ГОСУДАРСТВЕННОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Электропривод и автоматизация промышленных установок»

МИКРОПРОЦЕССОРНАЯ ТЕХНИКА В МЕХАТРОНИКЕ И РОБОТОТЕХНИКЕ

Методические рекомендации к лабораторным работам для студентов направления подготовки 15.03.06 «Мехатроника и робототехника» дневной формы обучения

Могилев 2018

УДК 004.3:62-83
ББК 32.973.26-04:31.291
М 76

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Электропривод и АПУ» «06» февраля 2018 г., протокол № 7

Составитель ст. преподаватель В. Н. Ситников

Рецензент канд. техн. наук, доц. С. В. Болотов

Методические рекомендации к лабораторным работам предназначены для студентов направления подготовки 15.03.06 «Мехатроника и робототехника» дневной формы обучения.

Учебно-методическое издание

МИКРОПРОЦЕССОРНАЯ ТЕХНИКА В МЕХАТРОНИКЕ И РОБОТОТЕХНИКЕ

Ответственный за выпуск Г. С. Леневский

Технический редактор А. А. Подошевко

Компьютерная верстка М. М. Дударева

Подписано в печать . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс.
Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 46 экз. Заказ №

Издатель и полиграфическое исполнение:
Государственное учреждение высшего профессионального образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 24.01.2014.
Пр. Мира, 43, 212000, Могилев.

Содержание

1 Методические указания к лабораторным работам

1.1 Общая информация

Перед началом лабораторного занятия необходимо изучить теоретические сведения к лабораторной работе.

По полученному заданию студент должен:

– произвести подробный анализ поставленной задачи;

– при необходимости разработать блок-схему алгоритма решения задачи;

– в среде Keil µVision создать текст программы в соответствии с разработанной блок-схемой;

– в режиме симулятора произвести отладку программы;

– составить отчет по работе;

– ответить на контрольные вопросы.

2 Лабораторная работа № 1. Изучение архитектуры ARM микроконтроллеров Cortex-M3

Цель работы: изучить структуру, организацию памяти ARM микроконтроллеров семейства Cortex-M3, назначение выводов микросхемы и технические характеристики на примере контроллера семейства STM32F10xx.

2.1 Что такое микропроцессор и как он работает

Процессор – это электронное программно управляемое устройство, предназначенное для обработки цифровой информации и выполненное в виде одной или нескольких БИС.

Из определения следует несколько важных свойств процессора:

1) процессор – это цифровая микросхема, полупроводниковое устройство. Его работа подчиняется тем же законам, что и других цифровых микросхем;

2) процессор – это устройство, которое работает по программе, т. е. алгоритм его действия заложен не в структуре микросхемы, а в последовательности команд, находящихся в памяти.

Вся информация, обрабатываемая процессором, делится на два вида: данные и команды.

Данные – это числовая информация, значения которой несут какие-либо сведения.

Команда – двоичное слово, которое, будучи помещенным в специальный регистр – регистр команд, инициирует выполнение действий по элементарной обработке информации.

Программа – это совокупность команд, выполняемых процессором.

Большинство современных процессоров, в той или иной мере, функционируют в соответствии с принципами фон Неймана. Цикл фон Неймана состоит из нескольких стадий:

– выборка команды – из памяти считывается численный код выполняемой команды;

– увеличение счетчика команд на 1; счетчик команд – специальный регистр процессора, хранящий адрес обрабатываемой команды;

– дешифрация команды – поскольку команды, как правило, хранятся в закодированном виде, то их необходимо расшифровать и определить, какие действия должен выполнять процессор; эту операцию осуществляет дешифратор команд;

– выполнение команды – после того как стало известно, что процессор должен делать и какие данные он должен обрабатывать, происходит исполнение элементарной операции по обработке информации.

2.2 Обзор архитектуры ARM микроконтроллеров семейства Cortex

2.2.1 Общие сведения.

Рынок микроконтроллеров поистине огромен – ежедневно в мире продаются десятки миллионов данных устройств. На этом рынке идёт непрерывная конкурентная борьба между различными производителями, моделями и архитектурами микроконтроллеров. Рост запросов со стороны промышленного сектора вызвал потребность в более производительных микроконтроллерах; в частности, возникла необходимость в микроконтроллерах, которые при той же частоте или потребляемой мощности выполняли бы большее число операций. Кроме того, микроконтроллеры становятся всё более «коммуникабельными», используя для связи с окружающим миром шину USB, Ethernet или радиоканал, и вполне естественно, что для поддержки этих каналов связи и развитых периферийных устройств требуются дополнительные вычислительные ресурсы. Одновременно растёт сложность самих приложений, что обусловлено использованием более изощрённых пользовательских интерфейсов, необходимостью поддержки мультимедиа и увеличением функциональности устройств. Все большую популярность среди разработчиков микропроцессорных устройств завоевывают микроконтроллеры, построенные на базе ARM-архитектуры. Так, например, подавляющее большинство мобильных устройств (мобильные телефоны, планшеты) выпускаются с контроллерами, построенными на базе данной архитектуры. И все большую популярность завоевывают такие устройства на рынке промышленных систем автоматизации. Главные «прорывные» разработки ARM в сфере промышленного применения – это 32-разрядное ядро Cortex-M0, дающее скачок производительности простых приложений при сохранении цены 8-разрядных решений, и 32-разрядное ядро Cortex-M4, чьи богатые вычислительные возможности дают революционный выигрыш в приложениях, где нужна быстрая обработка входного аналогового сигнала с широким динамическим диапазоном (промышленное и лабораторное оборудование, медицинская электроника, обработка изображения или звука), шифрование информации и т. д.

При этом стоит отметить, что в отличие от многих других полупроводниковых компаний компания ARM не занимается ни изготовлением процессоров, ни продажей микросхем. Вместо этого ARM предлагает своим бизнес-

партнерам, в числе которых большинство ведущих мировых полупроводниковых компаний, лицензии на использование разработанных ею процессорных ядер. На основе дешевых и экономичных решений от ARM её партнёры создают собственные процессоры, микроконтроллеры и системы на кристалле. На сегодняшний день партнёры компании ARM ежегодно поставляют более двух миллиардов процессоров ARM. И в этой связи не стоит путать, например, такие понятия, как «Процессор Cortex-M3» и «Микроконтроллер с ядром Cortex-M3». Процессор Cortex-M3 является центральным процессором микроконтроллера, т. е. всего лишь одним, хотя и центральным и наиболее важным, из его многочисленных узлов (рисунок 2.1).

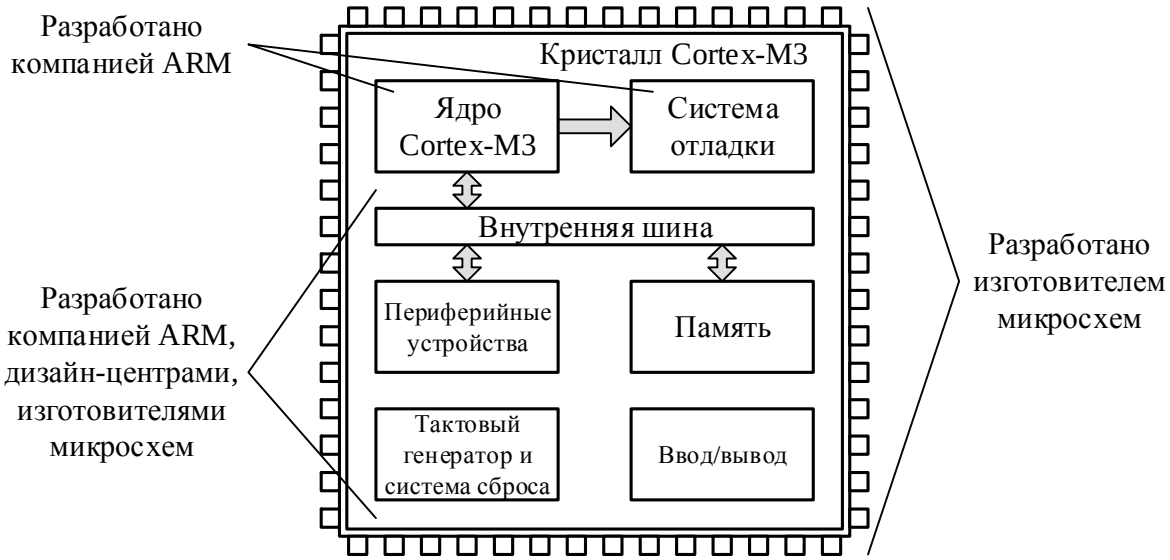


Рисунок 2.1 – Процессор Cortex-M3 и МК с ядром Cortex-M3

2.2.2 Обзор семейства Cortex.

Семейство ARM Cortex – новое поколение процессоров, которые выполнены по стандартной архитектуре и отвечают различным технологическим требованиям. В отличие от других ЦПУ ARM семейство Cortex является завершенным процессорным ядром, которое объединяет стандартное ЦПУ и системную архитектуру. В основе процессора Cortex лежит архитектура ARMv7, доступная в трех главных профилях:

1) профиль A (ARMv7-A) – для высокопроизводительных решений;

2) профиль R (ARMv7-R) – для реально-временных применений;

3) профиль M (ARMv7-M) – для чувствительных к стоимости и микроконтроллерных систем.

Архитектура ARMv7-M подробно описана в [7], где рассмотрены следующие ее ключевые элементы:

– модель программирования;

– набор команд;

– модель памяти;

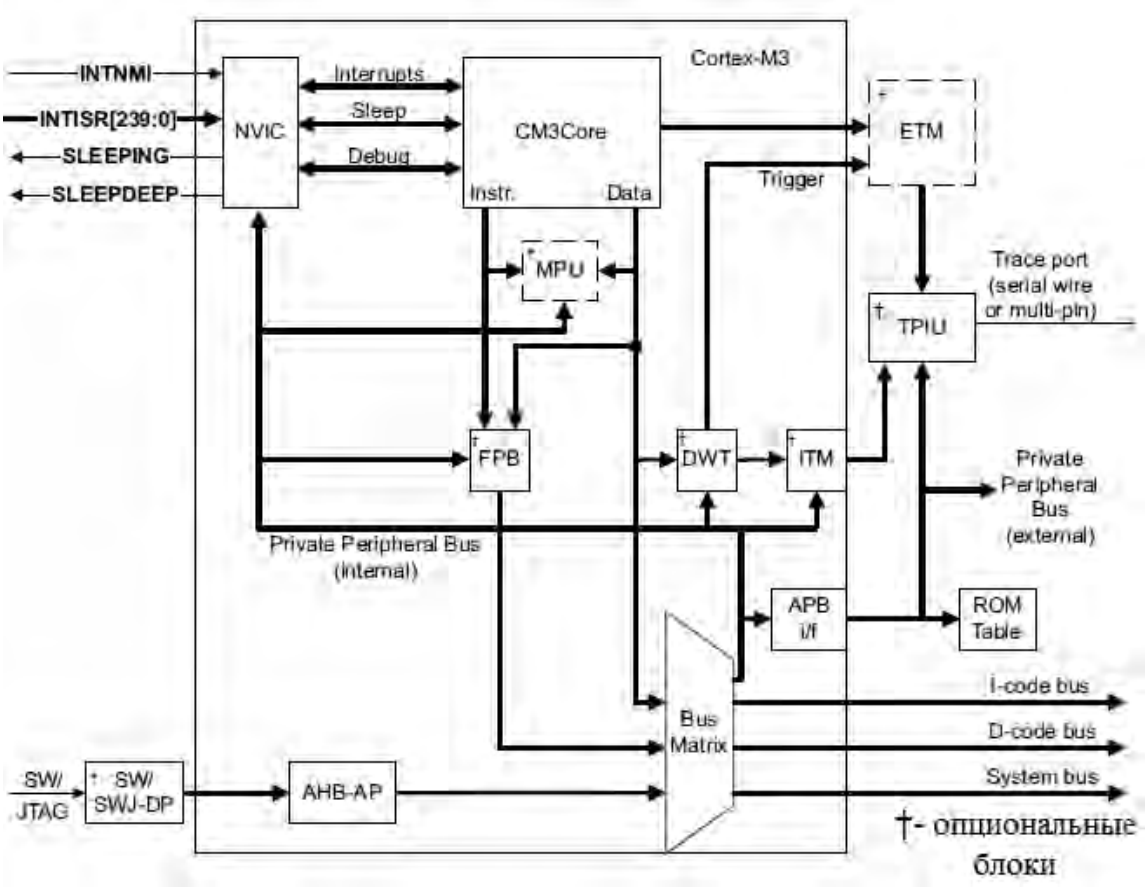
– архитектура отладки.

