ГОСУДАРСТВЕННОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Электропривод и автоматизация промышленных установок»

ОСНОВЫ МИКРОПРОЦЕССОРНОЙ ТЕХНИКИ

Методические рекомендации к лабораторной работе для студентов специальности 1-53 01 05 «Автоматизированные электроприводы» дневной и заочной форм обучения

Могилев 2017

УДК 004.3:62-83
ББК 32.973.26-04:31.291
О 75

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой ЭП и АПУ «06» февраля 2017 г., протокол № 7

Составитель ст. преподаватель В. Н. Ситников

Рецензент канд. техн. наук Ф. М. Трухачев

Методические рекомендации предназначены для студентов специальности 1-53 01 05 «Автоматизированные электроприводы» дневной и заочной форм обучения.

Учебно-методическое издание

ОСНОВЫ МИКРОПРОЦЕССОРНОЙ ТЕХНИКИ

Ответственный за выпуск	Г. С. Леневский
Технический редактор	А. А. Подошевко
Компьютерная верстка	Е. С. Лустенкова

Подписано в печать . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс.
Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 115 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 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 необходимо ответить на контрольные вопросы.

1.1 Общие сведения о системе команд

1.1.1 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.

Набор Thumb, имея меньшую разрядность, оптимизирован, в его состав ввели только команды, реализующие наиболее важные операции, и он несколько ограничивает доступ к ресурсам процессора. Например, с 16 до 8 изменяется число полностью доступных регистров. Некоторые команды из набора Thumb предоставляют ограниченный доступ к регистрам R8–R15, называемых в этом режиме «верхними», или «старшими», регистрами (high registers). Указатель стека SP (R13), регистр адреса возврата LR (R14), счетчик команд PC (R15) доступны в Thumb через специальные команды, прямой доступ к словам состояния CPRS и SPRS не предоставляется.

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 предусмотрено несколько улучшенных инструкций умножения, исполняющихся за один цикл, и аппаратный делитель, требующий от 2 до 7 циклов.

1.1.2 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 (два байта), 32-битные (четыре 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. В таблицах А.1–А.7 при-

ведены ассемблерная мнемоника, описание команд и их характеристики.

1.1.3 Основы синтаксиса языка ассемблера.

В ассемблерных программах при записи команд обычно используется следующее форматирование:

<метка>
<код операции> операнд1, операнд2, ... ; Комментарий

Элемент <метка> является необязательным. Метки могут размещаться перед некоторыми командами для того, чтобы с их помощью можно было определять адреса этих команд. После необязательной метки располагается <код операции> (команда), сопровождаемый некоторым количеством операндов. Операндом команды может быть регистр, константа или же другой параметр, специфичный для конкретной команды. Команда выполняет требуемые действия над операндами и во многих случаях сохраняет результат в регистр-приёмник. При наличии в команде такого регистра он обычно указывается перед операндами. Поэтому, как правило, первый операнд является приёмником результата операции. Число операндов зависит от типа команды, формат записи операндов также может быть различным. Константы обычно записываются в виде #число, как показано в следующем примере:

```
     MOV  R0,  #0x12    ;  Загрузить  в  R0  число  0x12
(шестнадцатеричная константа)
     MOV R1, #'A'    ; Загрузить в R1 символ ASCII 'A'
```

Текст после символа «;» является комментарием. Эти комментарии не влияют на выполнение программы, однако позволяют сделать текст программы более понятным для человека.

1.2 Типы операндов и способы адресации данных

1.2.1 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.

1.2.2 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 используемых в командах процессора и используется для доступа к регистрам общего назначения (рисунок 1, б). При регистровой адресации в команде задается номер регистра Rn, который является источником или приёмником результата операции. Например:

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

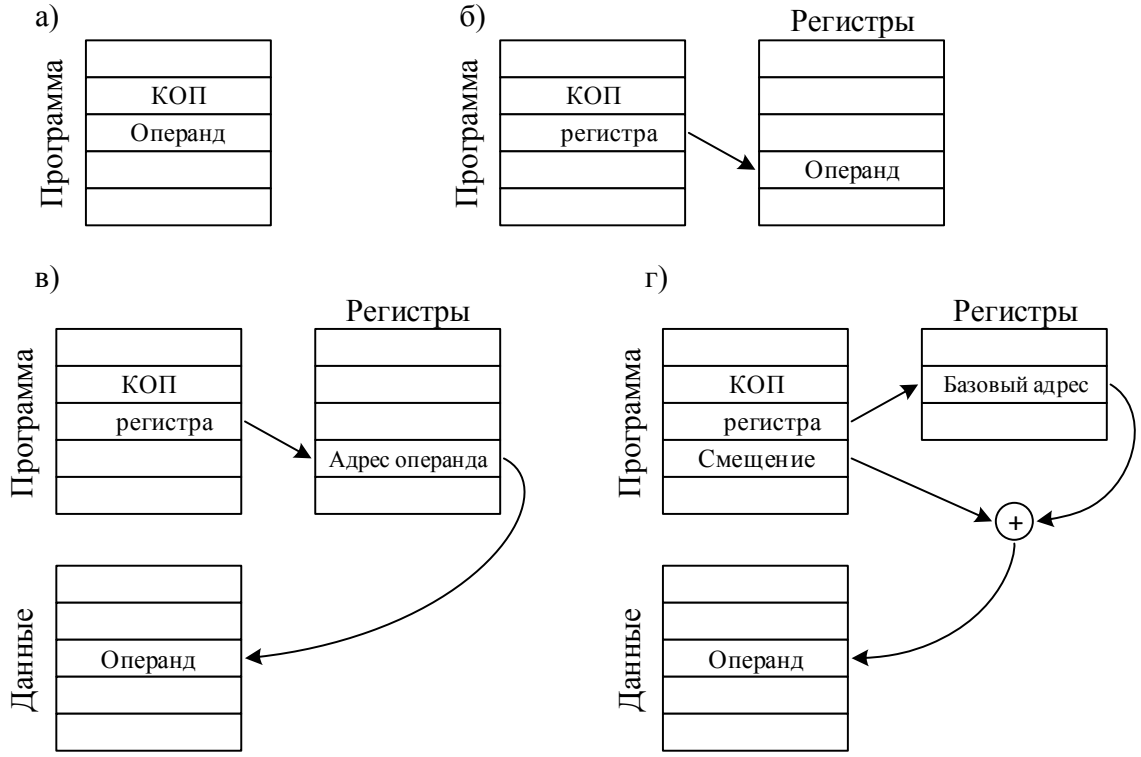
Косвенно-регистровая адресация используется для обращения к ячейкам памяти. При этом способе адресации в команде записан номер регистра, который выступает в качестве регистра-указателя адреса. Его значение является адресом искомой ячейки памяти (рисунок 1, в). Например:

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

При непосредственной адресации используемые операнды явно указаны в команде (рисунок 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 1, г). Базовый регистр может быть любым из регистров общего назначения. При загрузке базовым регистром может быть счетчик команд PC.



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

Рисунок 1 – Адресация данных в командах ЦП Cortex-M3

Смещение может быть одного из трех форматов:

1) непосредственное – смещение в данном случае представляет собой беззнаковое число, которое может быть добавлено или вычтено из значения базового регистра. Непосредственное смещение используется для доступа к элементам данных, которые находятся на фиксированном расстоянии от начала объекта данных, таких как поля структуры, стек и регистры ввода/вывода;

2) регистр – смещение представляет собой значение регистра общего назначения. Этот регистр не может быть счетчиком команд PC. Значение может быть добавлено к значению базового регистра или вычтено из него. Регистровые смещения могут быть использованы для доступа к мас-

сивам или блокам данных;

3) масштабируемый регистр – в данном случае смещение – регистр общего назначения, кроме счетчика команд PC, сдвигаемый на непосредственное значение, который затем добавляется или вычитается из базового регистра. Это означает, что индекс массива может быть масштабирован на размер каждого элемента массива.

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 преиндексированием
```

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 $0 \le shift \le 3$.

1.2.3 «Гибкий» второй операнд. 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

Rт {, 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, $1 \le n \le 32$.

LSL #n Логический сдвиг влево на n битов, $1 \le n \le 31$.

LSR #n Логический сдвиг вправо на n битов, $1 \le n \le 32$.

ROR #n Циклический сдвиг вправо на n битов, $1 \le n \le 31$.

RRX Расширенный циклический сдвиг вправо на 1 бит.

Например:

```
ADD R1,R1,R0,LSL #1    ;В первом и втором операн-
дах 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.

1.3 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 слу-

чае, 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 выполнения 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 1.

Таблица 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 ноль
VS	V = 1	Переполнение
VC	V = 0	Нет переполнения
HI	C = 1 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 = 1 и N ≠ V	Меньше или равно (знаковое «≤»)
AL	Безразлично	Всегда; действие по умолчанию при отсутствии суффикса

Для уменьшения числа команд переходов в коде программы можно использовать условное выполнение команды. Пример условного выполнения:

```
CMP R0, R1    ; Сравниваем R0 и R1
ITT GT        ; IT - пропускаем две следующие команды,
              ; если условие GT не выполняется
CMPGT R2, R3  ; Если «больше», сравниваем R2 и R3,
              ; устанавливаем флаги
MOVGT R4, R5  ; Если всё ещё «больше»,
              ; копируем R4 = R5
MOVS R0, R1   ; R0 = R1, изменяет состояние флагов
```

1.4 Выбор разрядности команды

Как было отмечено, набор инструкций Thumb-2 является смесью 16- и 32-битных инструкций. Поэтому многие команды могут формировать как 16-, так и 32-битный машинный код в зависимости от используемых операндов и регистра-приёмника. Для некоторых из этих команд можно явно задать разрядность с помощью специального суффикса. Суффикс

«.W» указывает на необходимость генерации 32-битного кода, а суффикс «.N» – 16-битного кода.

