ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Эта книга представляет собой практическое руководство для тех, кто собирается использовать в своих новых разработках те или иные микроконтроллеры семейств LPC2300 и LPC2400 компании NXP Semiconductors. Данная книга не является ни справочником, ни учебным пособием. Предполагается, что читатель имеет некоторый опыт в области программирования микроконтроллеров для встраиваемых систем и знаком с языком программирования Си. Основной объем технической информации содержится в четырех первых главах книги, поэтому если вы совершенно не знакомы с семействами LPC2300/2400 и, в частности, с процессорным ядром ARM7, вам необходимо внимательно прочитать эти главы. Обращаю ваше внимание, что в книге речь пойдет, в основном, о семействе LPC2300, а особенности, присущие семейству LPC2400, будут отмечены особо.

В первой главе рассматриваются основные характеристики процессорного ядра (ЦПУ) ARM7. После прочтения этой главы вы будете знать достаточно, чтобы начать писать программы для любых устройств, построенных на базе ядра ARM7. Если же вы хотите расширить свои знания, к вашим услугам имеется несколько прекрасных книг, описывающих эту архитектуру, часть из которых указана в списке литературы. Во второй главе рассказывается о том, как следует писать программы на языке Си для процессора ARM7. По существу, в этой главе описываются специфические расширения стандарта ANSI C, требуемые для программирования встраиваемых систем. Все примеры, встречающиеся в книге, были написаны с использованием коммерческого компилятора, однако в настоящее время на платформу ARM перенесен и бесплатный пакет программ GCC.

После прочтения первых двух глав книги вы должны хорошо разбираться в процессоре и средствах разработки для него. Третья глава посвящена системной периферии семейства LPC2300. В ней рассказывается о системной архитектуре микроконтроллеров семейства и рассматривается вопрос конфигурирования микросхем для достижения наибольшей производительности. В четвертой главе мы познакомимся со встроенными периферийными устройствами этих микроконтроллеров и узнаем, как их необходимо конфигурировать при использовании в своих программах. В пятой главе описываются более сложные периферийные устройства, такие как контроллер USB и контроллер Ethernet. Шестая и седьмая главы книги посвящены двум популярным операционным системам реального

времени для микроконтроллеров с ядром ARM — платной, разработки компании Keil, и бесплатной FreeRTOS.

На протяжении всей книги вам будут встречаться различные упражнения, которые подробно рассматриваются в восьмой главе, посвященной практическим занятиям. Все эти упражнения можно выполнить, используя ознакомительные версии компилятора и симулятора, имеющиеся на компакт-диске, который прилагается к книге. В продаже также имеется недорогой стартовый набор разработчика (starter kit), используя который вы можете загрузить учебную программу в реальный микроконтроллер и удостовериться, что она действительно работает. Я искренне надеюсь, что, читая эту книгу и выполняя упражнения, вы быстро освоите микроконтроллеры семейств LPC2300 и LPC2400.

ГЛАВА 1

ПРОЦЕССОРНОЕ ЯДРО ARM7

1.1. Основные положения

Все микроконтроллеры семейства LPC2300 построены на основе ЦПУ ARM7. Вообще говоря, чтобы использовать эти микроконтроллеры, вам совершенно не нужно быть экспертом в области программирования процессора ARM7, поскольку заботу о большинстве сложных моментов берет на себя компилятор языка Си. Тем не менее, чтобы разработать надежное устройство, вы должны иметь хотя бы общее представление о том, как работает ЦПУ и какие у него имеются особенности.

В этой главе мы рассмотрим основные характеристики ядра ARM7 вместе с его моделью программирования, а также обсудим набор команд, используемый данным процессором. В результате вы получите всю необходимую информацию о процессоре, являющемся «сердцем» семейства LPC2300. Для более углубленного изучения процессоров ARM рекомендую обратиться к книгам, указанным в списке литературы.

Ключевой принцип, лежащий в основе процессора ARM, — простота. Ядро ARM7 является RISC-машиной, предполагающей использование небольшого числа команд и соответственно состоящей из относительно небольшого количества логических элементов. Благодаря этому процессор ARM7 идеально подходит для использования во встраиваемых системах. Он имеет высокую производительность, низкое энергопотребление и занимает небольшую часть общей площади кристалла.