2.2.3 Структура микропроцессора.

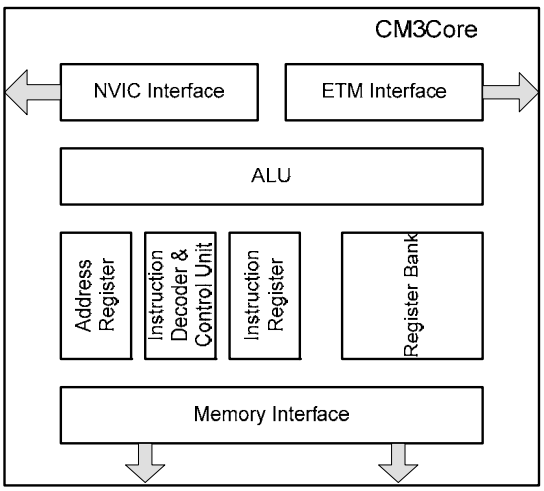
Как уже было указано, основой любого микроконтроллера с ядром Cortex-M3 является процессор Cortex-M3.

Упрощенная структурная схема процессора приведена на рисунке 2.2, *а*.

а)



б)



а – упрощенная структурная схема процессора; *б* – упрощенная структурная схема ядра Cortex-M3

Рисунок 2.2 – Процессор Cortex-M3

Он представляет собой стандартизованный микроконтроллер, интегрирующий 32-битное ЦПУ (ядро процессора CM3Core), шинную структуру (Bus Matrix), контроллер вложенных прерываний (NVIC), отладочную систему (DWT и ITM) и предопределенную организацию памяти. Четыре компонента, ETM (Embedded Trace Macrocell), TPIU (Trace Port Interface Unit), SW / SWJ-DP (Serial Wire / Serial Wire JTAG Debug Port) и ROM Table (таблица постоянной памяти), находятся вне уровня Cortex-M3, потому что они или опциональны, или есть гибкость в их реализации и использовании.

2.2.3.1 Ядро. Как показано на рисунке 2.2, *б*, ARM-ядро содержит следующие основные блоки:

1) Instruction Register – регистр команд или буфер команд;
2) Instruction Decoder и Control Unit – дешифратор команд и блок управления;
3) ALU – арифметическое логическое устройство;
4) Address Register – регистр адреса;
5) Register Bank – банк регистров общего назначения;
6) Memory Interface – интерфейс памяти;
7) NVIC Interface – интерфейс контроллера прерываний;
8) ETM interface – взаимодействует со встроенным трассировщиком ETM (Embedded Trace Macrocell) для того, чтобы обеспечить трассировку команд.

2.2.3.2 Контроллер вложенных векторных прерываний (NVIC) предоставляет стандартную структуру прерываний для всех Cortex-микроконтроллеров и способы их обработки. NVIC предписывает векторы прерываний для 240 источников, для каждого из которых может быть установлен свой приоритет. При разработке NVIC особое внимание уделялось быстродействию обработки прерываний. С момента получения запроса на прерывание до выполнения первой команды процедуры обработки прерывания проходит всего лишь 12 циклов.

2.2.3.3 В состав каждого процессора Cortex-M3 входит системный таймер – SysTick. Он представляет собой 24-битный вычитающий счетчик, предназначенный для формирования периодических сигналов прерываний через равные интервалы времени. Может использоваться операционными системами для генерации системных событий. Входит в состав контроллера прерываний NVIC.

2.2.3.4 Bus Matrix – шинная матрица. Предназначена для соединения процессора и отладочного интерфейса с внешними шинами.

2.2.3.5 В процессоре есть ряд блоков, предназначенных для возможности организации отладки приложений в микроконтроллере. Именно эти блоки позволяют пользователю загружать приложение в память контроллера и выполнять его по шагам, прерывать программу, останавливаясь по завершении каждой операции.

Для этих целей используются блоки:

– Flash Patch and Breakpoint (FPB) – модуль коррекции флэш-памяти и задания точки;

– Data Watchpoint and Trace (DWT) – модуль просмотра и трассировки данных. Реализует функцию точек наблюдения данных;

– Instrumentation Trace Macrocell (ITM) – макроячейка инструментальная.

2.2.3.6 Для поддержки сложных приложений, требующих более развитой

системы памяти, в процессоре Cortex-M3 предусмотрены опциональный модуль защиты памяти (Memory Protection Unit – MPU) и возможность использования внешней кэш-памяти.

2.2.3.7 ETM (Embedded Trace Macrocell) – модуль встроенной макроячейки трассировки – специальный блок процессора, который обеспечивает реконструкцию выполнения программы. ETM разработан как высокоскоростной, малопотребляющий инструмент отладки, поддерживающий трассировку только инструкций.

2.2.3.8 TPIU (Trace Port Interface Unit) – модуль интерфейса порта трассировки. Выступает в качестве моста между данными трассировки Cortex-M3 от ITM и ETM (если присутствует) и внешним трассировщиком-анализатором.

2.2.3.9 SW / SWJ-DP (Serial Wire / Serial Wire JTAG Debug Port) – представляет собой отладочный порт последовательный или JTAG.

2.2.3.10 AHB-AP – порт доступа к шине AHB. Преобразует команды от интерфейса SW/SWJ в транзакции шины AHB и обеспечивает доступ ко всей памяти процессора Cortex-M3 в режиме отладки через порт SW / SWJ.

2.2.4 Организация памяти.

Данные, обрабатываемые процессором, необходимо где-то хранить. Для этих целей микропроцессоры Cortex-M3 могут использовать регистры, являющиеся частью ядра, или запоминающие устройства (ЗУ), входящие в состав микроконтроллера.

2.2.4.1 Карта памяти. Процессор Cortex-M3 является стандартизованным микроконтроллерным ядром и поэтому его карта памяти четко расписана. Несмотря на использование нескольких внутренних шин, адресное пространство линейное и имеет размер 4 Гбайт (рисунок 2.3).

Первые 1 Гбайт памяти разделены равномерно между областью кода программы (Code) и областью статического ОЗУ (SRAM). Пространство кода программы оптимизировано для работы с шиной ICode. Аналогично пространство статического ОЗУ доступно через шину DCode. Несмотря на то, что в области статического ОЗУ поддерживается загрузка и исполнение инструкций, их выборка осуществляется через системную шину, что требует дополнительного состояния ожидания.

Таким образом, выполнение кода программы из статического ОЗУ будет более медленным, чем из встроенной Flash-памяти, расположенной в области кода программы.

Следующие 0,5 Гбайт памяти – область встроенных УВВ (Peripheral), где находятся все предоставляемые пользователю производителем микроконтроллера устройства ввода/вывода. Первые 1 Мбайт в областях статического ОЗУ и УВВ являются бит-адресуемыми. Для этого используется метод bit-banding. Таким образом, все данные, хранящиеся в этих областях, могут обрабатываться как пословно, так и побитно.

Следующие 2 Гбайт адресного пространства выделены для внешних статического ОЗУ (External RAM) и устройств ввода/вывода (External device).

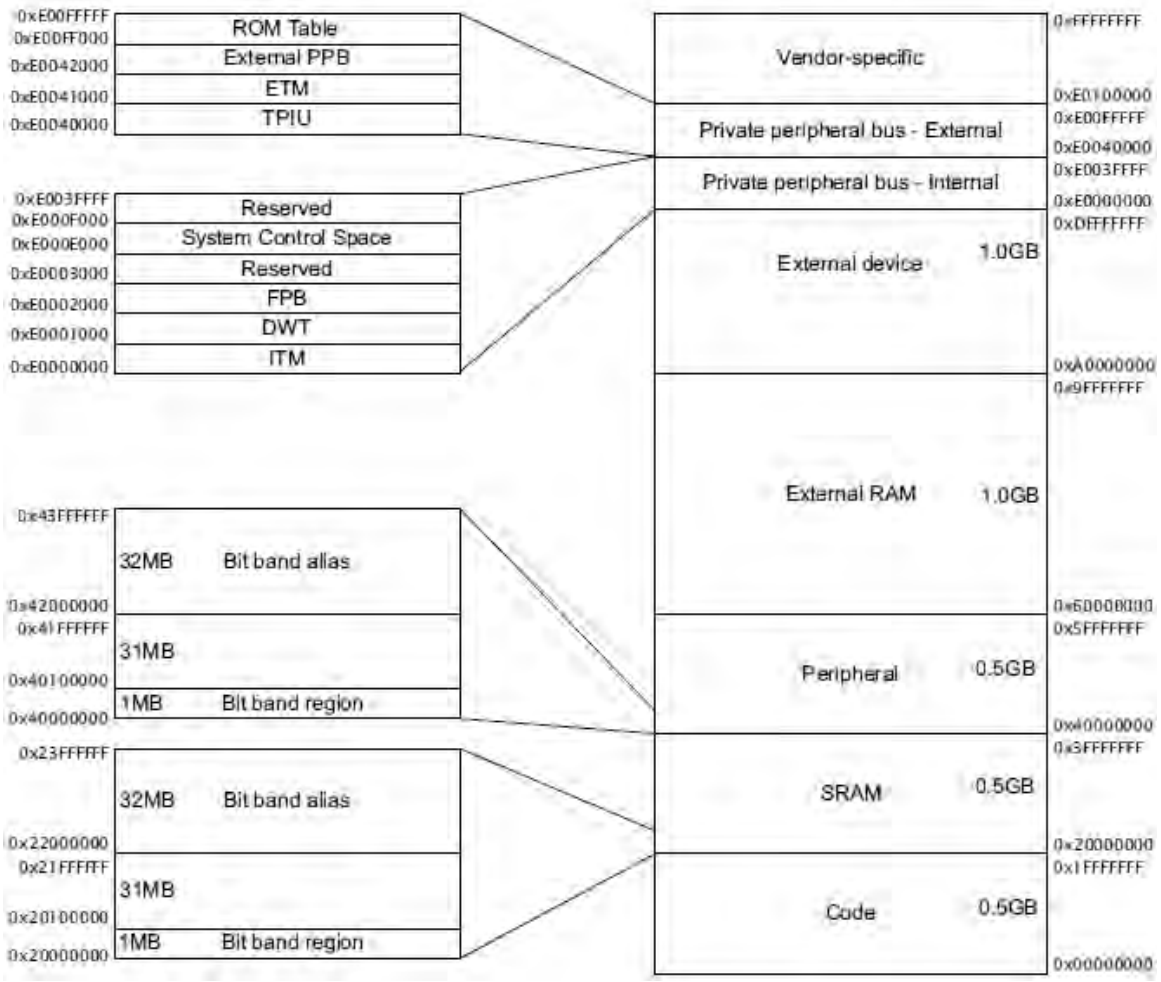


Рисунок 2.3 – Организация памяти микроконтроллеров Cortex-M3

Последние 0,5 Гбайт зарезервированы для системных ресурсов процессора Cortex (Private peripheral bus – Internal – локальная шина УВВ внутренняя, Private peripheral bus – External – локальная шина УВВ внешняя) и будущих расширений процессора Cortex (Vendor specific – специфическая область производителя. В данной области памяти производитель микроконтроллера может размещать свою информацию). Все регистры процессора Cortex расположены по фиксированным адресам во всех Cortex-микроконтроллерах. Благодаря этому облегчается портирование программ между различными Cortex-микроконтроллерами разных производителей.

2.2.4.2 Регистры процессора. Процессор Cortex является RISC-процессором, который выполнен по архитектуре чтения/записи. Для операций обработки данных вначале необходимо поместить операнды из памяти в центральный регистровый файл, затем выполнить требуемую операцию над данными в регистрах и, наконец, перезаписать результат обратно в память.

Следовательно, вся активность программы фокусируется вокруг регистрового файла ЦПУ. Данный регистровый файл образуют шестнадцать 32-битных регистров (рисунок 2.4).