Если ассемблер не сможет сформировать машинный код заданной разрядности, то он выдаст сообщение об ошибке.

1.5 Команды пересылки данных

Одной из основных функций процессора является пересылка данных. Процессор Cortex-M3 поддерживает следующие типы пересылок:

– пересылка данных между регистрами общего назначения;

– пересылка непосредственного значения в регистр общего назначения;

– пересылка данных между памятью и регистром общего назначения;

– пересылка данных между регистром специального назначения и регистром общего назначения.

Для пересылки данных между регистрами предназначена команда MOV. Например, пересылка содержимого регистра R3 в регистр R8 записывается следующим образом:

```
MOV R8, R3
```

Имеется ещё одна команда пересылки, которая пересылает инверсное значение исходного содержимого регистра, это команда MVN.

Очень часто возникает необходимость загрузки в регистр непосредственного значения – константы. Для загрузки значений разрядностью 8 бит и менее можно воспользоваться командой MOVS. Например,

```
MOVS R0, #0x12
```

Для загрузки значений большей разрядности (более 8 бит) следует использовать команду пересылки из набора Thumb-2. Например,

```
MOVW.W R0, #0x789A
```

В случае 32-битной константы можно использовать две команды для загрузки старшей и младшей частей значения. Так, например, для загрузки в регистр R0 константы 0x3456789A можно воспользоваться следующими командами:

```
MOVW.W R0,#0x789A
MOVT.W R0,#0x3456
```

Основными командами, посредством которых выполняются обращения к памяти, являются команды загрузки и сохранения (таблица А.1). Команда загрузки (LDR) пересылает данные из памяти в регистры процессора, а команда сохранения (STR) пересылает содержимое регистров в память. При этом поддерживается пересылка данных любого размера: байт

(LDRB, STRB), полуслово (LDRH, STRH), слово (LDR, STR), двойное слово (LDRD, STRD). Кроме того, необходимо отметить, что данные команды используют только косвенно-регистровую и индексную адресацию данных для указания адреса ячейки памяти, к которой происходит обращение (см. п. 1.2.2). Как видно из таблицы А.1, регистр Rn хранит адрес ячейки памяти, к которой будет обращаться процессор при выполнении команды, а в случае наличия смещения «offset» содержимое выбранного регистра является базовым значением, относительно которого адрес будет вычисляться. Таким образом, для того, чтобы инструкция обратилась к требуемой ячейке памяти (для записи или чтения), ее адрес должен быть предварительно загружен в соответствующий регистр.

Также процессоры ARM поддерживают обращение к памяти с использованием предындексации и постиндексации. В случае предындексации содержимое регистра, в котором хранится адрес ячейки памяти, перед выполнением операции изменяется. То есть пересылка данных будет осуществлена уже по новому адресу. Например:

```
LDR.W R0, [R1, #offset]! ; Читаем из памяти по адресу [R1+offset], после операции R1 становится равным R1+offset
```

Наличие символа «!» указывает на изменение базового регистра Rl. Этот символ является необязательным, при его отсутствии команда выполнит обычную операцию пересылки из ячейки памяти, определяемой смещением относительно базового адреса.

Команды обращения к памяти с постиндексацией осуществляют пересылку данных с использованием базового адреса, определяемого регистром, после чего изменяют содержимое этого регистра. Например,

```
LDR.W R0, [R1], #offset ; Читаем из памяти по адресу [R1], после операции R1 становится равным R1+offset
```

При применении команд с постиндексацией нет необходимости в использовании знака «!», поскольку команды данного типа в любом случае изменяют регистр, содержащий базовый адрес.

В группе команд обращения к памяти с предындексацией и постиндексацией имеются команды загрузки и сохранения данных различной разрядности: байта, полуслова и слова.

Для демонстрации рассмотрим карту памяти, приведенную на рисунке 2. Как видно на рисунке, индексный способ адресации используется в рассматриваемых командах, смещение задано непосредственно и равно 5 для первых трех команд и 4 для четвертой. Таким образом, во всех командах адрес ячеек, к которым будет производиться обращение, сформируется путем сложения содержимого регистра R2 и 1, т. е. будет равен

0x20000005 (0x20000004).

Как видно из таблицы А.1, первая из рассматриваемых команд читает слово начиная с указанной ранее ячейки, в результате рассматриваемых команд в регистр R0 будут помещены 4 байта, формирующие слово 0x0A433221.

Адрес	Данные
0x20000004	0x10
0x20000005	0x21
0x20000006	0x32
0x20000007	0x43
0x20000008	0x0A
0x20000009	0xFD
0x2000000A	0x3C
0x2000000B	0xB7
0x2000000C	0x1E
0x20000009	0x64

Пусть перед выполнением следующих команд R2 = 0x20000000

1) LDR R0,[R2,#5]

2) LDRH R0,[R2,#5]

3) LDRB R0,[R2,#5]

4) LDRD R0,R1,[R2,#4]

Рисунок 2 – Адресация данных в командах ЦП Cortex-M3

Вторая команда читает полуслово по аналогичному адресу, в результате чего в регистре R0 окажется число 0x00003221.

В результате выполнения третьей команды в регистр R0 будет помещен один байт 0x00000021.

Последняя из рассматриваемых команд читает двойное слово, т. е. 8 байт, помещая результат в пару регистров R0 и R1, в результате чего в них окажутся числа 0x43322110 и 0xB73CFD0A.

Команды групповой загрузки (LDM) и сохранения (STM) дают возможность за один раз выполнить несколько одноимённых операций. При этом символ «!» после первого операнда указывает на необходимость обновления регистра Rd после выполнения команды. Пусть в R8 содержится значение 0x8000, тогда

```
STMIA.W R8!, {R0-R3}   ;После сохранения данных
R8 станет равным 0x8010 (инкрементируется на 4 для
каждой операции пересылки)

STMIA.W R8 , {R0-R3}   ;R8 не изменится
```

При этом групповую загрузку можно выполнять двумя способами – на это указывает способ адресации в виде суффикса IA (Increment After) и DB (Decrement Before) – с инкрементированием и декрементированием регистра после каждой пересылки соответственно.

К классу операций с памятью относятся также операция загрузки данных в стек и операция извлечения данных из стека. Для выполнения этих операций предназначены команды PUSH и POP соответственно. Например,

```
PUSH {R0, R4-R7, R9} ; Загружаем R0, R4, R5, R6, R7, R9 в стек
POP {R2,R3} ; Извлекаем R2 и R3 из стека
```

Для обращения к регистрам специального назначения (APSR, IPSR, EPSR, IEPSR, IAPSR, EAPSR, PSR, MSP, PSP, PRIMASK, BASEPRI, BASEPRI_MAX, FAULTMASK или CONTROL) используются команды MRS и MSR. Команда MRS позволяет считать значение регистра, MSR – установить его. Следует отметить, что обращение ко всем регистрам специального назначения, кроме регистра APSR, допускается только в привилегированном режиме. Пример использования команды:

```
MRS R0, PSR ; Считать состояние процессора в R0
```

1.6 Команды обработки данных

Процессор Cortex-M3 поддерживает множество самых разных команд, предназначенных для обработки данных (см. таблицы А.2 и А.3). Среди них присутствуют инструкции, позволяющие выполнять основные арифметические и логические операции над данными, осуществлять перестановку байтов содержимого регистра и др. Большинство команд обработки данных могут иметь несколько форматов записи. Например, их можно выполнять над двумя регистрами, регистром и константой и т. д.

Необходимо отметить, что результат обработки информации помещается в регистр общего назначения, который указан первым при записи команды. Причем этот регистр в некоторых командах может также являться и одним из источников обрабатываемой информации. Для таких команд в таблицах регистр-приемник Rd отмечен как необязательный.

1.6.1 Команды арифметических операций позволяют выполнять основные арифметические действия над данными: сложение, вычитание, умножение и деление. Команды ADD и ADDC допускают сложение любого регистра с большим числом операндов. Аналогично командам ADD существуют команды SUBB и SBC, позволяющие достаточно просто производить вычитание 32-битных двоичных чисел с учетом или без учета заёма. Команда RSB производит обратное вычитание, когда из второго операнда вычитается значение первого, в отличие от остальных команд. Команда MUL позволяет перемножить между собой значения двух регистров общего назначения. Результат, получаемый после выполнения операции,

помещается в один из регистров. Команды UDIV и SDIV позволяют выполнить целочисленное (остаток от деления отбрасывается) деление беззнаковое и знаковое соответственно. При выполнении знакового деления обрабатываемые отрицательные числа должны быть представлены в дополнительном коде. В результате использования данных команд будут получены 32-разрядные числа.

Эти команды могут применяться как с суффиксом S, указывающим на то, что команда воздействует на регистр APSR, так и без данного суффикса. В большинстве случаев при использовании синтаксиса UAL и отсутствии в мнемонике команды суффикса S в код будет вставлен 32-битный вариант команды, поскольку почти все 16-битные команды Thumb влияют на состояние регистра APSR.

Кроме того, в системе команд есть команды UMULL (беззнаковое длинное умножение), UMLAL (беззнаковое длинное умножение со сложением), SMULL (знаковое длинное умножение) и SMLAL (знаковое длинное умножение со сложением). В результате этих операций перемножаются два 32-битных числа и получается 64-битный результат, сохраняющийся в паре регистров.