1.2. Конвейер

Основной элемент ЦПУ ARM7 — конвейер команд, который используется для обработки команд, считанных из памяти программ. Конкретно, в ядре ARM7 реализован трехступенчатый конвейер (**Рис. 1.1**).

Трехступенчатый конвейер является самой простой разновидностью конвейеров и не подвержен возникновению различных опасных ситуаций, таких как «чтение раньше записи», которые встречаются в конвейерах с бóльшим числом ступеней. Конвейер имеет три аппаратно-независимые ступени, благодаря кото-

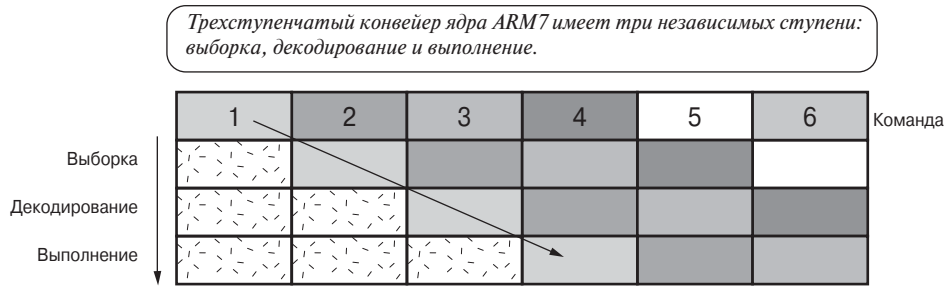


Рис. 1.1. Работа трехступенчатого конвейера.

рым одновременно с выполнением одной команды осуществляется декодирование второй и выборка третьей. Он настолько эффективно ускоряет прохождение команд через ЦПУ, что большинство команд ARM может выполняться за один такт. Конвейер наиболее эффективен при выполнении линейного кода. При обнаружении перехода конвейер сбрасывается, и для возобновления выполнения программы с максимальной скоростью он должен сначала заполниться. Позже мы с вами увидим, что набор команд процессора ARM имеет несколько интересных особенностей, позволяющих исключить из кода короткие переходы для улучшения прохождения кода по конвейеру. Поскольку конвейер является составной частью ЦПУ, он полностью скрыт от программиста. Тем не менее, важно помнить, что значение счетчика команд (Program Counter — PC) на 8 байт превышает значение адреса текущей выполняемой команды. В связи с этим необходимо аккуратно подходить к вычислению смещений в случае относительной адресации с использованием счетчика команд.

Например, команда:

```
0x40000     LDR     PC,[PC,#4]
```

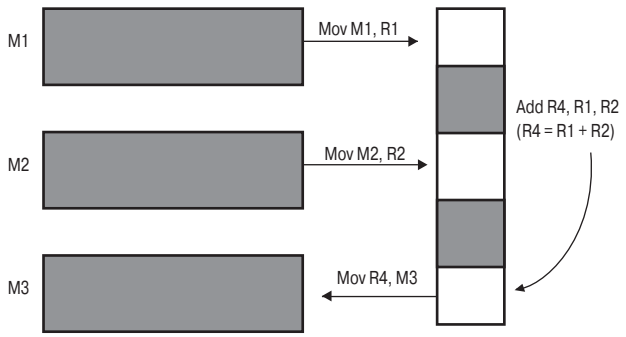
загрузит в счетчик команд PC содержимое, находящееся по адресу PC + 4. Поскольку PC опережает текущую команду на 8 байт, в него будет загружено содержимое по адресу 0x400C, а не 0x4004.

1.3. Регистры

Процессор ARM7 имеет архитектуру «load-and-store» (загрузка — сохранение), поэтому для выполнения любой обработки данных необходимо сначала перенести эти данные из памяти в определенные регистры, выполнить команду обработки данных и затем записать полученные значения обратно в память (**Рис. 1.2**).