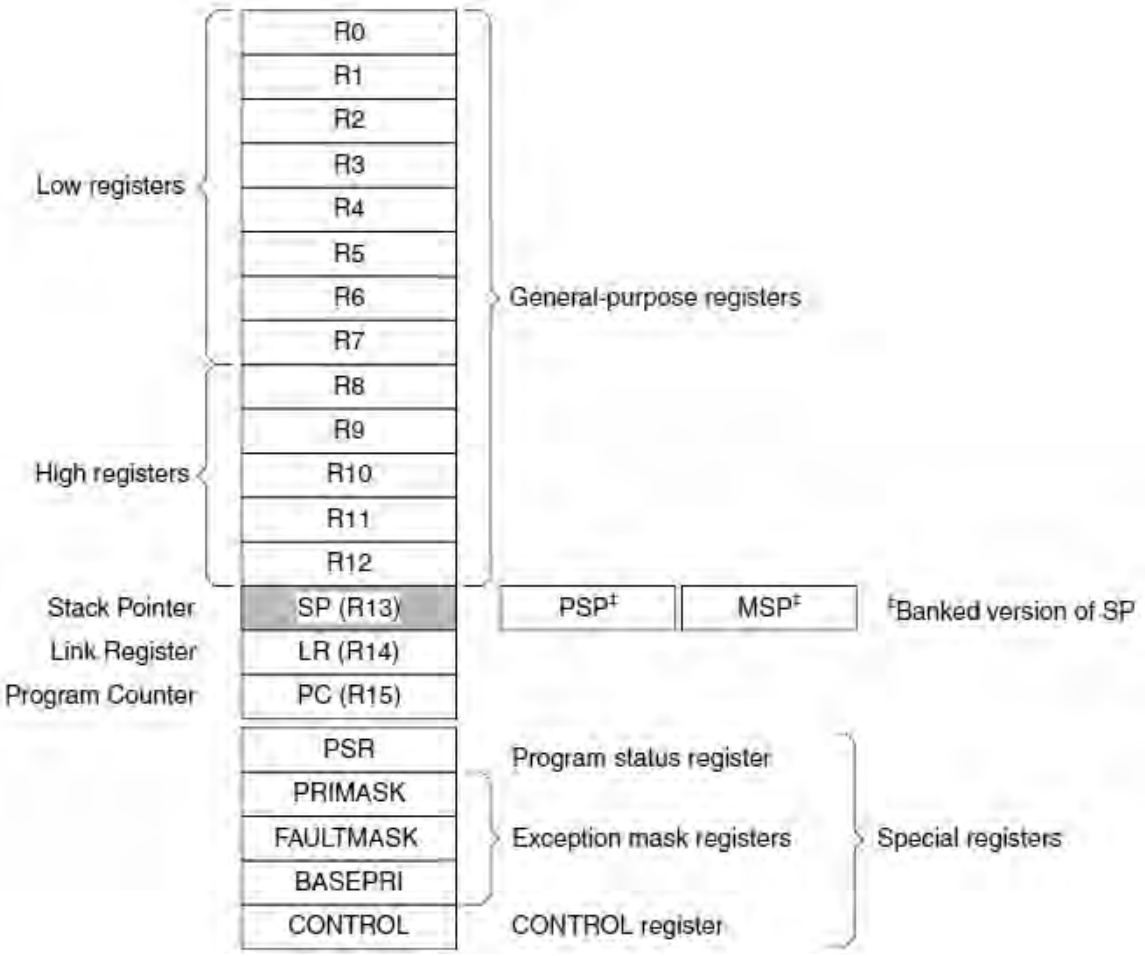


Рисунок 2.4 – Регистровый файл процессора Cortex-M3

Регистры R0…R12 – регистры общего назначения (РОН), которые могут использоваться для хранения программных переменных. У них нет специфического архитектурно-определенного назначения. Большинство инструкций, которые могут обращаться к РОН, могут использовать R0…R12.

При этом регистры делятся на два типа:

1) Low registers – младшие регистры. Регистры R0…R7 доступны всем инструкциям, которые могут обращаться к РОН;

2) High registers – старшие регистры. Регистры R8…R12 доступны всем 32-битным инструкциям, которые могут обращаться к РОН. Регистры R8…R12 недоступны всем 16-битным инструкциям.

У регистров R13…R15 имеются особые функции в рамках ЦПУ Cortex. Регистр R13 выступает в роли указателя стека. Следующий регистр R14 называется регистром связи. Он используется для хранения адреса возврата из подпрограммы. Последний регистр R15 – счетчик программы (или счетчик команд). Поскольку он является частью центрального регистрового файла, его чтение и обработка могут выполняться аналогично любым другим регистрам.

Помимо регистров общего назначения, в процессоре Cortex-M3 имеются следующие регистры специального назначения:

– регистры состояния программы (xPSR);

– регистры маскирования прерываний (PRIMASK, FAULTMASK и BASEPRI);

– регистр управления (CONTROL).

Регистр состояния программы (Program Status Registers (xPSR)) не входит в основной регистровый файл, а доступ к нему возможен с помощью двух специальных инструкций. В xPSR хранятся значения полей, влияющих на исполнение инструкций ЦПУ Cortex.

Регистр xPSR (рисунок 2.5) объединяет в себе три регистра: Application Program Status Register (APSR) – регистр состояния прикладной программы, Interrupt Program Status Register (IPSR) – регистр статуса прерываний, Execution Program Status Register (EPSR) –регистр статуса исполнения программы. К каждому из этих регистров можно обращаться отдельно, по его наименованию или как к комбинации двух или трех регистров.

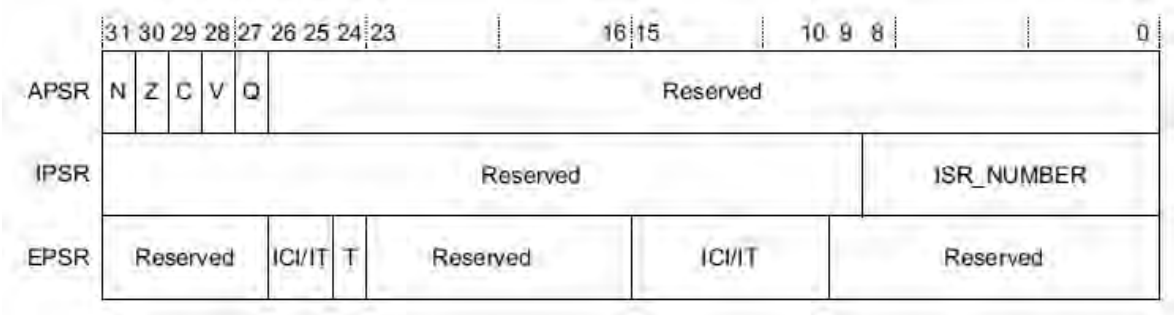


Рисунок 2.5 – Формат регистра статуса программы

Кроме РОН и регистров специального назначения, процессор имеет достаточно большое количество регистров специальных функций, которые контролируют функционирование различных узлов процессора. Подробный разбор каждого регистра выходит за рамки одной лабораторной работы, однако список регистров управления системой и идентификации, регистров таймера SysTick, а также регистров контроллера прерываний, блока защиты памяти MPU и регистров системы отладки можно изучить в [6–9].

2.2.4.3 Стек. Стеком называется упорядоченный набор элементов, в котором размещение новых и удаление существующих происходит с одного конца, называемого вершиной.

Стековая память является безадресной памятью с последовательным доступом. В стековом ЗУ ячейки образуют одномерный массив, в котором соседние ячейки связаны друг с другом разрядными цепями передачи слов. Слова становятся доступными для чтения и записи только в определенном порядке. Каждое хранящееся слово привязано не к конкретной ячейке, а к своему положению относительно других хранящихся слов. Слова могут перемещаться по ячейкам, но при этом сохраняют свою взаимную упорядоченность. Поэтому достаточно обеспечить средства для чтения только определенной ячейки. Конкретное слово считывается в тот момент, когда в процессе перемещения по памяти оно оказывается в ячейке, из которой может производиться чтение. Ана-

логично достаточно обеспечить средства для записи только в определенную ячейку ЗУ.

В процессоре Cortex-M3 используется модель «полного» убывающего стека. Указатель стека указывает на последнее значение, помещённое в стек, и инкрементируется перед выполнением новой операции загрузки в стек. В качестве примера на рисунке 2.6 показано выполнение команды PUSH {R0}.

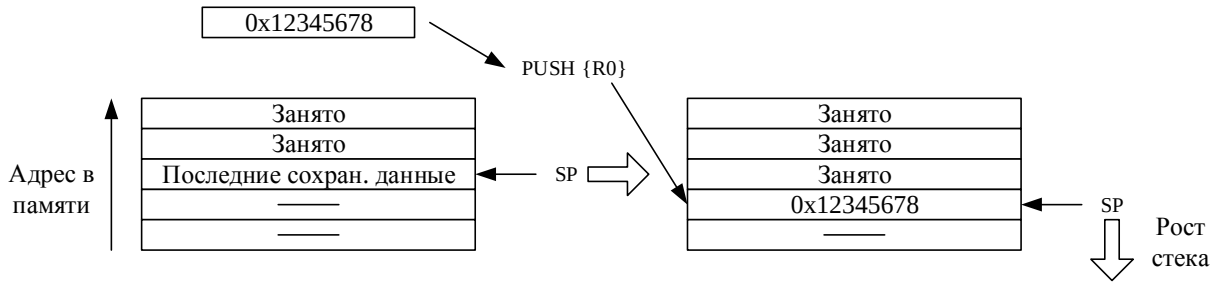


Рисунок 2.6 – Выполнение операции загрузки в стек

Во время операции извлечения из стека данные считываются из памяти по адресу, определяемому указателем стека, после чего указатель инкрементируется. Содержимое ячейки памяти не изменяется, однако оно будет перезаписано при выполнении последующей операции загрузки в стек. Поскольку при выполнении каждой операции загрузки в стек и извлечения из стека осуществляется пересылка 4 байт данных (каждый регистр содержит одно слово, или 4 байт), значение указателя стека уменьшается/увеличивается на 4 или, при одновременном сохранении/восстановлении нескольких регистров, на величину, кратную четырём.

Более подробные сведения об особенностях и режимах работы процессора семейства Cortex-M3, технических характеристиках микроконтроллера STM32F103 можно найти в методических указаниях по лабораторной работе № 1, электронная версия которых имеется в классе ПЭВМ кафедры (лаб. 207/2) по следующему адресу: D:\методические рекомендации\СТАНДАРТЫ РФ\Специальность 15.03.06 Мехатроника и робототехника\Микропроцессорная техника в мехатронике и робототехнике\МПТ ЛР 01.pdf, а также на сервере филиала кафедры по адресу \\Study\Методические указания\МПТ\МПТ ЛР 01.pdf.

Контрольные вопросы

1 Перечислите характерные черты ARM микроконтроллеров семейства Cortex-M3.

2 Какие структурные элементы входят в состав процессора Cortex-M3?

3 Назовите назначение структурных элементов, входящих в состав процессора Cortex-M3.

4 Каким образом организована память процессора Cortex-M3?

5 Укажите назначение регистров процессора Cortex-M3.

6 Поясните формат слова состояний xPSR.

7 Поясните особенности использования стека в процессоре Cortex-M3.

8 Перечислите основные особенности микроконтроллера STM32F103R8.

9 Назовите объем резидентной памяти данных и программ микроконтроллера STM32F103R8.

10 Укажите отличительные особенности организации памяти микроконтроллера STM32F103R8.

11 Перечислите назначение выводов микроконтроллера.

12 Перечислите альтернативные функции параллельных портов.

3 Лабораторная работа № 2. Изучение программного обеспечения для программирования микроконтроллеров ARM

Цель работы: изучить программное обеспечение для программирования микроконтроллеров ARM, освоить технику программной симуляции работы микроконтроллера ARM на примере построения элементарных конструкций.

3.1 Основные сведения о Keil µVision4

Keil ARM Microcontroller Development Kit (MDK-ARM®) – это среда разработки программного обеспечения для микроконтроллеров с микропроцессорными ядрами фирмы ARM: Cortex™-M, Cortex-R4, ARM7™ и ARM9. В ее состав входят:

– интегрированная среда разработки µVision;

– оптимизирующий C/C++ кросс-компилятор;

– макроассемблер ARM;

– компоновщик (редактор связей);

– ARM C/C++ библиотеки;

– отладчик µVision;

– библиотеки промежуточного программного обеспечения.

µVision – это платформа разработки программного обеспечения под управлением Windows, которая сочетает в себе эффективный и современный редактор с менеджером проекта и средство сборки. Она объединяет все инструменты, необходимые для разработки встраиваемых приложений, включая C/C++ компилятор, макроассемблер, компоновщик, а также генератор HEX файла. µVision помогает ускорить процесс разработки встраиваемых приложений, предоставляя:

– менеджер проектов;

– базу данных устройств, позволяющую автоматически настроить среду разработки в соответствии с используемым микроконтроллером;

– текстовый редактор;

– встроенную утилиту сборки, позволяющую скомпилировать и скомпоновать встроенное приложение;

– символьный отладчик;

– симулятор;

– профайлер;
– встроенную справочную систему;
– программатор и некоторые другие средства.