1.6.2 Команды логических операций содержат инструкции, реализующие побитовые логические операции над словами: «И» (AND), «ИЛИ» (ORR), «ИСКЛЮЧАЮЩЕЕ ИЛИ» (EOR), «ИЛИ с инверсией» (ORN), «И с инверсией» (или очистка битов BIC). Выполняться эти операции могут либо над двумя регистрами, либо над регистром и непосредственным операндом (константой).

Эти команды тоже могут применяться с суффиксом S, указывающим на необходимость изменения регистра APSR. При использовании синтаксиса UAL и отсутствии в записи команды суффикса S в код будет вставлен 32-битный вариант команды.

В процессоре Cortex-M3 имеются команды простого и циклического сдвига (рисунок 3), которые также можно отнести к классу логических. В некоторых случаях операция циклического сдвига может объединяться с другими операциями, например, при вычислении смещения адреса для команд загрузки/сохранения. При использовании синтаксиса UAL и отсутствии в мнемонике команды суффикса S в код вставляется 32-битный вариант команды.

В случае использования команд LSLS, LSRS, ASRS, RORS или одной из команд LSL #n, LSR #n, ASR #n, ROR #n во втором операнде команд MOVS, MVNS, ANDS, ORRS, ORNS, EORS, BICS, TEQ или TST во флаг переноса загружается последний перемещённый бит регистра Rm.

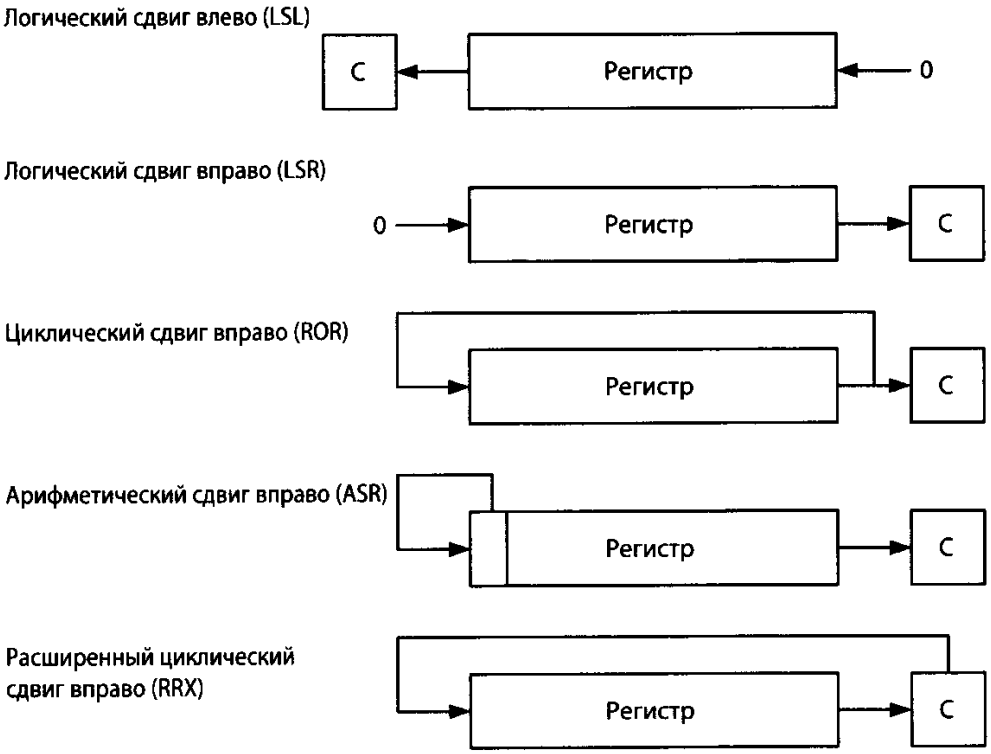


Рисунок 3 – Команды циклического и обычного сдвигов

Команда RRX перемещает все биты регистра Rm на одну позицию вправо и копирует содержимое флага переноса в 31-м бите результата. В случае использования команды RRXS или при использовании команды RRX во втором операнде команд MOVS, MVNS, ANDS, ORRS, ORNS, EORS, BICS, TEQ или TST во флаг переноса загружается 0-й бит регистра Rm.

Операцию ASR #n можно использовать для деления содержимого регистра Rm на 2^n с округлением результата к ближайшему меньшему целому. Операцию LSR #n можно использовать для деления содержимого регистра Rm на 2^n, если содержимое регистра рассматривается как целое число без знака. Операцию LSL #n можно использовать для умножения содержимого регистра Rm на 2^n , если содержимое регистра рассматривается как целое число без знака или как целое число со знаком, представленное в дополнительном коде.

Если необходимо произвести операцию циклического сдвига влево, ее можно заменить операцией циклического сдвига вправо с другим значением сдвига. Так, операция циклического сдвига на 4 бита влево может быть выполнена с помощью команды циклического сдвига на 28 бит вправо. Обе команды приводят к одинаковому результату и выполняются за одно и то же время.

1.6.3 Команды перестановки байтов позволяют изменить форму представления числа, сохраненного в регистре. Эти команды обычно ис-

пользуются для преобразования данных с прямым порядком байтов в данные с обратным порядком байтов и наоборот (рисунок 4), т. е. позволяют изменить форму представления чисел с little-endian на big-endian и обратно как для слов, так и для полуслов. Имеются как 16-битный, так и 32-битный варианты этих команд, причём 16-битные команды могут обращаться только к младшим регистрам.

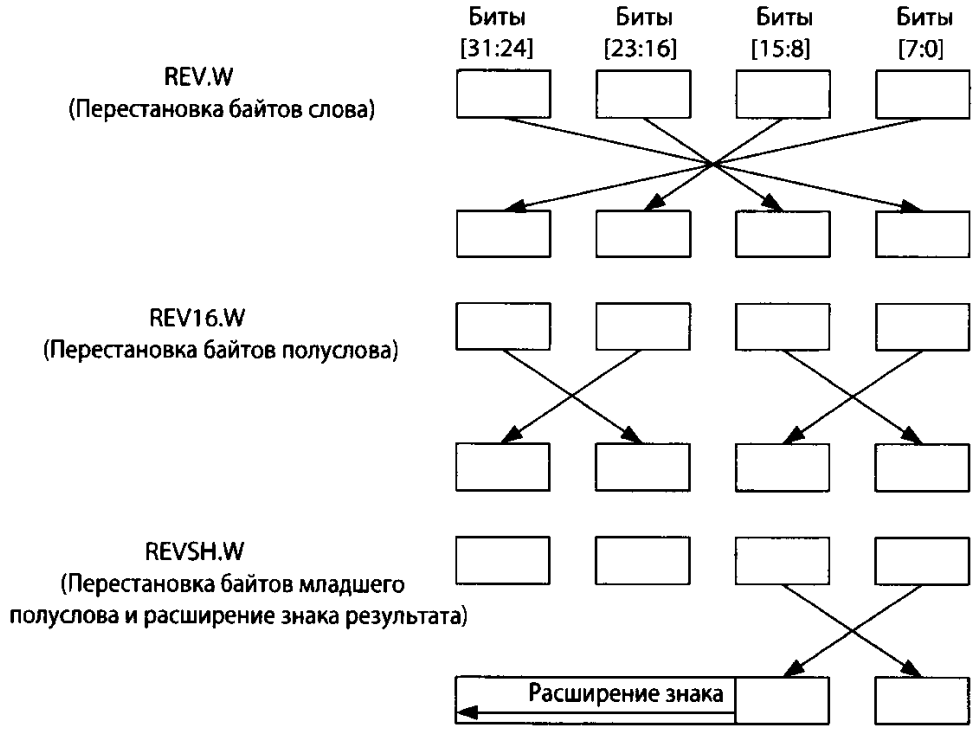


Рисунок 4 – Команды перестановки байтов

1.6.4 Еще одну группу команд обработки данных составляют команды сравнения и проверки, которые могут использоваться для организации ветвлений в программах. Команды сравнения CMP и CMN аналогичны по действию командам вычитания SUBBS и сложения ADDS соответственно, за исключением того, что они влияют только на состояние флагов результатов выполнения операций, не сохраняя при этом результат операции. Команды проверки TST и TEQ по действию аналогичны логическим командам ANDS и EORS соответственно, т. е. изменяют флаги, но не сохраняют нигде результат операции.

1.7 Операции насыщения

Процессор Cortex-M3 поддерживает две команды, обеспечивающие выполнение операции насыщения для знаковых (SSAT) и беззнаковых (USAT) данных (см. таблицу А.4).

Наиболее часто арифметика с насыщением используется при обработке сигналов, например, при их усилении. При усилении сигнала всегда

существует вероятность того, что выходной сигнал выйдет за пределы допустимого диапазона. Если корректировка значений осуществляется простым отбрасыванием неиспользуемых старших битов, то в результате переполнения получается совершенно искажённый сигнал (рисунок 5).