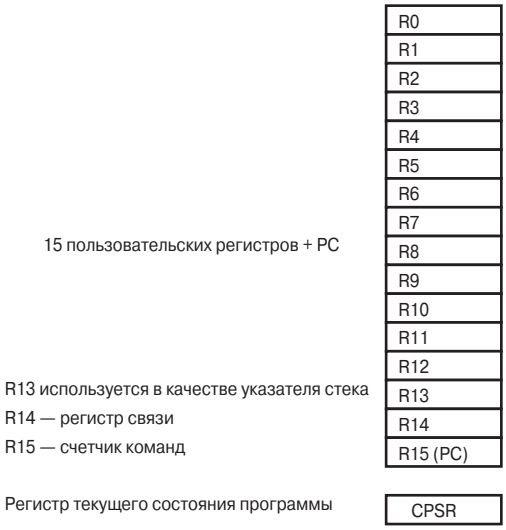
Основной регистровый файл состоит из 16 пользовательских регистров R0...R15 (**Рис. 1.3**). Каждый из этих регистров является 32-битным[1)]. Регистры

[1)] В отечественной литературе принято пользоваться понятиями «разряд», «разрядный». Двоичный разряд — это бит. В данном издании мы будем придерживаться зарубежной терминологии («бит», «битный»), что более соответствует современной тенденции в цифровой технике. — *Примеч. пер.*



ЦПУ ARM7 имеет архитектуру «load-and-store». Все команды обработки данных могут работать только с основным регистровым файлом.

Рис. 1.2. Обработка данных.



В основной регистровый файл входят шестнадцать 32-битных регистров и один дополнительный регистр ЦПУ, который называется регистром текущего состояния программы. R0...R12 — пользовательские регистры, а регистры R13...R15 выполняют специальные функции.

Рис. 1.3. Структура основного регистрового файла.

R0...R12 предназначены исключительно для нужд пользователя и не выполняют никаких других функций, в то время как регистры R13...R15 имеют дополнительные функции. Регистр R13 используется в качестве указателя стека (Stack Pointer — SP). Регистр R14 называется регистром связи (Link Register — LR). При вызове подпрограммы адрес возврата автоматически запоминается в регистре связи, откуда затем считывается при возврате. Такое решение позволяет быстро переходить к «концевым» функциям (функции, которые не вызывают других функций) и возвращаться из них. Если же функция входит в состав «ветви», т.е. вызывает другие функции, содержимое регистра связи необходимо сохранять в стеке (R13). Наконец, регистр R15 выполняет функции счетчика команд (PC). Что интересно, многие команды могут работать с регистрами R13...R15, как с обычными пользовательскими регистрами.

1.4. Регистр текущего состояния программы

Наряду с банком регистров в ЦПУ имеется дополнительный 32-битный регистр, который называется регистром текущего состояния программы (Current Program Status Register — CPSR). Регистр CPSR содержит набор флагов, которые управляют функционированием ЦПУ ARM7 и отображают его состояние (**Рис. 1.4**).

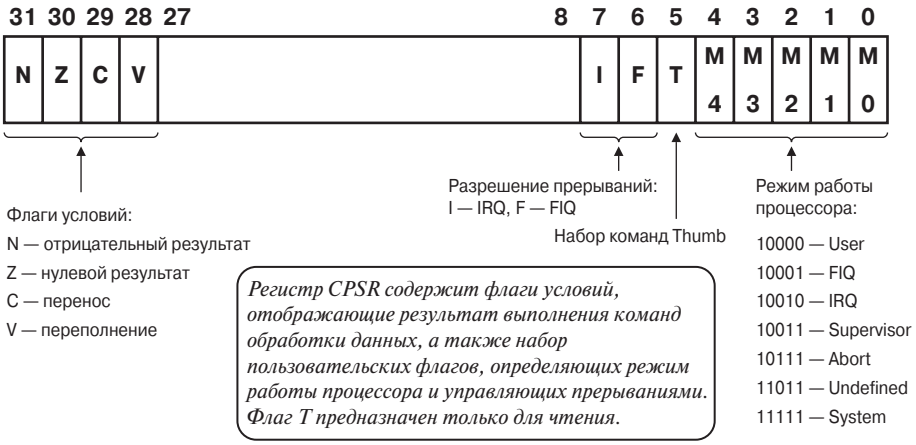


Рис. 1.4. Регистр текущего состояния программы.