Поскольку компилятор в среде Keil разработан самой фирмой ARM, то его можно считать лучшим компилятором среди всех имеющихся для чипов с ядром ARM. Не последнюю очередь при выборе среды Keil MDK-ARM сыграло наличие адаптированной для нее библиотеки промежуточного программного обеспечения (middleware), включающей:

– операционную систему реального времени (RTOS) RTX с открытым исходным кодом;
– файловую систему для чипов последовательной Flash-памяти и для SD/MMC карт памяти;
– процедуры работы с CAN-шиной;
– стек протоколов TCP/IP с прикладным уровнем;
– стеки протоколов USB device и USB host;
– графический пользовательский интерфейс (GUI).

Кроме того, при необходимости среда Keil может использовать компилятор GCC вместо компилятора фирмы ARM, что может представлять собой недорогой бюджетный вариант среды разработки.

Стоит отметить, что существуют версии µVision, позволяющие разрабатывать приложения не только для ARM-микроконтроллеров, но и для контроллеров семейств C51, C166, C251, предоставляя разработчикам программ единую среду разработки.

Первый этап разработки программы – запись её исходного текста на каком-либо языке программирования.

Затем производится компиляция или трансляция его в коды из системы команд микроконтроллера, используя транслятор или ассемблер. Трансляторы и ассемблеры — прикладные программы, которые интерпретируют текстовый файл, содержащий исходный текст программы, и создают объектные файлы, содержащие объектный код.

После компоновки объектных модулей наступает этап отладки программы, устранения ошибок, оптимизации и тестирования программы.

µVision объединяет все этапы разработки прикладной программы в единый рекурсивный процесс, когда в любой момент времени возможен быстрый возврат к любому предыдущему этапу.

Более подробные сведения о возможностях µVision, назначении элементов программы, способах разработки и отладки программ с примером выполнения работы можно найти в методических указаниях по лабораторной работе № 2, электронная версия которых имеется в классе ПЭВМ кафедры (лаб. 207/2) по следующему адресу: D:\методические рекомендации\СТАНДАРТЫ РФ\Специальность 15.03.06 Мехатроника и робототехника\Микропроцессорная техника в мехатронике и робототехнике\МПТ ЛР 02.pdf, а также на сервере филиала кафедры по адресу \\Study\Методические указания\МПТ\МПТ ЛР 02.pdf.

3.2 Варианты индивидуальных заданий к лабораторной работе

Составить программу по заданию. Вариант задания получить у преподавателя. Программа должна выполнять все действия и расчеты, приведенные в задании, в том числе и промежуточные. В режиме пошаговой отладки исследовать выполнение команд программы.

Найти время выполнения программы. Сгенерировать hex-файл.

Составить отчет.

Отчет должен содержать цель работы, постановку задачи, текст программы, таблицу хода выполнения программы, выводы по работе.

Контрольные вопросы

1 Как просмотреть содержимое памяти, регистров, портов и переменных?
2 Как узнать реальное время выполнения программы?
3 В чем отличие Step, Step Over и Step Out в пошаговом режиме отладки?
4 Какие существуют способы добавления файла в проект?

4 Лабораторная работа № 3. Изучение системы команд Cortex-M3 микроконтроллеров

Цель работы: изучить состав и особенности выполнения команд микроконтроллера, а также способы адресации операндов в командах.

В ходе работы необходимо:
– изучить систему команд микроконтроллера;
– составить фрагменты программ в редакторе µVision в соответствии с полученным вариантом по каждой группе команд;
– исследовать выполнение команд, входящих в состав разработанных программ;
– составить отчет о выполнении работы.
По окончании работы ответить на контрольные вопросы.

4.1 Общие сведения о системе команд

4.1.1 Система команд процессоров ARM.

Младшие версии ARM-процессоров имеют 32-разрядный набор команд, ориентированный преимущественно на обработку 32-разрядных данных.

Высокая разрядность данных, заложенная в ARM изначально, хорошо подходит для устройств с большой вычислительной нагрузкой. «Широкие» команды дают возможность в коде одной команды реализовать достаточно сложную обработку данных, для которой в других процессорах потребуется несколько команд, например, совместить сдвиг, сумму и перенос в другой регистр. Но, с другой стороны, эти же свойства снижают применимость младших версий ARM-процессоров в качестве вычислительных ядер для недорогих,

встраиваемых или мобильных устройств с малоразрядной периферией, получивших сейчас большое распространение. Для реализации простой логики управления 4-байтная команда занимает слишком много места в памяти, а 32 разряда данных используются неэффективно.

Для придания современным версиям ARM-процессоров большей универсальности добавили второй набор 2-байтных команд, гораздо более эффективный по размеру кода. Набор укороченных команд называется Thumb, а набор «стандартных» команд – ARM-наборами. Несмотря на то, что «внутри» процессора команды Thumb преобразуются перед исполнением в ARM-инструкции, с точки зрения программиста эти наборы настолько отличаются, что вполне могли бы принадлежать разным процессорам. Переход от одного набора к другому может выполняться в любом месте. Почти любой алгоритм можно написать либо на ARM, либо на Thumb по усмотрению программиста. Но поскольку в Thumb-наборе нет команд, требуемых для обработки исключительных ситуаций (к примеру, сброса или прерываний), любая программа требует включения фрагментов ARM. После сброса или наступления исключительной ситуации процессор принудительно переключается к исполнению команд ARM-набора.

Процессоры Cortex поддерживают набор инструкций Thumb-2, который является смесью 16- и 32-битных инструкций. Инструкции Thumb-2 дают улучшение плотности кода на 26 % по сравнению с 32-битными инструкциями ARM и производительности на 25 % по сравнению с 16-битными инструкциями Thumb. В наборе инструкций Thumb-2 предусмотрены несколько улучшенных инструкций умножения, исполняющихся за один цикл, и аппаратный делитель, требующий от 2 до 7 циклов.

4.1.2 Система команд процессоров Cortex-M3.

Система команд Cortex-M3 содержит 115 базовых команд набора Thumb-2, количество которых может быть значительно увеличено за счет использования суффиксов условного исполнения. Все многообразие команд компания ARM условно разделила по функциональному признаку на следующие группы: команды доступа к памяти, общие команды обработки данных, команды умножения и деления, команды насыщения, команды работы с битовыми полями, команды управления и ветвления, прочие команды.

Микросхемы семейства Cortex-M3 – это 32-разрядные микропроцессоры, а это означает, что регистры общего и специального назначения, арифметико-логическое устройство (АЛУ) и внешние шины имеют 32-битную организацию.

Все команды набора Thumb-2 по формату делятся на два типа: 16-битные (2 байта), 32-битные (4 байта).

Система команд Cortex-M3 включает команды умножения, деления, вычитания, логические операции и сдвиги, операций над битовыми полями, операций со стеком и расширенный набор команд передачи управления, команды загрузки/сохранения регистров и т. д. Большинство команд выполняются за один или два машинных цикла.

4.2 Типы операндов и способы адресации данных

Процессоры Cortex-M3 поддерживают следующие типы данных в памяти: 8-битные байты, 16-битные полуслова и 32-битные слова.

Регистры процессора 32-битные. Набор команд содержит инструкции, поддерживающие следующие типы данных в регистрах:

– 32-битные указатели;

– беззнаковые и знаковые 32-разрядные целые числа;

– беззнаковые 16-битные или 8-битные целые числа, представленные в дополненной нулями форме;

– знаковые 16-разрядные или 8-разрядные целые числа, представленные в дополненной знаком форме;

– знаковые или беззнаковые 64-разрядные целые числа, хранящиеся в двух регистрах.

Операции загрузки и сохранения могут передать байт, полуслова или слова в память и из памяти.

Система команд включает команды загрузки и сохранения, которые передают два или несколько слов в память и из памяти. Используя такие инструкции, можно загрузить и сохранить 64-битные целые числа.

Если любой из типов данных описывается как беззнаковый, то N-битное значение представляет собой неотрицательное целое в диапазоне от 0 до $2^N - 1$ с помощью обычного двоичного формата.

Если любой из этих типов описан как знаковый, то N-битное значение представляет собой целое число в диапазоне от -2^{N-1} до $+2^{N-1} - 1$ в дополнительном коде.

Прямая поддержка инструкций для 64-разрядных целых чисел ограничена, и большинство 64-разрядных операций требуют последовательности из двух или несколько инструкций, чтобы синтезировать их.

Как было отмечено, команды, выполняемые процессором, хранятся в закодированном виде.

При этом любая команда может состоять из нескольких полей:

– поля кода операции, в котором кодируется, какую операцию должен выполнять процессор;

– адресной части, в которой кодируется расположение данных, требуемых для выполнения операции.

Способ определения по адресной части команды физического расположения данных называется способом адресации. В процессорах Cortex-M3 используются следующие способы адресации данных: непосредственная, регистровая, косвенно-регистровая, индексная (в нескольких разновидностях), стековая.

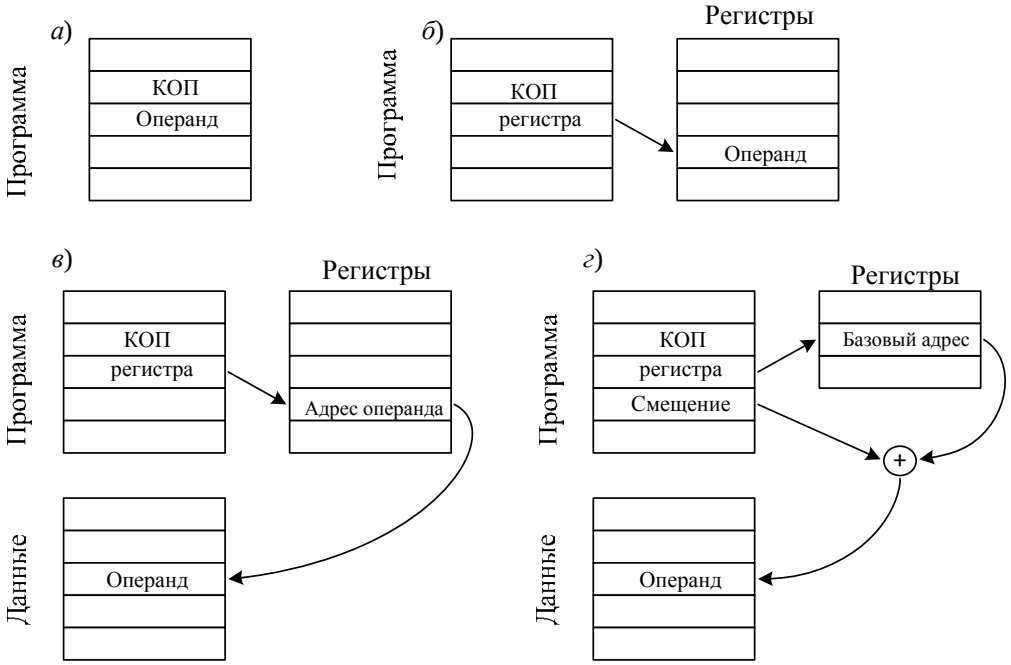
Регистровая адресация является одной из самых часто применяемых в командах процессора и используется для доступа к регистрам общего назначения (рисунок 4.1, *б*). При регистровой адресации в команде задается номер регистра Rn, который выступает источником или приёмником результата операции.

Например:

```
ORN  R2,R1,R0      ;Все операнды в команде используют регистровую адресацию
```

Косвенно-регистровая адресация используется для обращения к ячейкам памяти. При этом способе адресации в команде записан номер регистра, который выступает в качестве регистра-указателя адреса. Его значение является адресом искомой ячейки памяти (рисунок 4.1, *в*). Например:

```
LDR  R3,  [R2]      ;в первом операнде использована регистровая адресация, во втором – косвенно-регистровая
```



а – непосредственная; *б* – регистровая; *в* – косвенно-регистровая; *г* – индексная

Рисунок 4.1 – Адресация данных в командах ЦП Cortex-M3

При непосредственной адресации используемые операнды явно указаны в команде (рисунок 4.1, *а*), например:

```
MOV R0, #0x14    ;В первом операнде использована регистровая адресация, а во втором – непосредственная
```

В операциях загрузки/сохранения адрес для загрузки или сохранения образуется из двух частей: значения из базового регистра и смещения (рисунок 4.1, *г*). Базовый регистр может быть любым из регистров общего назначения. При загрузке базовым регистром может быть счетчик команд PC.

