различные гармонические компоненты временного сигнала.

Если временная функция содержит только действительные значения, что является случаем, часто встречающимся на практике, то спектр сигнала является сопряженной линейчатой функцией, т.е. компоненты положительной и отрицательной областей частот имеют одинаковые амплитуды, но противоположные фазы. Иначе говоря, только N/2 отсчетов в частотной области являются независимыми, а именно те, что расположены на отрезке $(0, f_k)$.

В общем случае при вычислении ДПФ от комплексной входной последовательности необходимо производить вычисление всех N спектральных компонент, так как в данном случае спектр не обладает свойством комплексно-сопряженной симметрии.

Еще одна особенность ДПФ связана с представлением анализируемого сигнала как одного из интервалов бесконечной периодической функции. Если начальное и конечное значения сигнала на периоде различны, то в моменты времени 0,T,2T,... происходит так называемый скачок функции, который может привести к существенному искажению спектра. Для устранения негативного влияния на спектр данной разрывности и получения лучшей избирательности необходимо использовать сглаживание с помощью одного из видов весовой функции (окна).

Возможность использования ДПФ при вычислении цифровых сверток вытекает из фундаментального свойства преобразования Фурье, гласящего, что Фурье-образ произведения двух временных сигналов эквивалент свертке преобразований от каждого из сомножителей и, наоборот, свертка сигналов во временной области соответствует умножению их Фурье-отображений и в частотной области. Поэтому возможно вычисление циклической (периодической) свертки косвенным путем - посредством выполнения обратного преобразования Фурье от произведения ДПФ исходных периодических временных последовательностей. Такое решение имеет существенное практическое значение в том случае, когда преобразования могут быть выполнены быстрее, чем сама свертка.

Эффективными процедурами вычисления ДПФ являются алгоритмы быстрого преобразования Фурье (БПФ), начало разработки которых было положено исследованиями Кули и Тьюки .Общий принцип построения алгоритмов БПФ заключается в разложении процесса нахождения ДПФ на вычисление преобразований последовательностей меньшего размера. Действительно, как видно из выражения (1), суть ДПФ заключается в получении суммы произведений вида $X(k)W^{-nk}$, где $W^{-nk}=\exp(-j2\pi/N)$.

Весовая функция является периодической с интервалом N. Для заданного k при изменении n от нуля до N-1 произведение $X(k)W^{-nk}$ также меняется периодически, причем число этих периодов равно k. Более того, даже в пределах одного периода могут образовываться комплексно-сопряженные пары. Используя данные свойства, можно существенно уменьшить необходимое для вычислений число умножений. Для этого входная последовательность $X(t_n)$ разбивается на две вспомогательные последовательности - $Y(t_n)$ и $Z(t_n)$ - так, что в $Y(t_n)$ попадают отсчеты только с четными номерами, а в $Z(t_n)$ - с нечетными, т.е. $Y(t_n)= X(t_{2n})$, $Z(t_n)= X(t_{2n+1})$.

Тогда выражение (1) записывается в виде:

$$G(f_k)=0,5*(Y(f_k) + Z(f_k)*W^k). \quad (3)$$

Вычислительные затраты на реализацию N - точечного ДПФ в соответствии с выражением (3) составляют по (N/2) - точечных ДПФ $Y(f_k)$ и $Z(f_k)$ и дополнительно N операций на их соединение, тогда как прямое вычисление $G(f_k)$ по (1) требует N^2 операций, что почти в два раза больше.

В выражении (4) индекс k изменяется от 0 до N-1. Однако $Y(f_k)$ и $Z(f_k)$ имеют период N/2 и вычисляются только в диапазоне от 0 до N/2-1. Используя свойство периодичности функции W^k можно найти искомое соотношение и для N/2 k N-1. Окончательно получаем :

$$G(f_k)=0,5(Y(f_k) + Z(f_k)W^k), \text{ при } 0<k<N/2-1; \quad (4)$$

$$G(f_k)=0,5(Y(f_{k-N/2}) - Z(f_{k-N/2})W^k), \text{ при } n/2<k<N-1. \quad (5)$$

Каждое из двух N/2-точечных преобразований (если число их членов четно) может быть найдено повторением указанного приема, т.е. последовательности $Y(t_n)$ и $Z(t_n)$ снова разбиваются на две последовательности длиной N/4, причем для каждой их пары подбираются соответствующие весовые коэффициенты. Если является степенью числа 2, то последовательное прореживание входных значений может быть продолжено до получения двухточечных преобразований.