В старших четырех битах регистра CPSR хранятся флаги условий, значения которых определяются процессором. Эти флаги отражают результат выполнения очередной команды обработки данных. Благодаря им вы можете узнать, не было ли получено в результате выполнения команды отрицательное или нулевое значение, а также не произошел ли перенос или переполнение. Младшие восемь битов регистра CPSR содержат флаги, значения которых прикладная программа может изменять. Биты 7 и 8 являются флагами I и F соответственно. Эти флаги используются для разрешения и запрещения двух линий прерываний, являющихся внешними по отношению к ЦПУ ARM7. Как мы с вами увидим позже, все периферийные модули микроконтроллеров LPC2300 подключены к этим двум линиям прерываний. При работе с данными битами нужно соблюдать осторожность, поскольку для запрещения любого из источников прерываний в соответствующий бит необходимо записать 1, а не 0, как можно было бы предположить. Пятый бит регистра является флагом THUMB.

ЦПУ ARM7 поддерживает два набора команд — 32-битный набор команд ARM и 16-битный набор команд THUMB. Соответственно флаг T показывает, какой из наборов команд используется. Учтите, что программа не должна пытаться напрямую устанавливать или сбрасывать этот флаг для переключения между наборами команд. Корректный механизм смены текущего набора команд мы рассмотрим чуть позже. Последние пять битов регистра CPSR являются флагами режима. В общей сложности процессор ARM7 поддерживает 7 режимов работы.

Прикладные программы, как правило, выполняются в режиме User. В этом режиме программа может обращаться к регистрам R0...R15 и CPSR. Однако при возникновении исключительных ситуаций (таких как прерывание, ошибка памяти или выполнение команды генерации программного прерывания) режим работы процессора изменяется. При этом регистры R0...R12 и R15 остаются теми же самыми, а регистры R13 (SP) и R14 (LR) заменяются новой парой регистров, уникальной для каждого режима. Таким образом, каждый режим имеет собственные регистр связи и указатель стека. Более того, в режиме быстрых прерываний (FIQ) дублируются и регистры R7...R12. Это позволит вам сразу же приступить к обработке прерывания FIQ, не тратя время на сохранение регистров в стеке.

В каждом из режимов, за исключением режима User, имеется дополнительный регистр, называемый регистром сохраненного состояния программы (Saved Program Status Register — SPSR). Если в момент возникновения исключительной ситуации программа находилась в режиме User, происходит смена режима и текущее содержимое регистра CPSR сохраняется в регистре SPSR. После обработки исключительной ситуации (при возврате из обработчика) содержимое регистра CPSR восстанавливается из SPSR, обеспечивая возобновление выполнения прикладной программы. Режимы работы процессора показаны на **Рис. 1.5**.

ЦПУ ARM7 имеет 6 различных рабочих режимов, которые используются для обработки исключительных ситуаций. Затененные регистры представляют собой дублирующие регистры, которые «включаются» при изменении режима работы. Регистр SPSR используется для сохранения содержимого регистра CPSR при переключении режимов.

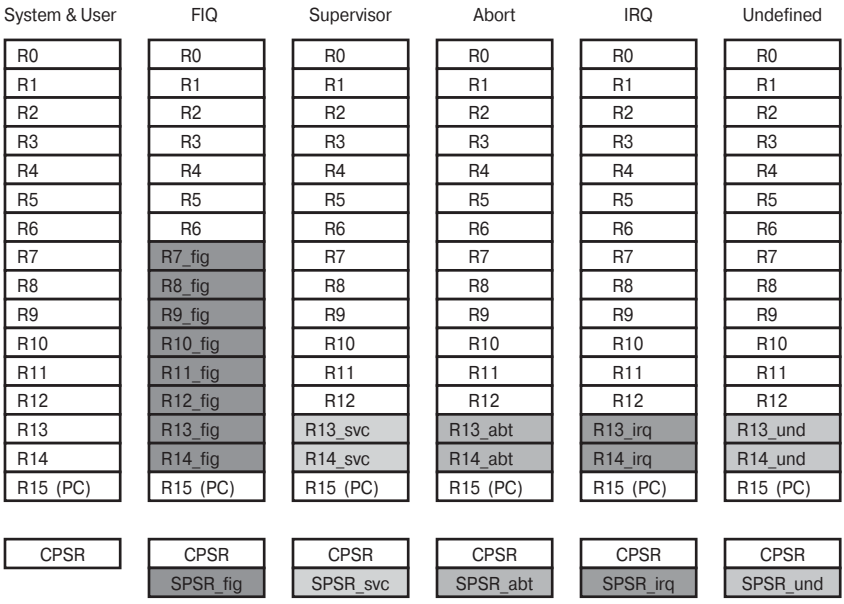


Рис. 1.5. Режимы работы процессора.