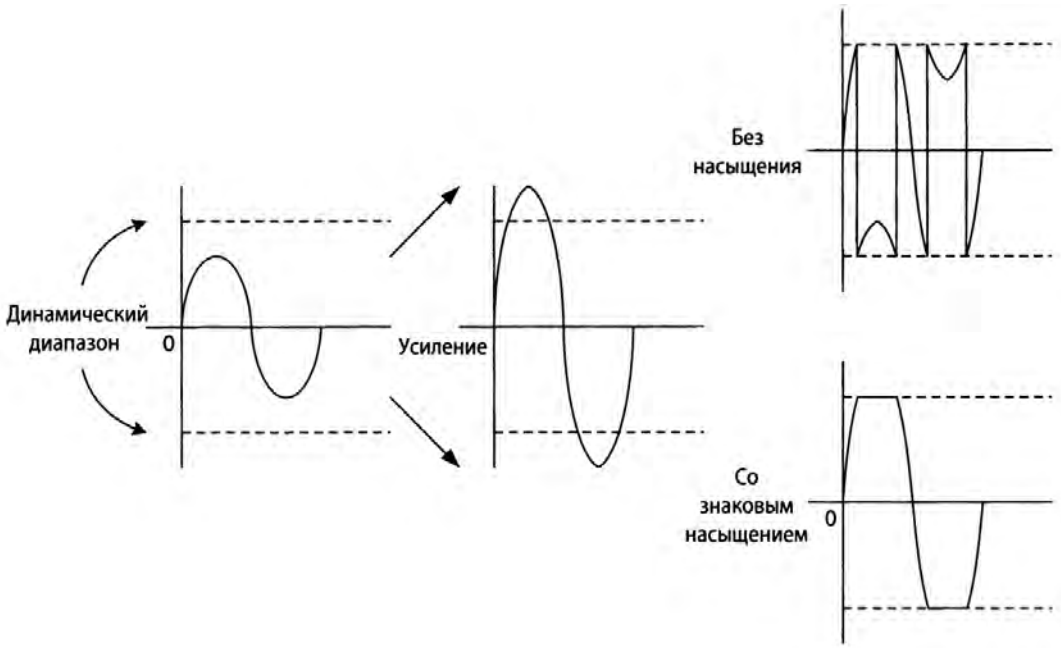


Рисунок 5 – Операция знакового насыщения

Операция насыщения не предотвращает искажение сигнала, однако она, по крайней мере, значительно уменьшает степень этих искажений.

Операции влияют не только на содержимое регистра-приёмника, но и на состояние флага Q регистра APSR. Этот флаг устанавливается в 1, если при выполнении операции произошло насыщение.

Диапазон значений, в которых может изменяться значение насыщения, устанавливается вторым операндом команды и может изменяться в пределах от -2^{n-1} до $2^{n-1}-1$ (n = 1–32) при знаковом насыщении и от 0 до $2^{n-1}-1$ (n = 0–31) при беззнаковом насыщении.

Например, если необходимо выполнить насыщение 32-битного значения в пределах 16-битного знакового диапазона, то можно использовать следующую команду:

```
SSAT.W R1, #16, R0
```

Команды насыщения также могут использоваться для преобразования типов данных. В частности, их можно использовать для преобразования 32-битного целого значения в 16-битное. Однако компиляторы языка Си могут оказаться не в состоянии напрямую использовать эти команды, что потребует использования встроенных ассемблерных функций (или же встроенного ассемблера) для выполнения преобразования.

1.8 Группа команд операций с битами

Данная группа команд содержит инструкции, позволяющие оперировать наборами смежных битов регистров (битовыми полями); полный список поддерживаемых команд приведен в таблице А.5.

Команда BFC осуществляет очистку битового поля в регистре-приемнике, BFI – копирует битовое поле из одного регистра в другой.

Команды SXTB и SXTH используются для изменения разрядности знакового значения с байта или полуслова до слова. В случае, если байт или полуслово регистра-источника отрицательные, то знак распространяется на все старшие разряды регистра-приемника. Команды UXTB и UXTH используются для изменения разрядности беззнакового значения с байта или полуслова до слова, заполняя старшие биты нулями.

Команда RBIT осуществляет перестановку всех битов в 32-битном регистре в обратном порядке, т. е. биты регистра-источника изменяют свой вес – старший становится младшим, младший – старшим и т. д.

Команда CLZ определяет число ведущих нулевых битов в содержимом регистра-источника и возвращает результат в регистр-приемник. Результат равен 32, если ни один бит источника не установлен, и 0, если установлен 31-й бит источника.

Команды SBFX и UBFX осуществляют извлечение битового поля из регистра и сохраняют результат в регистре-приемнике. При этом SBFX осуществляет расширение знакового разряда на старшие разряды результирующего числа, а UBFX – заполняет старшие разряды нулями. Поэтому команды SBFX и UBFX с одинаковыми операндами могут дать разный результат.

Так, предположим, что в регистре R0 находится число 0x5678ABCD и над его содержимым выполняются следующие команды:

```
UBFX.W R1, R0, #4, #8
SBFX.W R1, R0, #4, #8
```

Число 0x5678ABCD в двоичной системе счисления записывается как 0101 0110 0111 1010 **1011 1100** 1101. Обе команды извлекают одни и те же битовые поля из регистра-источника, однако UBFX старшие разряды заполняет нулями и в результате в R1 будет загружено число 0x000000BC. В данном случае команда SBFX старшие разряды заполнит единицами и в R1 будет загружено число 0xFFFFFFBC.

1.9 Группа команд передачи управления

К данной группе команд относятся команды, обеспечивающие условное и безусловное ветвление, вызов подпрограмм и возврат из них (см. таблицу А.6). В этих командах используется непосредственная или ре-

гистровая адресация, т. е. адрес перехода задается непосредственно в самой команде или находится в одном из регистров.

Наиболее часто в программах применяются следующие команды безусловных переходов:

```
B label ; Переход по адресу, обозначенному меткой
BX reg  ; Переход по адресу, определяемому регистром
```

В случае команды BX младший бит значения, находящегося в регистре, определяет следующее состояние (Thumb/ARM) процессора. Поскольку процессор Cortex-M3 всегда работает в состоянии Thumb, этот бит должен быть установлен в 1. Если он окажется сброшенным, то произойдёт генерация исключения Usage Fault, поскольку в данном случае команда попытается переключить процессор в состояние ARM.

Для вызова функции следует использовать команду перехода со ссылкой.

```
BL label  ; Вызов подпрограммы по адресу, обозначенному меткой, с сохранением адреса возврата в LR
BLX reg  ; Вызов подпрограммы по адресу, определяемому регистром, с сохранением адреса возврата в LR
```

При выполнении этих команд адрес возврата сохраняется в регистре связи (LR), что даёт возможность вернуться в вызвавший подпрограмму процесс с помощью команды BX LR. При использовании команды BLX необходимо убедиться, что младший бит содержимого регистра установлен в 1. В противном случае будет сгенерировано исключение Usage Fault, поскольку команда попытается переключить процессор в состояние ARM.

Операции перехода также можно выполнять, используя команды MOV и LDR, например:

```
MOV R15, R0  ; Переход по адресу, содержащемуся в R0
LDR R15, [R0]  ; Переход по адресу, содержащемуся в ячейке памяти с адресом, определяемым R0
POP { R15 }  ; Извлекаем значение адреса возврата из стека в счётчик команд
```

Выполняя переходы с помощью указанных методов необходимо устанавливать младший бит нового значения счётчика команд в 1. В противном случае будет генерироваться исключение Usage Fault, поскольку команды будут пытаться переключить процессор в состояние ARM, которое им не поддерживается.

Для организации условных переходов существует несколько способов. Во-первых, можно воспользоваться командами CBZ и CBNZ, которые осуществляют сравнение регистра с нулем и переход, если соблюдается равенство или неравенство соответственно.

Во-вторых, можно организовать условное выполнение команд с использованием команд IT. Такие конструкции, как IT-блоки (IF-THEN), очень полезны для реализации небольших по объёму фрагментов условно выполняемого кода. Они позволяют избавиться от недостатков, связанных с использованием команд перехода, поскольку не вмешиваются в ход выполнения программы. Эти блоки могут содержать до четырёх условно выполняемых команд.

В первой строке IT-блока располагается команда IT, после которой указывается проверяемое условие. Первое выражение после команды IT выполняется в случае истинности проверяемого условия. Последующие выражения (со второго по четвёртое) могут выполняться либо в случае истинности условия, либо при его ложности. В команде IT это указывается в виде ITxyz, где истинности соответствует символ «T» (THEN), а ложности – символ «E» (ELSE):

```
     IT<x><y><z> <cond>     ; Команда IT <x>, <y>, <z>
могут быть T или E)
     instr1<cond> <operands>   ; 1-я команда (условие
<cond> должно быть таким же, как в команде IT)
     instr2<cond или not cond> <operands>   ; 2-я ко-
манда (условие может быть таким же, как в команде IT,
или обратным)
     instr3<cond или not cond> <operands>   ; 3-я ко-
манда (условие может быть таким же, как в команде IT,
или обратным)
     instr4<cond или not cond> <operands>   ; 4-я ко-
манда (условие может быть таким же, как в команде IT,
или обратным)
```

Если выражение должно выполняться в случае ложности условия <cond>, то суффикс условия команды должен быть обратным тому условию, которое указано в команде IT. Так, для условия EQ обратным является условие NE, для условия GT – LE и т. д. Возможные для применения суффиксы проверяемых условий и флаги, на основании которых они проверяются, приведены в таблице 1.

В-третьих, условные переходы можно осуществить, используя дополнительные возможности системы команд процессора Cortex-M3. Так, большинство команд могут выполняться условно, и в этом случае команда безусловного перехода совместно с суффиксом условного выполнения становится командой перехода по условию. Так, например, следующая ко-

манда представляет собой команду B, используемую совместно с суффиксом EQ (условие «Равно»).

```
BEQ label
```

Соответственно, при выполнении этой команды процессор произведет проверку флага Z регистра APSR и в случае, если флаг установлен, будет осуществлена передача управления по метке label. Различные комбинации четырёх флагов (N, Z, C и V) позволяют определить 15 условий переходов (см. таблицу 1).

В таблице 2 представлен диапазон адресуемых переходов для различных команд ветвления. Для достижения максимального диапазона может потребоваться указать суффикс размера инструкции «.W».