Данный алгоритм получил название БПФ с прореживанием по времени. Базовая операция его, называемая "бабочкой" (баттерфляй), состоит в получении выходных чисел P и Q из входных A и B в соответствии с выражением

$$P = A+ W^K B; Q = A- W^K B. \quad (6)$$

Очевидно, что значения, получаемые в результате выполнения базовой операции, могут быть хранимы в запоминающемся устройстве на месте исходных операндов, что существенно сокращает требуемый объем запоминающих устройств. Алгоритм БПФ, в котором реализуется такая переадресация данных, получил название алгоритма с замещением.

Второй распространенной формой алгоритма БПФ является метод прореживания по частоте, базовая операция БПФ с прореживанием при этом имеет вид:

$$P = A + B;\ Q = (A - B)\,W^{K}. \qquad (7)$$

3. МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ РАБОТЫ

3.1. Получить допуск к выполнению лабораторной работы.

3.2. На основании формул (6) и (7) получить выражения, связывающие действительные и мнимые части выходных операндов (P и Q) с действительными и мнимыми частями входных операндов (A, B, W).

3.3. Подготовить программу на Ассемблере реализации базовой операции БПФ, соответствующую полученному заданию.

3.4. Запустить на компьютере систему «AVR Studio» (см. приложение 2), создать новый проект (имя проекта – фамилия студента) и ввести текст программы.

3.5. Осуществить компиляцию проекта с помощью команды Project/Build.

3.6. С помощью команды Debug/Start Debugging запустить симулятор.

3.7. Выполнить программу по шагам, выполняя команду Debug/Step Intro(F11). После выполнения текущей команды курсор в окне редактора текста указывает на следующую команду.

3.8. Проверить правильность выполнения команд и оформить отчет.

4. СОДЕРЖАНИЕ ОТЧЕТА

Отчет по лабораторной работе должен содержать:

- титульный лист;
- цель работы;
- текст задания;
- текст программы с подробными комментариями;
- выводы по работе.

5. РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. Горбунов А.Л. Микропроцессорные устройства РЭС: Конспект лекций. – М.:МГТУ ГА, 1999.

2. Калабеков Б.А. Микропроцессоры и их применение в системах передачи и обработки сигналов. – М.:Радио и связь, 1988.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

6.1. Для чего применяется ДПФ?
6.2. В чем суть алгоритмов БПФ?
6.3. Запишите базовую операцию БПФ с прореживанием по времени.
6.4. Запишите базовую операцию БПФ с прореживанием по времени.
6.5. Перечислите основные алгоритмы ЦОС.

Приложение 1

ОБЩИЕ СВЕДЕНИЯ О МИКРОКОНТРОЛЛЕРЕ ATMEGA 128

ATmega128 – маломощный 8-разрядный КМОП микроконтроллер, основанный на расширенной AVR RISC-архитектуре. За счет выполнения большинства инструкций за один машинный цикл ATmega128 достигает производительности 1 млн. операций в секунду/МГц, что позволяет проектировщикам систем оптимизировать соотношение энергопотребления и быстродействия.

Структура микроконтроллера ATmega128 включает следующие функциональные блоки:

- 8-разрядное арифметическо-логическое устройство (АЛУ);
- внутреннюю flash-память программ объемом 128 Кбайт с возможностью внутрисистемного программирования через последовательный интерфейс;
- 32 регистра общего назначения;
- внутреннюю EEPROM память данных объемом 4 Кбайт;
- внутреннее ОЗУ данных объемом 4 Кбайт;
- 6 параллельных 8-разрядных портов;
- 4 программируемых таймера-счетчика;
- 10-разрядный 8-канальный АЦП и аналоговый компаратор;
- последовательные интерфейсы UART0, UART0, TWI и SPI;
- блоки прерывания и управления (включая сторожевой таймер).

ATmega128 поддерживается полным набором программных и аппаратных средств для проектирования, в т.ч.: Си-компиляторы, макроассемблеры, программные отладчики/симуляторы, внутрисистемные эмуляторы и оценочные наборы.

Высокие характеристики семейства AVR обеспечиваются следующими особенностями архитектуры:

1. В качестве памяти программ используется внутренняя flash-память. Она организована в виде матрицы 16-разрядных ячеек и может загружаться программатором, либо через порт SPI;

2. Система команд включает 133 инструкций;

3. 16-разрядные память программ и шина команд вместе с одноуровневым конвейером позволяют выполнить большинство инструкций за один такт синхрогенератора (50 нс при частоте FOSC=20 МГц);

4. Память данных имеет 8-разрядную организацию. Младшие 32 адреса пространства занимают регистры общего назначения, далее следуют 64 адреса регистров ввода-вывода, затем внутреннее ОЗУ данных объемом до 4096 ячеек. Возможно применение внешнего ОЗУ данных объемом до 60 Кбайт;

5. Внутренняя энергонезависимая память типа EEPROM объемом до 4 Кбайт представляет собой самостоятельную матрицу, обращение к которой осуществляется через специальные регистры ввода-вывода.