1.5. Режимы обработки исключительных ситуаций

При возникновении исключительной ситуации изменяется режим работы ЦПУ, и в PC загружается адрес соответствующего вектора прерывания (**Табл. 1.1**). Таблица векторов начинается с нулевого адреса, первым в таблице расположен вектор сброса, а за ним остальные векторы (по 4 байта на каждый).

Таблица 1.1. **Адреса векторов прерываний**

Исключительная ситуация	Режим	Адрес вектора
Reset (сброс)	Supervisor	0x00000000
Undefined instruction (неопределенная команда)	Undefined	0x00000004
SWI (программное прерывание)	Supervisor	0x00000008
Prefetch Abort (ошибка обращения к памяти при выборке команды)	Abort	0x0000000C
Data Abort (ошибка обращения к памяти при доступе к данным)	Abort	0x00000010
IRQ (прерывание)	IRQ	0x00000018
FIQ (быстрое прерывание)	FIQ	0x0000001C

Каждому режиму работы соответствует свой вектор прерывания. При смене процессором режима производится переход по этому вектору. Обратите внимание! Вектор по адресу 0x00000014 отсутствует!

Замечание. В таблице векторов имеется «дырка», поскольку вектор с адресом 0x00000014 отсутствует. Этот адрес использовался в ранних версиях процессоров ARM, а в процессоре ARM7 он сохранен, чтобы обеспечить программную совместимость между различными архитектурами ARM. Однако, как мы увидим позже, в микроконтроллерах семейства LPC2300, эти четыре байта играют очень важную роль.

При одновременном возникновении нескольких исключительных ситуаций используется метод приоритетов. Приоритеты прерываний приведены в **Табл. 1.2**.

Таблица 1.2. **Приоритеты прерываний**

Приоритет	Исключительная ситуация
Высший 1	Reset
2	Data Abort
3	FIQ
4	IRQ
5	Prefetch Abort
Низший 6	Undefined instruction SWI

Каждый источник исключительной ситуации имеет фиксированный приоритет. Встроенные периферийные устройства обслуживаются прерываниями FIQ и IRQ. Приоритеты прерываний от периферийных устройств можно назначать внутри этих групп.

Когда возникает исключительная ситуация, например, прерывание IRQ, процессор выполняет следующие действия (**Рис. 1.6**). Во-первых, адрес следующей выполняемой команды (PC + 4) сохраняется в регистре связи. Затем регистр CPSR копируется в регистр SPSR конечного режима (в нашем случае SPSR_irq). После этого в PC заносится адрес вектора прерывания режима исключительной ситуации. Для режима IRQ этот адрес — 0x00000018. В то же время режим работы процессора меняется на IRQ, в результате чего регистры R13 и R14 заменяются соответствующими регистрами этого режима. При входе в режим IRQ устанавливается флаг I регистра CPSR, что приводит к отключению линии IRQ. Если требуется использовать вложенные прерывания, то вы должны вручную разрешить

прерывание IRQ в программе и занести содержимое регистра связи в стек, чтобы сохранить исходный адрес возврата. С вектора прерывания программа перейдет к выполнению подпрограммы обработки прерываний. Первое, что необходимо сделать в этой подпрограмме, — сохранить в стеке IRQ все регистры из диапазона R0...R12, которые будут в ней использоваться. Затем можно приступать собственно к обработке исключительной ситуации.

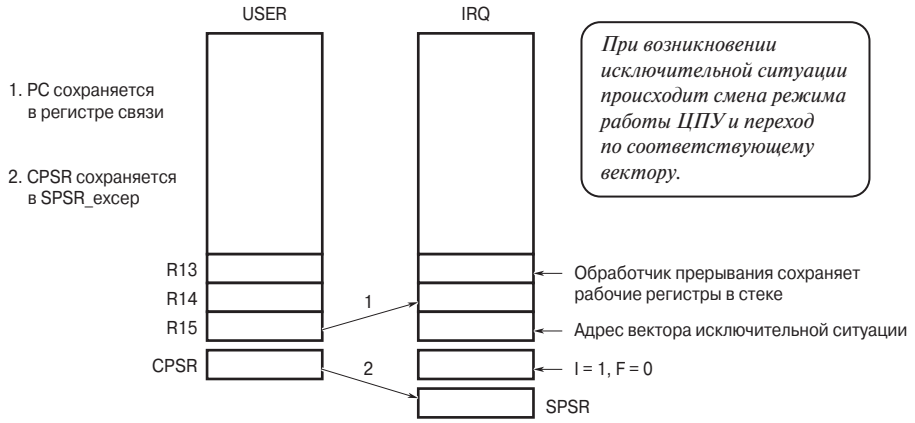


Рис. 1.6. Обработка исключительной ситуации.