Таблица 2 – Диапазон адресуемых переходов для команд ветвления

Инструкция	Диапазон адресации
B label	От –16 до +16 Мбайт относительно текущей позиции
B*cond* label (вне IT-блока)	От –1 до +1 Мбайт относительно текущей позиции
B*cond* label (внутри IT-блока)	От –16 до +16 Мбайт относительно текущей позиции
BL{*cond*} label	От –16 до +16 Мбайт относительно текущей позиции
BX{*cond*} Rm	Любое значение, записанное в регистре
BLX{*cond*} Rm	Любое значение, записанное в регистре

1.10 Прочие команды

В эту группу (см. таблицу А.7) входят следующие команды:

– CPS – позволяет изменить состояние процессора путем сброса или установки регистров PRIMASK и FAULTMASK;

– DMB, DSB и ISB – осуществляют барьер синхронизации доступа к данным и инструкциям. Так, команда DMB гарантирует, что все явные операции доступа к памяти, которые были инициированы перед ее выполнением, будут завершены до того, как начнется выполнение любой операции доступа к памяти после этой инструкции. Команды, которые будут следовать в порядке выполнения после инструкции DSB, не начнут исполняться до ее завершения. Инструкция DSB завершает свою работу после того, как будут выполнены все инициированные перед ней явные операции доступа к памяти. Команда ISB осуществляет сброс конвейера инструкций процессора, гарантируя таким образом, что все команды, расположенные после инструкции ISB, по окончании ее исполнения будут загружены в конвейер повторно;

– NOP – пустая операция;

– SEV – используется для передачи информации о событии всем процессорам в составе многопроцессорной системы;

– SVC – осуществляет вызов системного исключения SVC;

– WFE и WFI– позволяют организовать в программе ожидание события и прерывание соответственно.

2 Задания 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.

При составлении программ необходимо пользоваться таблицами А.1–А.7, содержащими полный набор команд процессора. Из них следует выбрать команду (или несколько команд), которая позволит выполнить каждое действие, приведенное в заданиях. При этом вместо обозначений, используемых в таблицах для указания регистров и констант в общем виде, необходимо применять номера регистров и значения констант, которые соответствуют заданию или которые разработчик планирует использовать. Таким образом вместо указанного в таблице Rd следует применить, например, R1, а вместо #imm8 использовать реальное число #123.

Отладить каждую программу в среде µVision.

Исследовать выполнение команд каждой программы.

Составить таблицу хода выполнения программы.

Составить отчет.

Методические указания к заданиям

1 Числа, указанные в скобках, обозначают адреса ячеек памяти, а задание требует обращения (чтения или записи) к ячейке памяти с адресом, равным указанному числу.

2 Числа, указанные без скобок, – константы.

3 Числа, начинающиеся символами 0x, представлены в шестнадцатеричной системе счисления, числа без дополнительных обозначений – в десятичной системе счисления.

4 Разрядность чисел при пересылке данных в регистры или ячейки памяти равна 32 (4 байта), если иное не указано.

5 Для удобства отладки рекомендуется заканчивать каждую программу конструкцией вида «**stop B stop**» или «**B .**».

6 Символами $\overset{C}{\circlearrowright}_{2}$, $\overset{C}{\circlearrowleft}_{2}$ обозначены циклические сдвиги вправо и влево соответственно, при этом цифра внизу символа указывает, на сколько разрядов необходимо произвести сдвиг, символ с вверху указывает на необходимость сдвига через признак переноса.

7 Символами $\overset{}{\underset{1}{\rightarrow}}$, $\overset{}{\underset{1}{\leftarrow}}$ обозначены логические сдвиги вправо и влево соответственно, при этом цифра внизу символа указывает, на сколько разрядов необходимо произвести сдвиг.

8 Использование нижнего индекса для регистра, ячейки памяти или константы обозначает обращение к биту (битам), номер которого равен индексу, например, $R0_7$ – 7-й бит регистра R0.

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

3 Пример выполнения лабораторной работы

Необходимо составить программы для приведенных блок-схем алгоритмов.

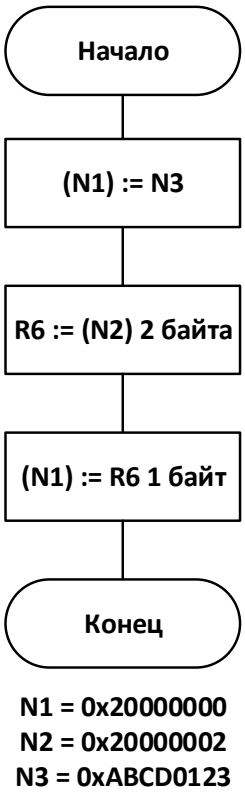
3.1 Исследование команд передачи данных

На рисунке 6 приведены блок-схема задания и текст программы для решения задачи, ход выполнения программы – в таблице 3.

Таблица 3 – Ход выполнения программы

Номер команды	Команда	Состояние операндов до выполнения команды	Состояние операндов после выполнения команды
1	MOV R2, #0x0123	R2 = 0x0	R2 = 0x0123
2	MOVT R2, #0xABCD	R2 = 0x0123	R2 = 0xABCD0123
3	MOV R3, #0x20000000	R3 = 0x0	R3 = 0x20000000
4	STR R2, [R3]	(0x20000000) = 0 (0x20000001) = 0 (0x20000002) = 0 (0x20000003) = 0	(0x20000000) = 0x23 (0x20000001) = 0x01 (0x20000002) = 0xCD (0x20000003) = 0xAB
5	LDRH R6, [R3,#2]	R6 = 0	R0 = 0xABCD
6	STRB R6, [R3]	(0x20000000) = 0x23	(0x20000000) = 0xCD
7	B .	–	–

а)



б)

```
MOV   R2, #0x0123
MOVT R2, #0xABCD
MOV   R3, #0x20000000
STR    R2, [R3]
LDRH R6, [R3,#2]
STRB R6, [R3]
B .
```

а – блок-схема алгоритма задания; б – текст программы

Рисунок 6 – Пример программы для исследования команд передачи данных

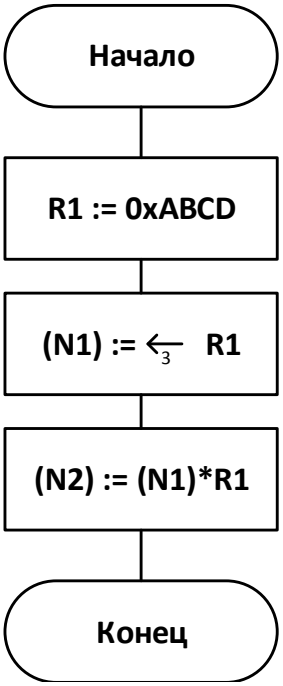
3.2 Исследование общих команд обработки информации

На рисунке 7 приведены блок-схема задания и текст программы для решения задачи, ход выполнения программы – в таблице 4.

Таблица 4 – Ход выполнения программы

Номер команды	Команда	Состояние операндов до выполнения команды	Состояние операндов после выполнения команды
1	MOV R1, #0xABCD	R1 = 0x0	R2 = 0xABCD
2	LSL R2, R1, #3	R2 = 0x0	R2 = 0x00055E68
3	MOV R3, #0x20000000	R3 = 0x0	R3 = 0x20000000
4	STR R2, [R3]	(0x20000000) = 0 (0x20000001) = 0 (0x20000002) = 0	(0x20000000) = 0x68 (0x20000001) = 0x5E (0x20000002) = 0x05
5	LDR R4, [R3]	R4 = 0x0	R4 = 0x00055E68
6	MUL R4, R1	R4 = 0x00055E68	R4 = 0x9A5C1148
	STR R4, [r3, #8]	(0x20000004) = 0 (0x20000005) = 0 (0x20000006) = 0 (0x20000007) = 0	(0x20000004) = 0x48 (0x20000005) = 0x11 (0x20000006) = 0x5C (0x20000007) = 0x9A
7	B .	–	–

а)



б)

```
MOV  R1, #0xABCD
LSL  R2, R1, #3
MOV  R3, #0x20000000
STR  R2, [R3]
LDR  R4, [R3]
MUL  R4, R1
STR  R4, [R3, #4]
B .
```

а – блок-схема алгоритма задания; б – текст программы

Рисунок 7 – Пример программы для исследования команд обработки данных

3.3 Исследование команд операций с битовыми полями

На рисунке 8 приведены текст задания и текст программы для решения задачи, ход выполнения программы – в таблице 5.

а)

Загрузить в ячейку памяти с адресом 0x20000000 число 0xCDCDCDCD. Используя операции с битовыми полями, получить слово N_5, битовые поля которого определяются как $N_{5_{15...20}} = \left(\bar{N}_1\right)_{4...9}$. Полученное число необходимо сохранить в ячейке с адресом 0x20000006.