Регистровый файл с быстрым доступом содержит 32 регистра общего назначения (РОН), которые включены в сквозное адресное пространство ОЗУ данных и занимают младшие адреса (рис. 1). РОНы предназначены для хранения адресов и данных. Файл регистров общего назначения прямо связан с арифметико-логическим устройством (АЛУ), каждый из регистров способен работать как аккумулятор. Шесть регистров из 32 могут использоваться как три 16-разррядных регистра косвенного адреса для эффективной адресации в пределах памяти данных. Данные 16-разрядные регистры называются X-регистр, Y-регистр и Z-регистр.

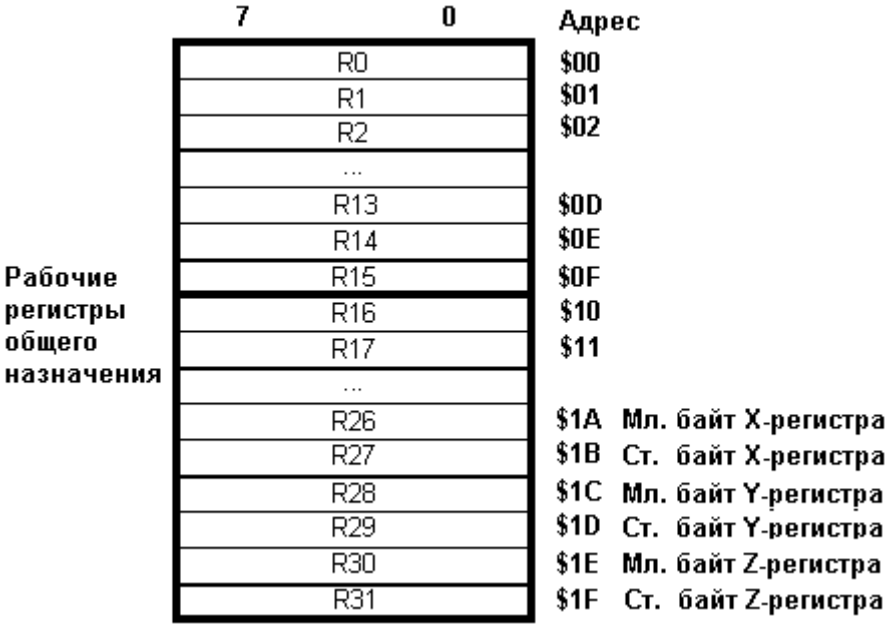

Рис. 1. Регистры общего назначения микроконтроллера ATmega128

Регистр состояния SREG 8-разрядный, в каждом разряде которого устанавливаются следующие флаги:

Bit7 – Разрешение всех прерываний. Для разрешения прерываний этот бит должен быть установлен (=1). Если этот бит очищен (=0), то ни одно из прерываний не обрабатывается. Бит аппаратно очищается после возникновения прерывания и устанавливается (разрешая последующие прерывания) командой RETI.

Bit 6 – Бит сохранения копии. Команды копирования бита BLD и BST используют этот бит как источник и приемник при операциях с битами.

Командой BST бит регистра общего назначения копируется в бит T, командой BLD бит T копируется в бит регистра общего назначения.

Bit 5 – Флаг полупереноса. Флаг полупереноса указывает на перенос между тетрадами при выполнении ряда арифметических операций. Более подробная информация приведена в описании системы команд.

Bit 4 – Бит знака. Бит имеет значение результата операции исключающее ИЛИ над флагами отрицательного значения и дополнения до двух флага переполнения.

Bit 3 – Дополнение до двух флага переполнения. Этот бит поддерживает арифметику дополнения до двух.

Bit 2 – Флаг отрицательного значения. Этот флаг указывает на отрицательный результат ряда арифметических и логических операций.

Bit 1 – Флаг нулевого значения. Этот флаг указывает на нулевой результат ряда арифметических и логических операций.

Bit 0 – Флаг переноса. Этот флаг указывает на перенос при арифметических и логических операциях.

Микроконтроллеры AVR оснащены 16-разрядным указателем стека SP, размещенным в двух регистрах ввода-вывода (рис. 2) – для хранения старших разрядов (регистр SPH) и младших (регистр SPL). Поскольку микроконтроллеры ATmega128 поддерживают объем ОЗУ до 64 Кбайт, то используются все 16 разрядов указателя стека.