После завершения обработки исключительной ситуации необходимо вернуться в режим User и продолжить выполнение программы с прерванного места. Однако в наборе команд ARM отсутствуют команды типа «возврат» или «возврат из подпрограммы», поэтому манипуляции со счетчиком команд PC необходимо осуществлять, используя обычные команды. Ситуация осложняется тем, что существует несколько различных вариантов возврата.

Для начала взглянем на команду `SWI`. При выполнении этой команды адрес следующей выполняемой команды сохраняется в регистре связи, после чего производится обработка исключительной ситуации. Все, что нам нужно сделать для возврата из исключительной ситуации, — это загрузить содержимое регистра связи обратно в PC, и программа продолжит свое выполнение. Однако чтобы ЦПУ при этом переключился обратно в режим User, необходимо использовать специальную команду пересылки `MOVS` (более подробно мы рассмотрим ее чуть позже). Таким образом, команда возврата из программного прерывания будет следующей:

```
MOVS    R15,R14         ; Скопировать регистр связи в PC и переключить режимы
```

А при возникновении исключительной ситуации по прерываниям FIQ и IRQ текущая выполняемая команда сбрасывается и выполняется переход к обработчику исключительной ситуации. При возврате из исключительной ситуации в регистре связи находится адрес отброшенной команды плюс 4. Чтобы возобновить выполнение программы с нужного места, мы должны уменьшить значение, хранящееся в регистре связи, на 4. В данном случае для уменьшения содержимого

регистра связи и сохранения результата в PC мы используем специальную команду вычитания, восстанавливающую также и режим работы ЦПУ. Таким образом, команда возврата из режимов FIQ, IRQ и Abort выглядит следующим образом:

```
SUBS    R15, R14, #4
```

В случае, если произошла ошибка обращения к памяти, исключительная ситуация возникнет через одну команду после той, выполнение которой явилось ее причиной. В идеале, в этом случае мы должны перейти к подпрограмме обработки прерывания Data Abort, выяснить и устранить причину затруднений и снова попытаться выполнить команду, вызвавшую исключительную ситуацию. Соответственно, мы должны «отмотать» PC назад на две команды — отброшенную и вызвавшую возникновение исключительной ситуации. Другими словами, нам нужно вычесть из регистра связи число восемь и сохранить результат в PC. Таким образом, команда возврата из прерывания Data Abort имеет вид:

```
SUBS    R15, R14, #8
```

При выполнении команды возврата модифицированное содержимое регистра связи загружается в счетчик команд, ЦПУ переключается обратно в режим User, а содержимое регистра SPSR переписывается обратно в CPSR. В случае возникновения исключительных ситуаций FIQ или IRQ дополнительно разрешаются соответствующие прерывания. В результате всех этих действий процессор выходит из привилегированного режима и возвращается к выполнению пользовательской программы (**Рис. 1.7**).