б)

```
MOV  R1, #0xCDCDCDCD
MOV  R3, #0x20000000
STR  R1, [R3]
LDR  R2, [R3]
MVN  R2, R2
UBFX R2, R2, #4,#6
BFI  R5, R2, #15, #6
STR  R5, [R3, #6]
B .
```

а – текст задания; б – текст программы

Рисунок 8 – Пример программы для исследования команд операций с битовыми полями

Таблица 5 – Ход выполнения программы

Номер команды	Команда	Состояние операндов до выполнения команды	Состояние операндов после выполнения команды
1	MOV R1, #0xCDCDCDCD	R1 = 0x0	R1 = 0xCDCDCDCD
2	MOV R3, #0x20000000	R3 = 0x0	R2 = 0x00055E68
3	STR R1, [R3]	(0x20000000) = 0 (0x20000001) = 0 (0x20000002) = 0 (0x20000003) = 0	(0x20000000) = 0xCD (0x20000001) = 0xCD (0x20000002) = 0xCD (0x20000003) = 0xCD
4	LDR R2, [R3]	R2 = 0	R2 = 0xCDCDCDCD
5	MVN R2, R2	R2 = 0xCDCDCDCD	R2= 0x32323232
6	UBFX R2, R2, #4,#6	R2 = 0x32323232	R2 =0x00000023
7	BFI R5, R2, #15, #6	R5 = 0	R5 = 0x00118000
8	STR R5, [R3, #6]	(0x20000007) = 0 (0x20000008) = 0	(0x20000007) = 0x80 (0x20000008) = 0x11
7	B .	–	–

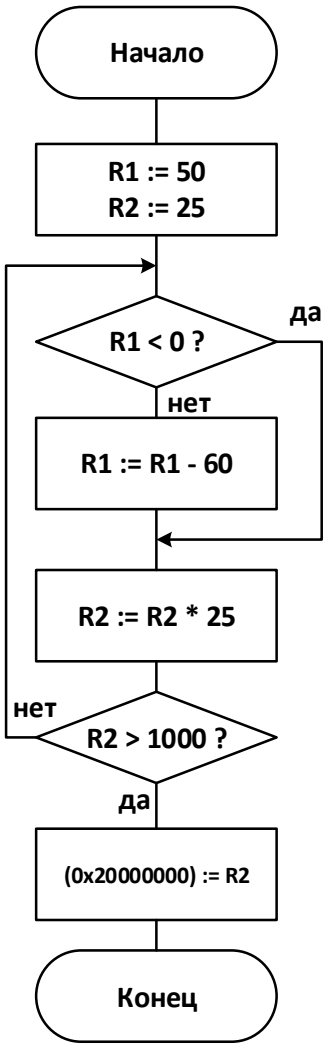
3.4 Исследование команд передачи управления

На рисунке 9 приведены блок-схема задания и текст программы для решения задачи, ход выполнения программы – в таблице 6.

Таблица 6 – Ход выполнения программы

Номер команды	Команда	Состояние операндов до выполнения команды	Состояние операндов после выполнения команды
1	MOV R1, #50	R1 = 0x0	R2 = 0x32
2	MOV R2, #25	R2 = 0x0	R2 = 0x19
3	CMP R1, #0	xPSR = 0x01000000	xPSR = 0x21000000
4	IT GE	xPSR = 0x21000000	xPSR = 0x2100A800
5	SUBGE R1, #60	R1 = 0x32 xPSR = 0x2100A800	R1 = 0xFFFFFFF6 xPSR = 0x21000000
6	MOV R3, #25	R3 = 0	R3 = 0x19
7	MUL R2, R3	R2 = 0x19	R2 = 0x271
8	CMP R2, #1000	xPSR = 0x21000000	xPSR = 0x81000000
9	BLE r2less		
3	CMP R1, #0	xPSR = 0x81000000	xPSR = 0xA1000000
4	IT GE	xPSR = 0xA1000000	xPSR = 0xA100A800
5	SUBGE R1, #60	xPSR = 0xA100A800	xPSR = 0xA1000000
6	MOV R3, #25	R3 = 0x19	R3 = 0x19
7	MUL R2, R3	R2 = 0x271	R2 = 0x3D09
8	CMP R2, #1000	xPSR = 0xA1000000	xPSR = 0x21000000
9	BLE r2less		
10	MOV R3, #0x20000000	R3 = 0x19	R3 = 0x20000000
11	STR R2, [R3]	(0x20000000) = 0 (0x20000001) = 0	(0x20000000) = 0x09 (0x20000001) = 0x3D
12	B .	–	–

а)



б)

```
    MOV  R1, #50
    MOV  R2, #25
r2less
    CMP  R1, #0
    IT   GE
    SUBGE R1, #60
    MOV  R3, #25
    MUL  R2, R3
    CMP  R2, #1000
    BLE  r2less
    MOV  R1, #0x20000000
    STR  R2, [R1]
    B .
```

а – блок-схема алгоритма задания; б – текст программы

Рисунок 9 – Пример программы для исследования команд передачи управления

Контрольные вопросы

1 Сколько команд включает система команд микроконтроллера?

2 На какие группы разделены эти команды?

3 Как длительность машинного цикла микроконтроллера соотносится с его тактовой частотой?

4 С какими типами данных может оперировать микроконтроллер?

5 Укажите назначение флагов слова состояния программы PSW.

6 Какие способы адресации используются в микроконтроллере?

7 Приведите примеры логических и арифметических команд.

8 Как инвертировать отдельные биты портов?

9 Какие переходы возможны в командах управления?

10 Поясните принципы организации условного выполнения команд.

11 Какие команды используются при организации подпрограмм?
12 Какие флаги используются командами условных переходов?

Список литературы

1 **Бородин, В. Б.** Микроконтроллеры. Архитектура, программирование, интерфейс / В. Б. Бородин, И. И. Шагурин. – Москва: ЭКОМ, 1999. – 400 с.

2 **Хвощ, С. Т.** Микропроцессоры и микроЭВМ в системах автоматического управления: справочник / С. Т. Хвощ, Н. Н. Варлинский, Е. А. Попов; под ред. С. Т. Хвоща. – Ленинград: Машиностроение, 1987. – 640 с.

3 **Нестеров, П. В.** Микропроцессоры : учебник в 3 кн. Кн. 1: Архитектура и проектирование микроЭВМ. Организация вычислительных процессов / П. В. Нестеров [и др.]; под ред. Л. Н. Преснухина. – Москва: Высшая школа, 1989. – 495 с.

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

5 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.

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

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

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

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

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

Приложение А
(рекомендуемое)

Обозначения, используемые при описании команд

В угловых скобках <> записываются альтернативные формы представления операндов, которые отделяются друг от друга символом « | ». Символы угловых скобок в команде не записываются.

В фигурных скобках {} указываются необязательные операнды. Символы фигурных скобок в команде не записываются.

Гибкий второй операнд Op2 может быть либо регистром, либо константой (см. п. 1.2.3).

Круглые скобки () в описании действий, выполняемых командой, обозначают доступ к ячейке памяти по адресу, вычисляемому с помощью выражения, находящегося внутри скобок.

{S} – необязательный суффикс, предписывающий изменить после выполнения операции флаги состояния (см. 1.3).

{cond} – большинство команд могут содержать необязательный суффикс кода условного выполнения (см. 1.3).

Ra, Rd, Rn, Rm, Rt, Rs – регистры общего назначения, используемые в качестве операндов в командах (могут выступать в качестве регистров-источников информации, а также регистров-приемников результатов выполнения операции), могут быть любыми регистрами общего назначения R0–R12.

$Rd_{y...x}$ – биты y…x регистра-приемника.

label – метка, выражение, определяемое относительно PC.

#imm8 – непосредственный операнд 8 бит.

#imm12 – непосредственный операнд 12 бит.

#imm16 – непосредственный операнд 16 бит.

#lsb – константа от 0 до 31.

#width – константа от 1 до (32-#lsb).

Carry – флаг переноса.

Остальные обозначения приведены в примечаниях к таблицам А.1–А.7.

Таблица А.1 – Команды доступа к памяти

Мнемокод команды	Операнды	Краткое описание	Операция	Флаги
ADR{cond}	Rd, label	Загрузка адреса, заданного относительно счетчика команд	Rd = label	–
CLREX{cond}	–	Сброс эксклюзивного доступа	–	–
LDM{adrmode}{cond}[1]	Rn{!}, reglist [2]	Загрузка множества регистров	$R_{reglist(1)}$ = (Rn) $R_{reglist(2)}$ = (Rn+4) . . .	–
LDR{cond}	Rt, [Rn {, offset}][3]	Загрузка слова в регистр из памяти, смещение задано непосредственно или в регистре	Rt = (Rn + offset)	–
LDRB{cond}	Rt, [Rn {, offset}][3]	Загрузка байта в регистр из памяти, смещение задано непосредственно или в регистре	Rt = (Rn + offset) 1 байт	–
LDRBT{cond}	Rt, [Rn, #offset]	Загрузка байта в регистр из памяти в режиме непривилегированного доступа	Rt = (Rn + offset) 1 байт	–
LDRD{cond}	Rt, Rt2, [Rn,#offset] [4]	Загрузка двойного слова в пару регистров из памяти	(Rt) = (Rn + offset) (Rt2) = (Rn + offset+4)	–
LDREX{cond}	Rt, [Rn, {#offset}]	Эксклюзивное чтение регистра	Rt = (Rn + offset)	–
LDREXB{cond}	Rt, [Rn]	Эксклюзивное чтение регистра, байт	Rt = (Rn + offset) 1 байт	–
LDREXH{cond}	Rt, [Rn]	Эксклюзивное чтение регистра, полуслово	Rt = (Rn + offset) полуслово	–
LDRH{cond}	Rt, [Rn {, offset}][3]	Загрузка полуслова в регистр из памяти, смещение задано непосредственно или в регистре	Rt = (Rn + offset) полуслово	–
LDRHT{cond}	Rt, [Rn, #offset]	Загрузка полуслова в регистр из памяти в режиме непривилегированного доступа	Rt = (Rn + offset) полуслово	–
LDRSB{cond}	Rt, [Rn {, offset}][3]	Загрузка в регистр из памяти байта со знаком, смещение задано непосредственно или в регистре	Rt = (Rn + offset) 1 байт	–
LDRSBT{cond}	Rt, [Rn, #offset]	Загрузка в регистр из памяти байта со знаком в режиме непривилегированного доступа	Rt = (Rn + offset) 1 байт	–