Указатель стека указывает на область в ОЗУ данных, в которой размещается стек подпрограмм и прерываний, который обычно используется для хранения временных данных, для хранения локальных переменных и для хранения адресов возврата при прерываниях и вызовах подпрограмм. Регистр указателя стека указывает на вершину стека. Начальный адрес указателя должен задаваться программно перед вызовом подпрограмм и разрешением прерываний. Начальное значение должно быть больше $60. Указатель стека декрементируется (уменьшается) на единицу при каждом занесении командой PUSH данных в стек, и на две единицы при занесении в стек адреса при вызове подпрограммы или процедуры прерывания.

Указатель стека инкрементируется (увеличивается) на единицу при извлечении данных из стека командой POP, и на две единицы при извлечении адреса из стека при возврате из подпрограммы (RET) или возврате из процедуры прерывания (RETI).

Приложение 2

ИНТЕГРИРОВАННАЯ СИСТЕМА ПРОГРАММИРОВАНИЯ AVR Studio V4.12

Интегрированная система программирования включает редактор текста, макроассемблер, редактор связей и символический отладчик. Система позволяет разрабатывать целевую программу и отлаживать ее в реальном масштабе времени с использованием аппаратных ресурсов платы контроллера. После создания рабочей программы разработчик имеет возможность, загрузить программу в память микроконтроллера. Отладка программ производится в исходном тексте, причем на каждом шаге можно наблюдать за изменениями внутренних ресурсов микроконтроллера и модифицировать их.

При программировании в среде AVR Studio надо выполнить стандартную последовательность действий:

- создание проекта;
- создание файла программы;
- компиляция;
- симуляция;
- загрузка hex-кода в микроконтроллер.

СОЗДАНИЕ ПРОЕКТА

Создание проекта начинается с выбора строки меню Project/New Project. В открывшемся окне «Create new Project» надо указать имя проекта, (например LS2), имя файла программы ls2.asm появится автоматически. В строке Location выбираем папку для сохранения проектов.

После нажатия кнопки «Next» открывается окно «Select debug platform and device», где выбирается отладочная платформа (симулятор или эмулятор) и тип микроконтроллера.

Выбираем в качестве отладочной платформы AVR Simulator и микроконтроллер ATmega128. После нажатия кнопки “Finish” переходим в рабочее окно программы.

В правом окне вводим исходный текст программы. В начале программы ввести следующие строки:

```
.device ATmega128
.include "m128def.inc"
```

КОМПИЛЯЦИЯ ПРОЕКТА

Компиляция проекта производится командой Project/Build. Процесс компиляции отображается в окне Output в нижней части экрана.

Если программа набрана без ошибок, компилятор создаст файл для прошивки в микроконтроллер ls2.hex и файл для симулятора.

В противном случае будут показаны строки, в которых обнаружена ошибка. После их исправления процесс компиляции повторяем заново.

Мы получили файл для микроконтроллера, который можно запрограммировать и убедиться в работоспособности написанной программы. Но для учебных целей лучше запустить симулятор и провести пошаговую отладку программы.

СИМУЛЯЦИЯ ПРОЕКТА.

Переход в режим симуляции производится командой Debug/Start Debugging.

В левой части окна появляются внутренние модули микропроцессора (порты, счетчики, регистры и др.) и их значения, которые можно изменять.

В правом окне появляется желтая стрелка, указывающая на следующую исполняемую команду. По команде Debug/Step Intro(F11) выполняется команда, на которую указывает стрелка. Выполняя пошагово команды можно наблюдать в левом окне изменение содержимого задействованных регистров и портов и находить ошибки в программе.

ПРИМЕР ПРОГРАММЫ НА АССЕМБЛЕРЕ В СИСТЕМЕ AVR STUDIO

```
.device ATmega128
.include "m128def.inc"

ldi  R20, $57 ; загрузка регистра R20 константой
ldi  R30, $00 ; загрузка регистровой пары Z (R30,R31) адресом $0100
ldi  R31, $01 ; 0100
st   Z, R20 ; загрузка косвенно адресуемой ячейки ОЗУ с адресом $0100
            ; значением из регистра R20 ($57)
lds  R19, $0100 ; загрузка регистра R19 из ячейки ОЗУ с адресом $0100
sts  $0101, R19 ; загрузка ячейки с адресом $0101 из регистра R19
loop:
rjmp loop ; зацикливание программы
```

СОДЕРЖАНИЕ