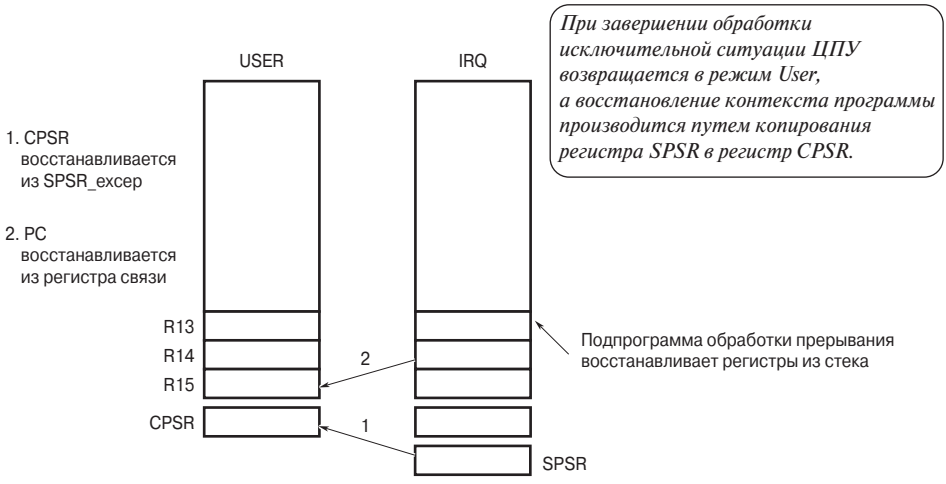


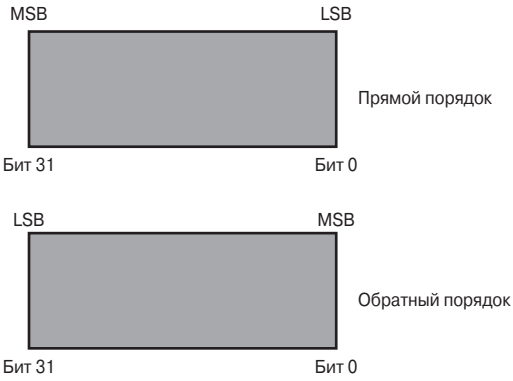
Рис. 1.7. Завершение обработки исключительной ситуации.

1.6. Набор команд ARM7

Теперь, когда мы получили общее представление о ядре ARM7, его модели программирования и режимах работы, пришла пора познакомиться с его набором или, точнее, наборами команд. Поскольку все примеры в книге написаны на

Си, вам нет необходимости быть экспертом в области программирования на ассемблере ARM7. Однако чтобы разрабатывать действительно эффективные программы, очень важно понимать машинный код, скрывающийся за строками программы на языке высокого уровня. Прежде чем мы приступим к изучению команд ARM7, необходимо отметить, что на самом деле ЦПУ ARM7 поддерживает два набора команд: набор команд ARM с 32-битными командами и набор команд THUMB с 16-битными командами. Далее в книге слово ARM будет означать 32-битный набор команд, а слово ARM7 — собственно ЦПУ.

Ядро ARM7 было разработано таким образом, чтобы его можно было использовать как в качестве процессора с обратным порядком байтов (big-endian processor), так и в качестве процессора с прямым порядком байтов (little-endian processor). В первом случае старший бит (Most Significant Bit — MSB) 32-битного слова располагается в начале слова, а во втором случае — в конце (**Рис. 1.8**). Думаю, вы обрадуетесь, узнав, что в семействе LPC2300 используется только прямой порядок байтов (т.е. MSB является битом с самым старшим адресом), что значительно облегчает работу с процессором. Однако используемый вами компилятор для ARM7 должен уметь компилировать код в обоих форматах. В связи с этим необходимо удостовериться, что формат слов задан правильно, в противном случае полученный код будет «вывернут наизнанку».

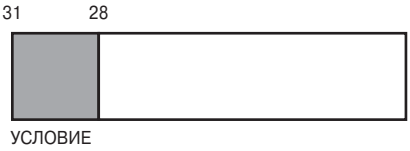


ЦПУ ARM7 поддерживает как прямой, так и обратный порядок байтов. Тем не менее, в микроконтроллерах NXP используется только прямой порядок байтов (little endian).

Рис. 1.8. Прямой и обратный порядок байтов.

Одна из наиболее интересных особенностей набора команд ARM заключается в том, что каждая команда поддерживает условное выполнение. В традиционных микроконтроллерах единственными условными командами являются команды условных переходов, и, быть может, ряд других, таких как команды проверки либо изменения состояния отдельных битов. А в наборе команд ARM старшие 4 бита кода команды всегда сравниваются с флагами условий в регистре CPSR (**Рис. 1.9**). Если их значения не совпадают, команда не выполняется и проходит через конвейер как команда `NOP` (нет операции).

Таким образом, можно выполнить какую-либо команду обработки данных, изменяющую флаги условий в регистре CPSR. Затем, в зависимости от результата, следующая команда может быть выполнена, а может и нет. К базовым мнемо-



Каждая команда ARM (32-битная) является условно выполняемой. Между 4 старшими битами кода команды и флагами условий регистра CPSR производится операция «Логическое И». Если значения не совпадают, выполняется команда NOP.