Смещение может быть одного из трех форматов:

1) непосредственное;

2) регистр;

3) масштабируемый регистр.

Чтобы сформировать адрес ячейки, смещение и базовый регистр могут использоваться тремя различными способами.

1 Смещение – смещение добавляется или вычитается из базового регистра для формирования адреса ячейки памяти. Синтаксис ассемблера для этого способа

[<Rn>,<offset>]

Например:

```
STRB R1,[R0,#1]   ;В первом операнде использована регистровая адресация, а во втором – индексная (косвенно-регистровая со смещением)
```

2 Предындексирование – смещение добавляется или вычитается из базового регистра, чтобы сформировать адрес ячейки памяти. Базовый регистр после этого обновляется полученным адресом, чтобы обеспечить автоматическую индексацию через массив или блок памяти. Синтаксис ассемблера для этого способа

[<Rn>,<offset>]!

Например:

```
STRB  R1,[R0,#1]!    ;В первом операнде использована регистровая адресация, а во втором – индексная с предындексированием
```

3 Постиндексирование – значение базового регистра используется в качестве адреса ячейки памяти. Затем смещение добавляется или вычитается из базового регистра, и это значение сохраняется в базовом регистре, чтобы разрешить автоматическое индексирование массива или блока памяти. Синтаксис ассемблера для этого способа

[<Rn>],<offset>

Например:

```
STRB R1,[R0],#1   ;В первом операнде использована регистровая адресация, а во втором – индексная с постиндексированием
```

Во всех случаях <Rn> – это базовый регистр, в качестве которого выступают регистры общего назначения, <offset> – смещение, которое может быть:

– непосредственной константой, 8-битной <imm8> или 12-битной <imm12>;

– индексным регистром <Rm>;

– сдвинутым индексным регистром <Rm>, LSL #<shift>, где 0 ≤ shift ≤ 3.

Многие команды обработки данных имеют «гибкий» второй операнд. В описании синтаксиса таких команд этот операнд обозначается как Operand2 или Op2. В зависимости от команды второй операнд Op2 может быть:

– константой;

– регистром с необязательным сдвигом.

Константа, используемая в качестве Op2, записывается в виде

#constant,

где constant может быть:

– любой константой, которая может быть получена сдвигом 8-битного значения влево на произвольное число битов в пределах 32-битного слова;

– любой константой вида 0x00XY00XY;

– любой константой вида 0xXY00XY00;

– любой константой вида 0xXYXYXYXY.

Символы X и Y выше обозначают шестнадцатеричные числа.

В случае, если Op2 используется как регистр с необязательным сдвигом, он записывается в виде

Rm {, shift},

где Rm – регистр, содержащий значение для второго операнда;

shift – необязательная операция сдвига, выполняемая над содержимым регистра Rm. Для указания этой операции может использоваться одна из следующих мнемоник.

ASR #n Арифметический сдвиг вправо на n битов, 1 ≤ n ≤ 32.

LSL #n Логический сдвиг влево на n битов, 1 ≤ n ≤ 31.

LSR #n Логический сдвиг вправо на n битов, 1 ≤ n ≤ 32.

ROR #n Циклический сдвиг вправо на n битов, 1 ≤ n ≤ 31.

RRX Расширенный циклический сдвиг вправо на 1 бит.

Например:

```
ADD  R1,R1,R0,LSL  #1     ;В первом и втором операндах использована регистровая адресация, в третьем – регистровая с масштабированием (причем для указания номера регистра используется регистровая адресация, а сдвиг указывается непосредственно)
```

Если мнемоника отсутствует, то сдвиг не выполняется.

При указании операции сдвига она выполняется над содержимым регистра Rm, а итоговое 32-битное значение используется в команде. Содержимое регистра Rm при этом не изменяется. Некоторые команды при использовании регистра со сдвигом также изменяют состояние флага переноса.

4.3 Признаки результата выполнения операции и условное выполнение

В регистре состояния прикладной программы APSR содержатся следующие флаги условия:

– N (отрицательное значение) – устанавливается в 1, если результат операции был отрицателен, иначе сбрасывается в 0;

– Z (ноль) – устанавливается в 1, если результат операции был равен нулю, иначе сбрасывается в 0;

– C (перенос) – устанавливается в 1, если в результате операции произошёл перенос, иначе сбрасывается в 0 (флаг используется при обработке беззнаковых данных);

– V (переполнение) – устанавливается в 1, если в результате операции произошло переполнение, иначе сбрасывается в 0 (флаг используется при обработке данных со знаком).

В 27-м бите регистра расположен пятый флаг – Q. Этот флаг предназначен для выполнения математических операций с насыщением и не применяется в командах условных переходов.

Перенос возникает:

– если результат сложения больше или равен 2^{32};

– если результат вычитания положителен или равен нулю;

– в результате работы внутренней схемы циклического сдвига при выполнении команд пересылки данных или команд логических операций.

Переполнение возникает, когда знак результата, содержащийся в 31-м бите, отличается от знака, который получился бы при бесконечно большой точности, например:

– если при сложении двух отрицательных чисел получается положительное число;

– если при сложении двух положительных чисел получается отрицательное число;

– если при вычитании положительного числа из отрицательного получается положительное число;

– если при вычитании отрицательного числа из положительного получается отрицательное число.

Большинство команд изменяют флаги состояния только в том случае, если в мнемонике команды указан суффикс S.

В системе команд присутствуют команды, допускающие так называемое условное выполнение. Такие команды имеют необязательное поле кода условия, определяемого двухсимвольным суффиксом {cond}. Условное выполнение требует предварительного исполнения команды IT. Команда, содержащая код условия, выполняется только в том случае, если состояние флагов условий регистра APSR соответствует заданному условию. Все допустимые суффиксы условия выполнения приведены в таблице 4.1.

Таблица 4.1 – Суффиксы условия выполнения

Суффикс	Флаги	Значение
EQ	Z = 1	Равно
NE	Z = 0	Не равно
CS или HS	C = 1	Выше или равно (беззнаковое «≥»)
CC или LO	C = 0	Ниже (беззнаковое «<»)
MI	N = 1	Отрицательный результат
PL	N = 0	Положительный результат либо нуль
VS	V = 1	Переполнение
VC	V = 0	Нет переполнения
HI	C = l и Z = 0	Выше (беззнаковое «>»)
LS	C = 0 или Z = 1	Ниже или равно (беззнаковое «≤»)
GE	N = V	Больше или равно (знаковое «≥»)
LT	N ≠ V	Меньше (знаковое «<»)
GT	Z = 0 и N = V	Больше (знаковое «>»)
LE	Z = l и N ≠ V	Меньше или равно (знаковое «≤»)
AL	Безразлично	Всегда; действие по умолчанию при отсутствии суффикса

Более подробные сведения о наборе команд, сгруппированных по функциональному назначению, можно найти в методических указаниях по лабораторной работе № 2, электронная версия которых имеется в классе ПЭВМ кафедры (лаб. 207/2) по следующему адресу: D:\методические рекомендации\СТАНДАРТЫ РФ\Специальность 15.03.06 Мехатроника и робототехника\Микропроцессорная техника в мехатронике и робототехнике\МПТ ЛР 03.pdf, а также на сервере филиала кафедры по адресу \\Study\Методические указания\МПТ\МПТ ЛР 03.pdf.

4.4 Варианты индивидуальных заданий к лабораторной работе

Изучение системы команд производится по группам. При выполнении лабораторной работы предлагается исследовать следующие группы команд:

– команды передачи данных;

– команды общей обработки данных;

– команды операций с битовыми полями;

– команды передачи управления.

При выполнении работы необходимо составить программы по каждой группе команд. Задания к лабораторной работе следует получить у преподавателя. Программы должны выполнять все действия и расчеты, приведенные в задании, в том числе и промежуточные.

Отладить каждую программу в среде μVision.

Исследовать выполнение команд каждой программы.

Составить таблицу хода выполнения программы.

Составить отчет.

Отчет должен содержать цель работы, постановку задачи, текст программы, таблицы хода выполнения программ по всем частям лабораторной работы, выводы по работе.

Более подробные сведения о системе команд процессора Cortex-M3 можно найти в методических указаниях по лабораторной работе № 3, электронная версия которых имеется в классе ПЭВМ кафедры (лаб. 207/2) по следующему адресу: D:\методические рекомендации\Специальность 15.03.06 Мехатроника и робототехника\Микропроцессорная техника в мехатронике и робототехнике\МПТ ЛР 03.pdf, а также на сервере филиала кафедры по адресу \\Study\Методические указания\МПТ\МПТ ЛР 03.pdf.

Контрольные вопросы

1 Сколько команд включает система команд микроконтроллера?
2 На какие группы разделены эти команды?
3 С какими типами данных может оперировать микроконтроллер?
4 Укажите назначение флагов слова состояния программы xPSR.
5 Какие способы адресации используются в микроконтроллере?
6 Приведите примеры логических и арифметических команд.
7 Как инвертировать отдельные биты двоичных слов?
8 Какие переходы возможны в командах управления?
9 Поясните принципы организации условного выполнения команд.
10 Какие команды используются при организации подпрограмм?
11 Какие флаги используются командами условных переходов?

5 Лабораторная работа № 4. Разработка типовых программ обработки информации

Цели работы: изучить алгоритмы типовых программ обработки информации, основные правила программирования на языке ассемблера и наиболее употребительные директивы. Составить алгоритм и программу обработки, исследовать ход ее работы в отладчике μVision.

5.1 Методика разработки прикладного программного обеспечения микроконтроллерных систем

5.1.1 Общие сведения.

Если задача на разработку прикладной программы для микроконтроллера поставлена, то для получения текста исходной программы необходимо выполнить ряд последовательных действий.

1 Подробно описать задачу.

2 Проанализировать задачу.

3 Выполнить инженерную интерпретацию задачи, желательно с привлечением того или иного аппарата формализации (граф автоматов, сети Петри, матрицы состояний и т. п.).

4 Разработать общую схему алгоритма работы контроллера.

5 Разработать детализированные схемы отдельных процедур, выделенных на основе модульного принципа составления программ.

6 Детально проработать интерфейс контроллера и внести исправления в общую и детализированные схемы алгоритмов.

7 Распределить рабочие регистры и память.

8 Сформировать текст исходной программы.

9 Определить, что должен делать модуль.

10 Определить способы получения модулем исходных данных (от датчиков через порты ввода, из таблиц в памяти или через рабочие регистры).

11 Определить необходимость какой-либо предварительной обработки введенных исходных данных (маскирование, сдвиг, масштабирование, перекодировка).

12 Определить метод преобразования входных данных в требуемые выходные. Используя операторы процедур и условные операторы принятия решения, отобразить на языке схемы алгоритма выбранный метод.

13 Определить способы выдачи из модуля обработанных данных (передать в память, в вызывавшую программу или в порты вывода).

14 Определить необходимость какой-либо вторичной обработки выводимых данных (изменение формата, перекодирование, масштабирование, маскирование).

15 Вернуться к пункту 1 настоящего перечня и проанализировать полученный результат. Выполнить итеративную корректировку схемы алгоритма с целью сделать её простой, логичной, стройной и обладающей четким графическим образом.

16 Проверить работоспособность алгоритма на бумаге путем подстановки в него действительных данных. Убедиться в его сходимости и результативности.

17 Рассмотреть предельные случаи и попытаться определить граничные значения информационных объектов, с которыми оперирует алгоритм, за пределами которых он теряет свойства конечности, сходимости и результативности. Особое внимание при этом следует уделить анализу возможных ситуаций переполнения разрядной сетки, изменения знака результата операции, деления на переменную, которая может принять нулевое значение.

18 Провести мысленный эксперимент по определению работоспособности алгоритма в реальном масштабе времени, когда стохастические события, происходящие в объекте управления, могут оказать влияние на работу алгоритма. При этом самому тщательному анализу следует подвергнуть реакцию алгоритма на возможные прерывания с целью определения критических операторов, которые необходимо защитить от прерываний. Кроме того, в ходе мысленного эксперимента нужно проанализировать логику алгоритма с целью определения таких последовательностей операторов, при выполнении которых микроконтроллер может «не заметить» кратковременных событий в объекте управления. При обнаружении таких ситуаций в логику следует внести коррективы.