Продолжение таблицы А.1

Мнемокод команды	Операнды	Краткое описание	Операция	Фла-ги
LDRSH{cond}	Rt, [Rn {, offset}][3]	Загрузка в регистр из памяти полуслова со знаком, смещение задано непосредственно или в регистре	Rt = (Rn + offset) полуслово	–
LDRSHT{cond}	Rt, [Rn, #offset]	Загрузка в регистр из памяти полуслова со знаком в режиме непривилегированного доступа	Rt = (Rn + offset) полуслово	–
LDRT{cond}	Rt, [Rn, #offset]	Загрузка в регистр из памяти слова в режиме непривилегированного доступа	Rt = (Rn + offset)	–
POP{cond}	reglist [2]	Извлечь регистры из стека	Rreglist(1) = (SP) Rreglist(2) = (SP+4) . . . Rreglist(n) = (SP+4*(n – 1)) SP = SP + 4*n	–
PUSH{cond}	reglist [2]	Занести регистры в стек	(SP-4) = Rreglist(n) . . . (SP-4*n) = Rreglist(1) SP = SP – 4*n	–
STM{adrmode}{cond}[1]	Rn{!}, reglist [2]	Сохранение множества регистров	(Rn) = Rreglist(1) (Rn+4) = Rreglist(2) . . .	–
STR{cond}	Rt, [Rn {, offset}][3]	Сохранение регистра в памяти, смещение задано непосредственно	(Rn + offset) = Rt	–
STRB{cond}	Rt, [Rn {, offset}][3]	Сохранение регистра в памяти, байт, смещение задано непосредственно или в регистре	(Rn + offset) = Rt 1 байт	–
STRBT{cond}	Rt, [Rn, #offset]	Сохранение регистра в памяти, байт, в режиме непривилегированного доступа	(Rn + offset) = Rt 1 байт	–
STRD{cond}	Rt,Rt2,[Rn,#offset] [4]	Сохранение пары регистров в памяти, двойное слово	(Rn + offset) = Rt (Rn + offset+4) = Rt2	–

Окончание таблицы А.1

Мнемокод команды	Операнды	Краткое описание	Операция	Флаги
STREX{cond}	Rd, Rt, [Rn, #offset]	Эксклюзивная запись регистра	(Rn + offset) = Rt	–
STREXB{cond}	Rd, Rt, [Rn]	Эксклюзивная запись регистра, байт	(Rn + offset) = Rt 1 байт	–
STREXH{cond}	Rd, Rt, [Rn]	Эксклюзивная запись регистра, полуслово	(Rn + offset) = Rt полуслово	–
STRH{cond}	Rt, [Rn {, offset}][3]	Сохранение регистра в памяти, полуслово, смещение задано непосредственно или в регистре	(Rn + offset) = Rt полуслово	–
STRHT{cond}	Rt, [Rn, #offset]	Сохранение регистра в памяти, полуслово, в режиме непривилегированного доступа	(Rn + offset) = Rt полуслово	–
STRT{cond}	Rt, [Rn, #offset]	Сохранение регистра в памяти, слово, в режиме непривилегированного доступа	(Rn + offset) = Rt	–

Примечания

1 adrmode – режим адресации – IA с увеличением адреса после каждого доступа (по-умолчанию), DB с уменьшением адреса перед каждым доступом, FD с увеличением адреса после каждого доступа при обращении к стеку, EA с уменьшением адреса перед каждым доступом при обращении к стеку.

2 reglist – список регистров, заключенных в {}, отдельные регистры отделяются «,», диапазон регистров «–», в обозначении операции $R_{reglist(1)}$ – первый регистр из списка, $R_{reglist(n)}$.– последний.

3 Необязательное смещение может быть:

а) непосредственным, которое записывается следующими способами (примеры команд с использованием данного типа смещения см. в подразд. 1.2):

– [Rn] – смещение отсутствует;

– [Rn , #offset] – значение смещения добавляется к содержимому регистра Rn или вычитается из него, результат используется в качества адреса чтения или записи;

– [Rn, #offset]! – с предындексированием – значение смещения добавляется к содержимому регистра Rn или вычитается из него, результат используется в качестве адреса чтения или записи и записывается обратно в регистр Rn;

– [Rn], #offset – с постиндексированием – содержимое регистра Rn используется в качества адреса чтения или записи, значение смещения к содержимому регистра Rn или вычитается из него, после чего записывается обратно в регистр Rn.

б) задаваемым при помощи регистра с необязательным сдвигом в формате [Rn, Rm {, LSL #n}] и определяемым как offset = Rm * 2n (в случае использования логического сдвига LSL); другие типы сдвига при использовании данного типа смещения см. в подразд. 1.2.

4 Допускается только непосредственное смещение

Таблица А.2 – Общие команды обработки информации

Мнемокод команды	Операнды	Краткое описание	Операция	Флаги
ADC{S}{cond}	{Rd,} Rn, Op2	Сложение с переносом	Rd = Rn + Op2 + Carry	N, Z, C, V
ADD{S}{cond}	{Rd,} Rn, Op2	Сложение	Rd = Rn + Op2	N, Z, C, V
ADD{cond} ADDW{cond}	{Rd,} Rn, #imm12	Сложение с константой	Rd = Rn + imm12	N, Z, C, V
AND{S}{cond}	{Rd,} Rn, Op2	Логическое И	Rd = Rn AND Op2	N, Z, C
ASR{S}{cond}	Rd, Rm, <Rs | #n>	Арифметический сдвиг вправо	Примечание 1	N, Z, C
BIC{S}{cond}	{Rd,} Rn, Op2	Сброс бит по маске	$Rd = Rn \ AND \ \overline{Op2}$	N, Z, C
CLZ{cond}	Rd, Rm	Определить количество ведущих нулей	Rd = количество старших нулей	–
CMN{S}{cond}	Rn, Op2	Сравнить с противоположным знаком	Rn + Op2	N, Z, C, V
CMP{S}{cond}	Rn, Op2	Сравнить	Rn – Op2	N, Z, C, V
EOR{S}{cond}	{Rd,} Rn, Op2	Исключающее ИЛИ	$Rd = Rn \oplus Op2$	N, Z, C
LSL{S}{cond}	Rd, Rm, <Rs | #n>	Логический сдвиг влево	Примечание 2	N, Z, C
LSR{S}{cond}	Rd, Rm, <Rs | #n>	Логический сдвиг вправо	Примечание 3	N, Z, C
MOV{S}{cond}	Rd, Op2	Загрузка	Rd = Op2	N, Z, C
MOVT{cond}	Rd, #imm16	Загрузка в старшее полуслово	$Rd_{31..16}$ = imm16	–
MOVW{cond} MOV{cond}	Rd, #imm16	Загрузка 16-битной константы	Rd = imm16	N, Z, C
MVN{S}{cond}	Rd, Op2	Загрузка инверсного значения	$Rd = \overline{Op2}$	N, Z, C
ORN{S}{cond}	{Rd,} Rn, Op2	Логическое ИЛИ-НЕ	$Rd = Rn \ OR \ \overline{Op2}$	N, Z, C
ORR{S}{cond}	{Rd,} Rn, Op2	Логическое ИЛИ	Rd = Rn OR Op2	N, Z, C
RBIT{cond}	Rd, Rn	Изменить на обратный порядок бит в слове	$Rd_{31} = Rn_0$ $Rd_{30} = Rn_1$ . . . $Rd_0 = Rn_{31}$	–

Продолжение таблицы А.2

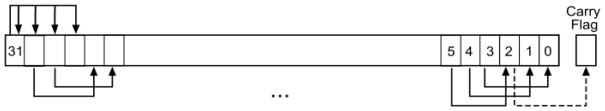
Мнемокод команды	Операнды	Краткое описание	Операция	Флаги
REV{cond}	Rd, Rn	Изменить на обратный порядок байтов в слове	$Rd_{31...24} = Rn_{7...0}$ $Rd_{23...16} = Rn_{15...8}$ $Rd_{15...8} = Rn_{23...16}$ $Rd_{7...0} = Rn_{31...24}$	–
REV16{cond}	Rd, Rn	Изменить на обратный порядок байтов в полусловах	$Rd_{31...24} = Rn_{23...16}$ $Rd_{23...16} = Rn_{31...24}$ $Rd_{15...8} = Rn_{7...0}$ $Rd_{7...0} = Rn_{15...8}$	–
REVSH{cond}	Rd, Rn	Изменить на обратный порядок байтов в младшем полуслове, произвести распространение знакового бита в старшее полуслово	$Rd_{15...8} = Rn_{7...0}$ $Rd_{7...0} = Rn_{15...8}$ если $Rd_{15...0} < 0$ то $Rd_{31...16} = 0xFFFF$	–
ROR{S}{cond}	Rd, Rm, <Rs|#n>	Циклический сдвиг вправо	Примечание 4	N, Z, C
RRX{S}{cond}	Rd, Rm	Циклический сдвиг вправо на 1 бит с учетом переноса	Примечание 5	N, Z, C
RSB{S}{cond}	{Rd,} Rn, Op2	Вычитание с противоположным порядком аргументов	Rd = Op2 – Rn	N, Z, C, V
SBC{S}{cond}	{Rd,} Rn, Op2	Вычитание с учетом переноса	Rd = Rn – Op2 – Carry	N, Z, C, V
SUB{S}{cond}	{Rd,} Rn, Op2	Вычитание	Rd = Rn – Op2	N, Z, C, V
SUB{cond} SUBW{cond}	{Rd,} Rn, #imm12	Вычитание	Rd = Rn – imm12	N, Z, C, V
TEQ{cond}	Rn, Op2	Проверка равенства	$Rn \oplus Op2$	N, Z, C