5.1.2 Процедуры и подпрограммы.

При разработке микроконтроллерных систем могут быть использованы два способа организации прикладных программ: монолитный и модульный. При первом способе вся прикладная программа разрабатывается как единое целое. При втором она строится из отдельных программных блоков, каждый из которых реализует некоторую процедуру обработки данных или управления. Взаимосвязь блоков определяется разработчиком при монтаже из этих блоков законченной прикладной программы.

Отдельные фрагменты прикладной программы могут быть получены в виде линейной последовательности блоков, другие (многократно используемые) обычно оформляются в виде программ, к которым прикладная программа, называемая основной, имеет возможность обратиться по мере необходимости. Программа должна обладать следующими свойствами: выполнять законченную процедуру обработки данных, иметь только один вход и один выход и не обладать эффектом последействия, при котором текущее выполнение подпрограммы оказывало бы влияние на её последующие выполнения.

5.1.3 Сохранение параметров основной программы.

Иногда при обращении к подпрограмме возникает необходимость сохранить не только адрес возврата в основную программу, но и содержимое отдельных рабочих регистров. У некоторых микроконтроллеров эта задача решается удобным способом – переключением банка регистров. Например, если основная программа использует банк регистров 0, то подпрограмма может использовать банк регистров 1. В случае отсутствия такой возможности можно сохранить параметры основной программы в стеке.

5.1.4 Передача параметров.

Для успешной работы любой подпрограммы необходимо однозначно определить способ передачи в неё исходных данных и способ вывода результата её работы. Подпрограмма, которой требуется дополнительная информация в виде параметров её настройки или операндов, называется параметризуемой.

Получили распространение четыре способа передачи параметров: через память, через регистры общего назначения, через регистр признаков и через стек.

При передаче входных параметров через память основная программа обязательно содержит команды загрузки некоторых ячеек памяти, а подпрограмма – команды считывания из этих ячеек. При передаче входных параметров подпрограмма должна загрузить некоторые ячейки памяти, а основная программа – считать.

Передача параметров через регистры общего назначения осуществляется аналогичным образом.

Способ передачи через стек позволяет использовать в качестве параметра содержимое указателя стека. Часто этот способ оказывается удобнее. Через регистры можно передать столько параметров, сколько регистров общего назначения имеет используемый микроконтроллер (например, контроллеры MCS-51

имеют восемь регистров, ARM могут использовать 13), а через стек – сколько угодно. Однако следует помнить, что при вызове подпрограммы многие микроконтроллеры в стеке сохраняют адрес возврата из неё.

Передачу параметров через регистр признаков удобно использовать при передаче выходных параметров (например, в подпрограммах сравнения чисел). В этом случае подпрограмма должна установить (или сбросить) соответствующие признаки, а основная программа – проанализировать их значение. Например, микроконтроллеры семейства MCS-51 обладают большими возможностями для передачи параметров через признаки. В нем имеется 128 флагов пользователя, доступных для модификации и анализа.

Использование процедур, оформленных в виде подпрограмм, при разработке программного обеспечения имеет ряд достоинств. Прежде всего, относительно простые модули, выделенные из сложной программы, могут программироваться несколькими разработчиками с целью сокращения времени проектирования. Еще более важным является то, что любая подпрограмма допускает автономную отладку. Это, как правило, многократно сокращает время отладки всего прикладного программного обеспечения. И, наконец, механизм использования подпрограмм уменьшает длину прикладной программы, что имеет своим следствием уменьшение требующейся емкости памяти программ.

5.2 Особенности разработки типовых программ для ARM в ассемблере

5.2.1 Структура строки программы, написанной на ассемблере.

Ассемблер – это специфический язык. Программирование на нём осуществляется почти на уровне железа, на уровне архитектуры ядра микроконтроллера. Следовательно, сколько архитектур, столько и ассемблеров. К ассемблеру мало подходит слово «изучить», к нему больше подходит слово «выучить» как, например, таблицу умножения. Если в таблице умножения требуется запомнить значения сомножителей и результат перемножения, то в случае ассемблера требуется запомнить мнемонику команды и результат её элементарного воздействия на состояние микроконтроллера. В руководствах по семейству микроконтроллеров система команд записывается в виде таблицы, где кратко поясняется назначение каждой команды. Если чуть прищуриться, то она будет очень похожа на таблицу умножения.

Несмотря на достаточно большое разнообразие систем команд, есть в них общая черта – это структура команды, она у всех одинакова. Каждая ассемблерная команда состоит из нескольких полей.

Общая форма строки программы на языке ассемблер имеет вид:

{символ} {инструкция|директива|псевдо-инструкция} {;комментарий}

Все три поля строки опциональны, т. е. в зависимости от используемой инструкции или директивы могут опускаться.

Символ, как правило, представляет собой метку. В инструкциях и псевдоинструкциях это всегда метка. В некоторых директивах – символ для обозначе-

ния переменной или константы. Описание директивы проясняет это в каждом конкретном случае.

Символ должен начинаться в первом столбце. Он не может содержать символов пробела или табуляции, если *символ* не огорожен чертой (|). То есть символ «|*label one*|» допустим, в то время как «*label one*» – нет.

Для любого рода ссылок используются специальные имена, задаваемые программистом, которые называются *метками*. Метки – символические представления адресов. Их можно использовать, чтобы отметить конкретные адреса, к которым программист хочет обратиться из других частей кода.

Директивы содержат важную информацию для ассемблера, которая либо влияет на процесс сборки, либо на окончательный вид образа компилируемой программы.

Инструкции и *псевдоинструкции* составляют код, который процессор использует для выполнения задач.

Инструкциям, псевдоинструкциям и директивам должен предшествовать символ пробела или табуляции, независимо от того, есть предшествующие метки или нет.

Некоторые директивы не позволяют использовать метки.

Комментарий – заключительная часть строки программы. Разделителем поля комментария от инструкции служит символ «точка с запятой» (;). Первая «точка с запятой» в строке знаменует начало комментария, за исключением случаев, когда «точка с запятой» появляется внутри строковой переменной. Конец строки – конец комментария. На каждой строке допустим один комментарий. Компилятор игнорирует все комментарии. Можно использовать пустые строки, чтобы сделать код программы более читабельным.

В свою очередь, известно, что любая инструкция процессора состоит из двух полей – поля мнемокода операции и поля операндов. Таким образом, если рассматривать строку программы, реализующую произвольную инструкцию процессора, она состоит из следующих полей:

Поле меток	*Поле мнемоник*	*Поле операндов*	*Поле комментария*
LABEL	*MOV*	*R0, #10*	*; remark*

Каждое поле служит для определенных целей, поэтому мнемоника, например, не может быть записана в поле меток – это неминуемо приведет к ошибке. Рассмотрим назначение каждого из этих полей.

Как уже было указано, метки представляют собой символические обозначения некоторых участков в программе. Поле меток не обязательно должно заполняться в каждой строке программы. Метки расставляются только там, где в них действительно возникает необходимость. Их удобно использовать для организации условных и безусловных переходов, а также для объявления переменных и констант.

В ассемблере инструкции микропроцессора представляются в виде сокращений английских слов и такие аббревиатуры называются мнемониками или

мнемокодами. При наборе строк программы мнемоники команд должны располагаться обязательно в своем поле. При необходимости это поле так же, как и предыдущее, может быть пустым.

Если у команды имеются какие-либо операнды, они записываются в следующем и последнем поле строки – в поле операндов. Таким образом, полная команда может занимать в строке программы одно или два поля.

Поле комментария используется для записи комментария к программе.

При выборе имени метки необходимо соблюдать следующие правила:

– имя метки должно быть уникальным;

– имя должно состоять из одного слова, содержащего только латинские буквы и цифры;

– допускается также применять символ подчеркивания;

– первым символом метки обязательно должна быть буква или символ подчеркивания.

Имена меток чувствительны к регистру (строчные, прописные буквы) и все символы имеют значение. Имена меток не должны совпадать с именами встроенных переменных, регистров или предопределенных символов.

Мнемоники инструкций, псевдоинструкции, директивы и символические имена регистров можно записывать в верхнем или нижнем регистре, но не смешивать. Метки и комментарии могут быть в верхнем или нижнем регистре, или смешанном.

В ассемблерных файлах исходных кодов литералы могут быть выражены как:

– десятичные числа, например 123;

– шестнадцатеричные числа, например 0x7B;

– цифры в любой системе счисления от 2 до 9, например 8_204 – это число в восьмеричной системе счисления;

– числа с плавающей запятой, например 123,4;

– логические значения {TRUE} или {FALSE} (истина и ложь соответственно);

– отдельные значения символов, заключенных в одинарные кавычки, например 'W';

– строки, заключенные в двойные кавычки, например "This is a string".

Более подробные сведения о возможностях языка программирования ассемблер, доступном наборе директив и операторов выражений можно найти в методических указаниях по лабораторной работе № 4, электронная версия которых имеется в классе ПЭВМ кафедры (лаб. 207/2) по следующему адресу: D:\методические рекомендации\СТАНДАРТЫ РФ\Специальность 15.03.06 Мехатроника и робототехника\Микропроцессорная техника в мехатронике и робототехнике\МПТ ЛР 04.pdf, а также на сервере филиала кафедры по адресу \\Study\Методические указания\МПТ\МПТ ЛР 04.pdf.

5.2.2 Директивы языка программирования ассемблер и операторы выражения.

Язык программирования ассемблер всегда включает в себя машинные коды микроконтроллера, но этим не ограничивается набор команд данного языка.

Дело в том, что нужно уметь управлять самим процессом трансляции программы. Директивы выполняются ассемблером в процессе преобразования исходного модуля в объектный.

Макроассемблер armasm поддерживает следующие виды директив:

1) директивы определения символов;
2) директивы определения данных;
3) директивы управления ассемблером;
4) директивы определения фреймов;
5) директивы вывода сообщений;
6) директивы выбора набора команд и синтаксиса;
7) прочие директивы.

Рассмотрим наиболее часто используемые в программах директивы.

Обобщенная форма записи директив в программе имеет вид:

[<имя>] <дир> [<арг>],

где <имя> – поле имени;

<дир> – поле мнемокода директивы;

<арг> – поле аргументов.

В поле мнемокода записывается мнемоническое обозначение директивы языка; в поле имени – идентификатор, определяемый пользователем; в поле аргументов – один или несколько аргументов, которые определяют дополнительную информацию, требуемую для выполнения директивы.

Разделителем поля имени и поля мнемокода, а также поля мнемокода и поля аргументов является символ интервала ("пробел" или "горизонтальная табуляция"); разделителем аргументов – знак "запятая" или символ интервала в зависимости от вида директивы.

Оператор выражения – это конструкция языка, используемая для указания операций (над одним или двумя операндами), выполняемых ассемблером в процессе вычисления выражения. Операторы рассчитываются на этапе трансляции исходного текста программы для определения конкретного числа, которое используется в команде.

В языке имеются следующие виды выражений: числовые, логические, строковые.

5.3 Варианты индивидуальных заданий к лабораторной работе

Составить программу по заданию. Задания к лабораторной работе следует получить у преподавателя. При разработке программы необходимо придерживаться модульного стиля программирования, при этом программа и основная программа должны находиться в отдельных файлах. Операции по обработке данных нужно оформить в виде подпрограмм. При написании программы необходимо использовать директивы ассемблера.

Произвести отладку программы в среде µVision.

Составить отчет по выполненной работе.

Отчет должен содержать цель работы, постановку задачи, интерфейс программных модулей, спецификацию ресурсов контроллера (распределение памяти, использование регистров), алгоритм, текст отлаженной программы с комментариями, результаты работы программы, выводы по работе.

Контрольные вопросы

1 Перечислите и охарактеризуйте этапы разработки прикладной программы.

2 Какова структура строки программы, написанной на языке ассемблера, какие поля строки являются обязательными?

3 Укажите правила выбора имени метки.

4 Какие директивы используются для указания начала и конца программного модуля?

5 Перечислите директивы, используемые для выделения памяти.

6 Укажите функции следующих директив: EQU, ALIAS.

7 Укажите назначение директив DCD, DCW.

8 Какие типы операторов выражения используются в ассемблере?

9 Каким образом используется стек при выполнении подпрограмм?

10 Поясните механизм передачи параметров подпрограммы через память.

11 Поясните механизм передачи параметров подпрограммы через регистры общего назначения.