Окончание таблицы А.2

Мнемокод команды	Операнды	Краткое описание	Операция	Флаги
TST{cond}	Rn, Op2	Проверка значения бит по маске	Rn AND Op2	N, Z, C

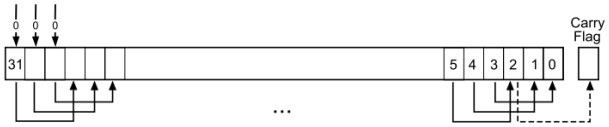
Примечания

1 Пример выполнения инструкции ASR … #3

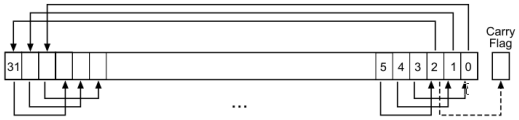


2 Пример выполнения инструкции LSL … #3

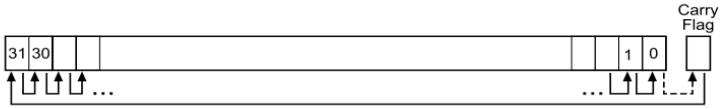
3 Пример выполнения инструкции LSR … #3



4 Пример выполнения инструкции ROR … #3



5 Пример выполнения инструкции RRX



6 В случае, если необязательный первый операнд {Rd} в соответствующих командах (например, ADD, EOR и т. д.) отсутствует, результат операции заносится в регистр, указанный в качестве первого операнда (Rn или Rm).

Таблица А.3 – Команды умножения и деления

Мнемокод команды	Операнды	Краткое описание	Операция	Фла-ги
MLA{cond}	Rd, Rn, Rm, Ra	Умножение и сложение, 32-битный результат	Rd = Ra + Rn * Rm	–
MLS{cond}	Rd, Rn, Rm, Ra	Умножение и вычитание, 32-битный результат	Rd = Ra – Rn * Rm	–
MUL{S}{cond}	{Rd,} Rn, Rm	Умножение, 32-разрядный результат	Rd = Rn * Rm	N, Z
SDIV{cond}	{Rd,} Rn, Rm	Деление чисел со знаком	Rd = Rn / Rm	–
SMLAL{cond}	RdLo, RdHi, Rn, Rm	Умножение чисел со знаком с накоплением, 64-битный результат	| RdHi, RdLo | = | RdHi, RdLo | + (Rn * Rm)	–
SMULL{cond}	RdLo, RdHi, Rn, Rm	Умножение чисел со знаком, 64-битный результат	| RdHi, RdLo | = Rn * Rm	–
UDIV{cond}	{Rd,} Rn, Rm	Деление чисел без знака	Rd = Rn / Rm	–
UMLAL{cond}	RdLo, RdHi, Rn, Rm	Умножение чисел без знака с накоплением, 64-битный результат	| RdHi, RdLo | =| RdHi, RdLo | + Rn * Rm	–
UMULL{cond}	RdLo, RdHi, Rn, Rm	Умножение чисел без знака, 64-битный результат	| RdHi, RdLo | = Rn * Rm	–
Примечание – В случае если необязательный первый операнд {Rd} в соответствующих командах (например, UDIV) отсутствует, результат операции заносится в регистр, указанный в качестве первого операнда (Rn или Rm)				

Таблица А.4 – Команды преобразования данных с насыщением

Мнемокод команды	Операнды	Краткое описание	Операция	Флаги
SSAT{cond}	Rd,#n,Rm{,shift#s}	Преобразование 32-разрядного числа в n-разрядное со знаком, с насыщением	Если Rm < -2^{n-1}, то Rd = -2^{n-1}, иначе если Rm > $2^{n-1}-1$, то Rm = $2^{n-1}-1$, иначе Rd = Rm	Q
USAT{cond}	Rd,#n,Rm{,shift#s}	Преобразование 32-разрядного число в n-разрядное без знака, с насыщением	Если Rm < 0, то Rd = 0, иначе если Rm > $2^{n-1}-1$, то Rm = $2^{n-1}-1$, иначе Rd = Rm	Q

Таблица А.5 – Команды работы с битовыми полями

Мнемокод команды	Операнды	Краткое описание	Операция	Флаги
BFC{cond}	Rd, #lsb, #width	Сброс элемента битового поля	$Rd_{lsb} = 0$ $Rd_{lsb+1} = 0$ . . . $Rd_{lsb+width-1} = 0$	–
BFI{cond}	Rd, Rn, #lsb, #width	Запись заданного значения битового поля	$Rd_{lsb} = Rn_0$ $Rd_{lsb+1} = Rn_1$ . . . $Rd_{lsb+width-1} = Rn_{width-1}$	–
SBFX{cond}	Rd,Rn,#lsb, #width	Чтение значения битового поля, интер-претируемого как число со знаком	$Rd_0 = Rn_{lsb}$ $Rd_1 = Rn_{lsb+1}$ . . . $Rd_{width-1} = Rn_{lsb+width-1}$ если $Rd_{width-1}$=1, то $Rd_{32..width}$ = 11…1, иначе $Rd_{32...width}$ = 00…0	–
SXTB{cond}	{Rd,}Rm{,ROR#n}	Преобразовать байт со знаком в слово	$Rd_{7...0} = ROR(Rm,n)_{7...0}$, если Rd_7 = 1, то $Rd_{32...8}$ = 0xFFF, иначе $Rd_{32...8}$ = 0x000	–
SXTH{cond}	{Rd,}Rm{,ROR#n}	Преобразовать полуслово со знаком в слово	$Rd_{15...0} = ROR(Rm,n)_{15...0}$, если Rd_{15} = 1, то $Rd_{32...16}$ = 0xFF, иначе $Rd_{32...16}$ = 0x00	–
UBFX{cond}	Rd, Rn, #lsb, #width	Чтение значения битового поля, интер-претируемого как число без знака	$Rd_0 = Rn_{lsb}$ $Rd_1 = Rn_{lsb+1}$ . . . $Rd_{width-1} = Rn_{lsb+width-1}$ $Rd_{32..width}$ = 00…0	–
UXTB{cond}	{Rd,}Rm{,ROR#n}	Преобразовать байт без знака в слово	$Rd_{15...0} = ROR(Rm,n)_{15...0}$ $Rd_{32...16}$ = 0x00	–
UXTH{cond}	{Rd,}Rm{,ROR#n}	Преобразовать полуслово без знака в слово	$Rd_{7...0} = ROR(Rm,n)_{7...0}$ $Rd_{32...8}$ = 0x000	–
Примечание – В случае если необязательный первый операнд {Rd} в соответствующих командах (например, SXTH) отсутствует, результат операции заносится в регистр, указанный в качестве первого операнда (Rn или Rm)				

Таблица А.6 – Команды передачи управления

Мнемокод команды	Операнды	Краткое описание	Операция	Флаги
B{cond}	label	Переход по непосредственно заданному адресу	R15 = label	–
BL{cond}	label	Переход со связью по непосредственно заданному адресу	R14 = <ad_n_i> [1] R15 = label	–
BLX{cond}	Rm	Косвенный переход со связью	R14 = <ad_n_i> [1] R15 = Rm	–
BX{cond}	Rm	Косвенный переход по адресу, заданному значением регистра	R15 = Rm	–
CBNZ	Rn, label	Сравнение с нулем и переход по неравенству	Если Rn ≠ 0, то R15 = label	–
CBZ	Rn, label	Сравнение с нулем и переход по равенству	Если Rn = 0, то R15 = label	–
IT{x{y{z}}} [2]	cond	Начало блока условно исполняемых инструкций	–	–
TBB	[Rn, Rm]	Табличный переход по индексу, смещения – байты	R15 = R15 + 2*|Rn + Rm|	–
TBH	[Rn, Rm, LSL #1]	Табличный переход по индексу, смещения – полуслова	R15 = R15 + 2*|Rn + Rm*2|	–

Примечания

1 ad_n_i – адрес следующей инструкции.

2 x, y, z – определяют выбор условия для второй, третьей, четвертой инструкции в IT-блоке; x, y, z могут быть T (Then) – инструкция выполняется, если условие истинно, E (Else) – инструкция выполняется, если условие ложно

Таблица А.7 – Прочие команды

Мнемокод команды	Операнды	Краткое описание	Операция	Флаги
BKPT	#imm8	Точка останова		–
CPSID	iflags	Изменить состояние процессора, запретить прерывания	Если iflags = i, то PRIMASK = 1 Если iflags = f, то FAULTMASK = 1	–
CPSIE	iflags	Изменить состояние процессора, разрешить прерывания	Если iflags = i, то PRIMASK = 0 Если iflags = f, то FAULTMASK = 0	–
DMB{cond}	–	Барьер синхронизации доступа к памяти данных	–	–
DSB{cond}	–	Барьер синхронизации доступа к памяти данных	–	–
ISB{cond}	–	Барьер синхронизации доступа к инструкциям	–	–
MRS{cond}	Rd, spec_reg	Считать специальный регистр в регистр общего назначения	Rd = spec_reg	–
MSR{cond}	spec_reg, Rm	Записать регистр общего назначения в специальный регистр	spec_reg = Rm	N, Z, C, V
NOP{cond}	–	Нет операции	–	–
SEV{cond}	–	Установить признак события	–	–
SVC{cond}	#imm8	Вызов супервизора	–	–
WFE{cond}	–	Ожидать событие	–	–
WFI{cond}	–	Ожидать прерывание	–	–