12 Поясните механизм передачи параметров подпрограммы через стек.

6 Лабораторная работа № 5. Изучение машинно ориентированного языка программирования

Цели работы: изучить правила программирования микроконтроллера на языке программирования С, порядок трансляции программы в машинные коды. По заданному алгоритму составить программу, откомпилировать и исследовать ход ее работы в отладчике µVision.

6.1 Расширение ANSI стандарта языка С для микроконтроллеров ARM

6.1.1 Общие сведения.

В состав интегрированной среды µVision входит компилятор языка С armcc. Компилятор armcc является оптимизирующим компилятором с языков С и С ++, который компилирует стандартный С и С ++ исходные коды в машинный код для процессоров, базирующихся на ARM-архитектуре. Компилятор armcc осуществляет компиляцию в соответствии с Базовым Стандартом Двоичного Интерфейса Приложений для ARM-архитектуры (Base Standard Application Binary Interface for the ARM Architecture (BSABI)) и генерирует выходные объекты в ELF-файл с поддержкой отладки в формате DWARF (Debug

With Arbitrary Record Format). Многие особенности компилятора разработаны, чтобы воспользоваться особенностями целевого процессора или архитектуры, для которого предназначен исходный код, именно поэтому при работе с компилятором знание целевого процессора или архитектуры полезно, а в некоторых случаях важно.

Программы в целом должны соответствовать ANSI-стандарту языка C. Компилятор позволяет скомпилировать исходный код, написанный с использованием ISO Standard C:1990, ISO Standard C:1999, ISO Standard C++:2003, в 32-битный ARM-код, 16/32-битный Thumb®-код, 16-битный Thumb-код.

Кроме этого, компилятор обеспечивает прямую поддержку для элементов архитектуры ARM-контроллеров, включающую расширения:

– ключевые слова и операторы функций;

– специальные атрибуты объектов и функций;

– специальные атрибуты переменных или полей структуры, функций и типов;

– ARM-специфические директивы;

– встроенные функции для реализации инструкции на машинном языке ARM из C или C ++ кода;

– доступ к регистрам процессоров, базирующихся на ARM-архитектуре.

Архитектура ARMv7-M процессоров Cortex-M3 поддерживает единое унифицированное адресное пространство. Все инструкции доступа к памяти могут быть использованы для доступа к любой ячейке во всех регионах памяти. В отличие, например, от компилятора языка C51, которому необходимы отдельные инструкции для доступа к различным областям памяти, это является гораздо более гибким решением, что позволяет свободно распределять код и данные по всем доступным областям памяти.

Подробные сведения о назначении ключевых слов, атрибутов объектов, функций, переменных можно найти в документе Compiler Reference Guide, входящем в пакет справочной системы интегрированной среды µVision, а также в методических указаниях по лабораторной работе № 5, электронная версия которых имеется в классе ПЭВМ кафедры (лаб. 207/2) по следующему адресу: D:\методические рекомендации\СТАНДАРТЫ РФ\Специальность 15.03.06 Мехатроника и робототехника\Микропроцессорная техника в мехатронике и робототехнике\МПТ ЛР 05.pdf, а также на сервере филиала кафедры по адресу \\Study\Методические указания\МПТ\МПТ ЛР 05.pdf.

6.1.2 Типы данных.

Компилятор поддерживает следующие типы данных (таблица 6.1).

Таблица 6.1 – Типы данных, поддерживаемые компилятором

Тип данных	Количество битов	Количество байт	Диапазон значений
signed char	8	1	–128...+127
unsigned char	8	1	0...255
enum	8	1	0...255
signed short	16	2	–32768…+32767
unsigned short	16	2	0...65535
signed int	32	4	–2147483648...2147483647
unsigned int	16	4	0…4294967295
signed long	32	4	–2147483648...2147483647
unsigned long	32	4	0…4294967295
long long	64	8	–9223372036854775808… 9223372036854775807
unsigned long long	64	8	0…18446744073709551615
float	32	4	±1,175494E-38...±3.402823E+38
double	64	8	±2.2250738585072014E-308… ±1,7976931348623158E+308
long double	64	8	
Все указатели	32	4	
Bool (только для C) [1]	8	1	
bool (только для C++)	8	1	
wchar_t (только для C++)	16	2	
Примечание – Должен быть использован stdbool.h для объявления макроса bool в C-файлах			

6.1.3 Описание функций.

При описании функций в программах для Cortex-M3 процессоров не требуются дополнительные указатели. Обработчики прерываний объявляются и программируются точно так же, как стандартные процедуры C. Аппаратное обеспечение, которое обрабатывает прерывания Cortex-M3, вместе с правилами использования регистров, указанными в стандарте вызова процедур ARM-архитектуры, автоматически гарантирует, что все необходимые регистры будут сохранены.

Поэтому общая форма описания функций соответствует стандартной для языка C/C++:

ret_type funcname(args)

где *ret_type* – тип возвращаемого функцией значения;

funcname – имя функции;

args – список аргументов.

6.1.4 Передача параметров функций.

Все функции в программе можно разделить на два класса: с переменным числом аргументов (variadic функции) и с фиксированным числом аргументов (nonvariadic функции). Для функций variadic и nonvariadic отличаются правила передачи параметров.

В функциях с фиксированным количеством значения аргументов параметров передаются следующим образом.

1 Первые целочисленные аргументы передаются через регистры R0…R3 в соответствующем порядке (первый параметр – через регистр R0 и т. д.).

2 Остальные параметры передаются через стек в порядке друг за другом.

Доступ к стеку является «дорогостоящим» по размеру кода и скорости исполнения, поэтому необходимо стараться использовать меньше пяти параметров функций, если это возможно.

Целочисленный параметр, размер которого больше чем 32 бита, например long long, может передаваться частично через регистр, а частично через стек. В этом случае часть, передающаяся через стек, загружается перед любыми числами с плавающей запятой, даже если это не соответствует порядку в списке параметров.

В функциях с переменным количеством аргументов значения параметров передаются через целочисленные регистры R0-R3 и через стек, если это необходимо. Порядок используемых слов, такой как если бы значения параметров были сохранены в последовательных словах памяти и затем помещены в:

– регистры R0…R3, R0 – первый;

– стек – младшие адреса в первую очередь (это означает, что они помещаются в стек в обратном порядке.)

6.2 Варианты индивидуальных заданий к лабораторной работе

Составить программу по заданию. Вариант задания получить у преподавателя. При разработке программы необходимо придерживаться модульного стиля программирования. Модули оформить в виде отдельного С-файла. Операции по обработке массивов оформить в виде функций в подключаемом С-файле.

Произвести отладку программы в среде µVision.

Составить отчет по выполненной работе.

Отчет должен содержать цель работы, постановку задачи, интерфейс программных модулей, спецификацию параметров и переменных, алгоритм, текст отлаженной программы с комментариями, результаты работы программы, выводы по работе.

Контрольные вопросы

1 Укажите достоинства и недостатки языка С.

2 Какие специальные атрибуты переменных или полей структур, функций и типов можно использовать в программах для контроллеров ARM?

3 Как размещать переменные по абсолютным адресам памяти?

4 Укажите особенности передачи параметров функций.

7 Лабораторная работа № 6. Изучение библиотеки драйверов для стандартных периферийных устройств Cortex-M3 контроллеров STM32F10x

Цель работы: изучить организацию библиотеки «STM32F10x Standard Peripherals Library».

7.1 Библиотека «STM32F10x Standard Peripherals Library»

7.1.1 Общие сведения.

Разработка прикладного программного обеспечения – достаточно трудоемкая задача. Разработка встраиваемого ПО, которое должно загружаться в целевой контроллер, – задача еще более сложная из-за того, что программисту необходимо работать с периферийными устройствами контроллера, загружать и читать информацию из его регистров. Регистры контроллера располагаются по определенным адресам и для успешной работы программисту приходится досконально изучить документацию на микроконтроллер. Работа с ARM-контроллерами в этом плане ничем не отличается – так же, как и у других контроллеров, периферийные устройства имеют соответствующие регистры статуса и управления. Однако разработчики контроллеров и фирма ARM – разработчик ядра – стараются упростить программисту задачу и разработали комплект библиотек, достаточно полно описывающих основные архитектурные особенности микроконтроллеров.

Так, например, компания STMicroelectronics для каждого из своих семейств ARM-контроллеров предоставляет библиотеку стандартных периферийных устройств Standard Peripherals Library. Ее можно свободно скачать с сайта www.st.com. Например, контроллер STM32F103R8 относится к классу устройств на основе процессора Cortex-M3, и библиотека для данного устройства носит название STM32F10x_StdPeriph_Lib_VX.Y.Z. Архив с библиотекой содержит:

– _htmresc – эта директория содержит ресурсы для всех html-файлов всего пакета;

– Libraries – эта директория содержит все файлы библиотек CMSIS и драйверов стандартных периферийных устройств STM32F10x (в дальнейшем Standard Peripherals Drivers);

– Project – эта директория содержит шаблоны проектов и примеры использования периферийных устройств контроллеров STM32F10x;

– Utilities – эта директория содержит исходные коды файлов, описывающих ресурсы демонстрационных плат STMicroelectronics, поддиректорию STM32_EVAL, в которой в иерархическом порядке приведены файлы исходных кодов, облегчающих написание программ для демонстрационных плат STM32100E_EVAL, STM32100B_EVAL, STM3210C_EVAL, STM3210E_EVAL, STM3210B_EVAL.

7.1.2 Библиотека CMSIS.

CMSIS – Cortex Microcontroller Software Interface Standard – Стандарт программного интерфейса микроконтроллеров с ядром Cortex. Представляет собой независимый от производителя уровень абстрагирования от аппаратных средств (в дальнейшем HAL – Hardware Abstraction Layer) для процессоров Cortex. CMSIS разрабатывается в тесном сотрудничестве с различными производителями полупроводниковых устройств и программного обеспечения и предоставляет общий подход для взаимодействия с периферийными устройствами, операционными системами реального времени и промежуточными компонентами.

CMSIS предоставляет простой программный интерфейс к процессору и его периферии, упрощающий и ускоряющий разработку ПО для новых устройств. CMSIS определяет следующие базовые функции:

– функции доступа к периферии ядра;

– встроенные (intrinsic) функции.

ARM обеспечивает как часть CMSIS несколько программных слоев, который доступен для различных реализаций компиляторов. Один из слоев предоставляет уровень доступа к периферийным устройствам ядра: содержит определения имен, определения адресов и вспомогательные функции для доступа к регистрам ядра и периферийным устройствам. Также он определяет независимый интерфейс для ОСРВ (операционных систем реального времени).

Программные слои могут быть расширены разработчиками микросхем.

Ядро CMSIS реализует базовую систему доступа к ядру процессора Cortex-M и его периферийным блокам. В этой библиотеке реализованы:

– HAL для регистров процессора Cortex-M со стандартизированными объявлениями для таймера SysTick, контроллера прерываний NVIC, регистров управления процессором, регистров блока защиты памяти MPU, регистров блока расчетов с плавающей запятой FPU, а также функции доступа к ядру;

– имена системных исключений (прерываний), предоставляющие интерфейс к системе прерываний контроллера без необходимости устранения проблем совместимости;

– методы организации заголовочных файлов, которые облегчают изучение новых контроллеров Cortex-M и улучшают портируемость приложений, включая правила объявления имен специфических для устройства прерываний;

– методы инициализации системы, используемые каждым производителем микросхем. Например, стандартизированная функция SystemInit(), необходимая для настройки тактового генератора микроконтроллера;

– встроенные функции, используемые для генерации инструкций микропроцессора, неподдерживаемых стандартными функциями языка С;

– переменная, определяющая системную тактовую частоту, которая упрощает настройку системного таймера SysTick.

Для того чтобы использовать CMSIS, нужно добавить следующие файлы в разрабатываемое встраиваемое приложение:

1) файл со стартовым кодом, имя вида «startup_<device>.s;

2) файлы конфигурации системы с названиями вида «system_<device>.c» и «system_<device>.h»;

3) заголовочный файл устройства вида «<device.h>», предоставляющий доступ к ядру процессора и его периферии.

Примечание – Файлы startup_<device>.s, system_<device>.c и system_<device>.h могут потребовать некоторой специфической адаптации и поэтому они должны быть скопированы в директорию проекта. Файл <device.h> должен включаться во все файлы, где требуется доступ к микроконтроллеру, и может храниться в одной из директорий, являющейся общей для всех проектов.

7.1.3 Библиотека Standard Peripherals Drivers.

Данная библиотека является надстройкой над CMSIS, которая представляет набор драйверов для каждого периферийного модуля, входящего в состав контроллеров семейства. Каждый драйвер состоит из набора функций, реализующих все возможные функции периферийного устройства. Разработка каждого драйвера выполняется в соответствии с общим API, который стандартизирует структуру драйвера, функции и названия параметров. Это позволяет легко переносить разработанные приложения на другие контроллеры семейства, а также изменять исходные коды самих драйверов.

Библиотека реализована с обнаружением ошибок в процессе выполнения приложения путем проверки входных величин для всех функций библиотеки. Такая динамическая проверка повышает надежность приложения.

Драйвер каждого периферийного устройства реализован в виде двух файлов: исходного кода stm32f10x_ppp.c и заголовочного файла stm32f10x_ppp.h. Запись ppp в имени файла обозначает, что этот файл для периферийного устройства ppp. Таким образом, например, драйвер АЦП будет иметь обозначение stm32f10x_adc, а драйвер порта общего назначения – stm32f10x_gpio. При этом для удобства разработки ПО каждый драйвер содержит набор функций, имеющих в названии наименование модуля. Например, для АЦП могут быть использованы функции ADC_Cmd, ADC_Init, а драйвер параллельного порта – GPIO_Init, GPIO_ReadInputData. Вторая часть названия функции указывает на выполняемое ею действие.

Более подробные сведения о составе библиотеки, принципах и примере ее использования в программе можно найти в методических указаниях по лабораторной работе № 6, электронная версия которых имеется в классе ПЭВМ кафедры (лаб. 207/2) по следующему адресу: D:\методические рекомендации\СТАНДАРТЫ РФ\Специальность 15.03.06 Мехатроника и робототехника\Микропроцессорная техника в мехатронике и робототехнике\МПТ ЛР 06.pdf, а также на сервере филиала кафедры по адресу \\Study\Методические указания\МПТ\МПТ ЛР 06.pdf.

7.2 Варианты индивидуальных заданий к лабораторной работе

Разработать программу для учебного стенда, позволяющую, используя систему прерываний, обрабатывать события (включение/выключение) на ДД, вы-

водить поясняющие надписи и результат вычислений на МЖКИ или ДСИ. Вариант задания получить у преподавателя.

Контрольные вопросы

1 Назовите назначение библиотеки «STM32F10x Standard Peripherals Library».

2 Перечислите состав библиотеки.

3 Перечислите драйверы периферийных устройств, входящие в состав библиотеки.

8 Лабораторная работа № 7. Изучение параллельных портов контроллеров STM32F10x

Цель работы: изучить организацию и принципы работы с параллельными портами микроконтроллеров STM32F10x.

8.1 Параллельные порты ввода/вывода информации микроконтроллера STM32F103x

8.1.1 Общие сведения.

Порты контроллеров данного семейства представляют собой 16-разрядные устройства ввода/вывода информации. Основное назначение – ввод/вывод дискретных сигналов в микроконтроллер, однако большинство линий портов можно использовать и для других целей. Каждый порт содержит управляемые регистры-защёлки входа и выхода, входной буфер и выходной драйвер. Структурная схема одной линии порта приведена на рисунке 8.1.

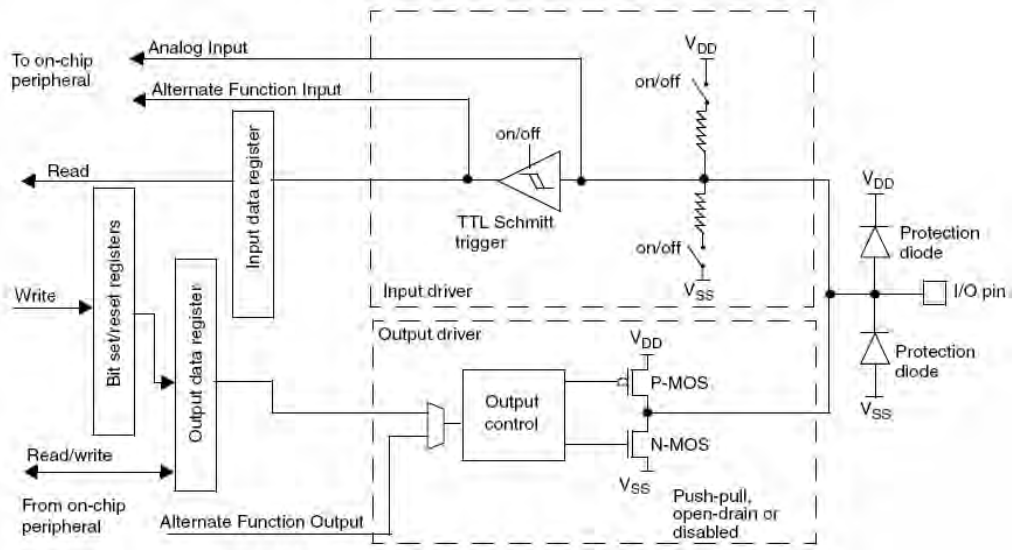


Рисунок 8.1 – Базовая структура стандартного вывода параллельного порта

Как видно, каждый вывод порта включает в свой состав входной и выходной драйверы (Input driver, Output driver), регистр входных данных (Input data register), а также регистры управления выходом порта: установки/сброса битов (Bit set/reset registers) и выходных данных (Output data register). Каждая линия порта имеет на выходе защитные диоды (Protection diode).

Как понятно из названия:

– регистр входных данных позволяет прочитать состояние порта;

– регистры выходных данных управляют состоянием порта;

– входной драйвер осуществляет ввод данных с линии порта;

– выходной драйвер осуществляет изменение состояния линии порта.

Входной драйвер состоит из управляемого триггера Шмитта и подтягивающих резисторов, которые можно подключить к линии через ключи.

Выходной драйвер состоит из устройства управления (Output control) и выходного каскада, представленного двумя полевыми транзисторами, которые могут находиться в трех состояниях.

Порты контроллера могут работать в разных режимах:

– плавающий (floating) вход (находящийся в высокоимпедансном состоянии);

– вход с подтяжкой вверх;

– вход с подтяжкой вниз;

– аналоговый вход (для каналов АЦП);

– выход с открытым коллектором (стоком, если быть точным);

– push-pull (тяни-толкай) – выход;

– альтернативные функции, т. е. работа от периферии.

Каждый порт содержит два 32-битных регистра конфигурации (GPIOx_CRL, GPIOx_CRH), два 32-битных регистра данных (GPIOx_IDR, GPIOx_ODR), 32-битный регистр установки/сброса (GPIOx_BSRR), 16-битный регистр сброса (GPIOx_BRR) и 32-битный регистр блокировки (GPIOx_LCKR).

Каждый вывод порта свободно настраивается, однако регистры доступны только как 32-битные слова (побайтный доступ или доступ по полслова не возможны). Регистры GPIOx_BSRR и GPIOx_BRR предоставляют атомарный доступ на чтение/модификацию к каждому выходу порта.

В таблице 8.1 приведены возможные варианты конфигурации выводов порта.

Для работы с портом необходимо выполнить следующие действия:

1) включить тактирование порта;

2) настроить выводы порта как входы/выходы;

3) произвести настройку линий порта (установить скорость, тип и подтяжку по выходу);

4) установить/прочитать значение порта.

Таблица 8.1 – Конфигурация вывода порта

Режим конфигурации		CNF[1:0]	MODE[1:0]	Регистр PxODR
Выход общего назначения	Push-pull	00	01 10 11	0 или 1
	Открытый коллектор	01		0 или 1
Выход альтернативной функции	Push-pull	10		*
	Открытый коллектор	11		*
Вход	Аналоговый	00	00	*
	«Плавающий» вход	01		*
	Вход подтянут вниз	10		0
	Вход подтянут вверх			1
Примечание – * – не имеет значения				

8.1.2 Настройка, чтение и запись информации в порт в программе.

Как было указано в лабораторной работе № 6, регистры, периферийные устройства контроллера описываются при помощи библиотечных файлов. Использование портов может быть реализовано двумя способами: при помощи CMSIS, при помощи библиотеки стандартных периферийных устройств.

Чтобы применить CMSIS, необходимо воспользоваться макросами, описывающими периферию контроллера, из файла stm32f10x.h.

Так, для включения тактового генератора можно воспользоваться следующим кодом в программе:

RCC->APB2ENR |= RCC_APB2ENR_IOPCEN; /* для порта C */

Выключение генератора можно осуществить кодом

RCC->APB2ENR &= ~(RCC_APB2ENR_IOPCEN); /* для порта C */

Настроить линию 1 порта C как push -pull -выход с максимальной выходной частотой 10МГц можно следующим кодом:

GPIOC->CRL &= 0xFFFFFF0F; /* сброс предыдущей настройки: MODE[1:0]=002, CNF[1:0]=002 */

GPIOC->CRL |= GPIO_CRL_MODE1_0; /* установка новой: MODE[1:0]=012 */

Установить/сбросить линию 1 порта B можно так:

GPIOB->BSRR = GPIO_BSRR_BS_1; /* установка */

GPIOB->BSRR = GPIO_BSRR_BR_1; /* сброс */

Состояние линии 2 порта B можно узнать при помощи следующего фрагмента кода:

int PB2 = ((GPIOB->IDR & GPIO_IDR_IDR_2)!=0);

Объявления макросов, определяющих биты различных регистров, можно найти в файле stm32f10x.h.

Для работы с портами с использованием драйверов периферийных необходимо применить функции из файлов stm32f10x_gpio.c и stm32f10x_rcc.c.

Более подробные сведения о возможностях и режимах работы портов STM32F103, настройке и использовании их в программе можно найти в методических указаниях по лабораторной работе № 7, электронная версия которых имеется в классе ПЭВМ кафедры (лаб. 207/2) по следующему адресу:

D:\методические рекомендации\СТАНДАРТЫ РФ\Специальность 15.03.06 Мехатроника и робототехника\Микропроцессорная техника в мехатронике и робототехнике\МПТ ЛР 07.pdf, а также на сервере филиала кафедры по адресу \\Study\Методические указания\МПТ\МПТ ЛР 07.pdf.

8.2 Варианты индивидуальных заданий к лабораторной работе

Разработать программу, позволяющую выполнять действия в соответствии с заданием. Вариант задания получить у преподавателя.

Произвести отладку программы в среде µVision.

Составить отчет по выполненной работе.

Отчет должен содержать цель работы, постановку задачи, интерфейс программных модулей, текст отлаженной программы с комментариями, результаты работы программы, выводы по работе.

Контрольные вопросы

1 Перечислите элементы, входящие в состав порта.
2 В каких режимах могут работать выводы порта?
3 Перечислите регистры, используемые для настройки линий порта.
4 Поясните работу порта в режиме входа.
5 Поясните работу порта в режиме выхода.
6 Поясните работу порта в режиме альтернативной функции.
7 Поясните работу порта в режиме аналогового входа.

Список литературы

1 **Джозеф, Ю.** Ядро Cortex-M3 компании ARM. Полное руководство / Ю. Джозеф, – Москва : Додэка XXI, 2012. – 552 с.

2 The Insider’s Guide To The STM32 ARM®Based Microcontroller – An Engineer’s Introduction To The STM32 Series [Электронный ресурс]. – Режим доступа : http://www2.hitex.com/download-isg. – Дата доступа : 01.05.2017.

3 Cortex-M3 Technical reference manual [Электронный ресурс]. – Режим доступа : http://infocenter.arm.com. – Дата доступа : 01.05.2017.

4 ARMv7-M architectural reference manual [Электронный ресурс]. – Режим доступа : http://infocenter.arm.com. – Дата доступа : 01.05.2017.

5 ARM Architectural reference manual Thumb2 supplement [Электронный ресурс]. – Режим доступа : http://infocenter.arm.com. – Дата доступа : 01.05.2017.

6 STM32F103xx User Manual ST Microelectronics [Электронный ресурс]. – Режим доступа : http://st.com. – Дата доступа : 01.05.2017.

7 STM32F10xxx FLASH Programming manual ST Microelectronics [Электронный ресурс]. – Режим доступа : http://st.com. – Дата доступа : 01.05.2017.

8 **Недяк, С. П.** Лабораторный практикум по микроконтроллерам семейства Cortex-M: методическое пособие по проведению работ / С. П. Недяк, Ю. Б. Шаропин. – Томск: ТУСУР, 2013. – 76 с.