Рис. 1.9. Расположение битов сравнения в команде ARM.

ническим обозначениям команд ассемблера, таким как `MOV` или `ADD`, можно добавить любой из шестнадцати префиксов, определяющих тестируемые состояния флагов условий (**Табл. 1.3**).

Таблица 1.3. **Префиксы команд**

Префикс	Флаги	Значение
EQ	Z установлен	Равно
NE	Z сброшен	Не равно
CS	C установлен	Выше или равно (беззнаковое)
CC	C сброшен	Ниже (беззнаковое)
MI	N установлен	Отрицательный результат
PL	N сброшен	Положительный результат
VS	V установлен	Переполнение
VC	V сброшен	Нет переполнения
HI	C установлен, Z сброшен	Выше (беззнаковое)
LS	C сброшен, Z установлен	Ниже или равно (беззнаковое)
GE	N равен V	Больше или равно (знаковое)
LT	N не равен V	Меньше (знаковое)
GT	Z сброшен И (N равен V)	Больше (знаковое)
LE	Z установлен ИЛИ (N не равен V)	Меньше или равно (знаковое)
AL	(игнорируются)	Безусловное выполнение

К обозначению любой команды ARM (32-битной) можно добавить один из 16 префиксов, определяющих тестируемые флаги условий. Соответственно, существует 16 вариантов каждой команды

К примеру, команда:

```
EQMOV   R1, #0x00800000
```

выполнит загрузку числа 0x00800000 в регистр R1 только в том случае, если результат выполнения последней команды обработки данных был «равно» и соответственно установлен флаг Z регистра CPSR. Целью такого условного выполнения команд является обеспечение непрерывности потока команд через конвейер, так как при каждом выполнении команд перехода конвейер сбрасывается и на его повторное заполнение требуется время, что резко снижает общую производительность. На практике существует некоторый порог, при котором принудительное «проталкивание» команд `NOP` через конвейер оказывается эффективнее выполнения традиционных команд условного перехода и связанного с этим повторным заполнением буфера. Указанный порог равен трем командам, поэтому короткий переход, такой как:

```
if(x < 100)
{
   x++;
}
```

при использовании условно-выполняемых команд ARM будет реализован более эффективно.

Все множество команд ARM можно разбить на 6 основных групп: команды ветвления, команды обработки данных, команды передачи данных, команды передачи блоков данных, команды умножения и команда программного прерывания.

1.6.1. Команды ветвления

Базовая команда перехода (B), как следует из ее названия, позволяет выполнять переход в диапазоне до 32 Мбайт как вперед, так и назад. Модифицированная версия команды, команда перехода с сохранением адреса (BL), выполняет ту же операцию, однако при этом сохраняет в регистре связи текущее значение PC, увеличенное на четыре (**Рис. 1.10**).

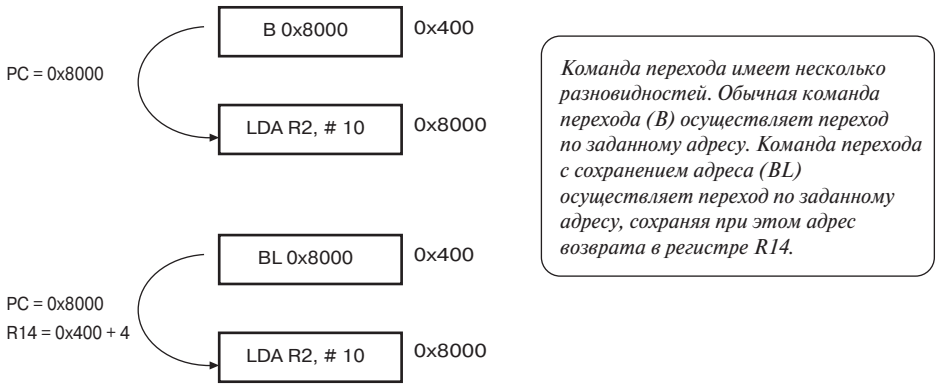


Рис. 1.10. Команды перехода B и BL.

Таким образом, команда перехода с сохранением адреса используется в качестве команды вызова подпрограмм, сохраняющей адрес возврата в регистре связи. Для возврата из подпрограмм можно использовать команду обычного перехода, выполняющую переход по адресу, находящемуся в регистре связи. Используя флаги условий, мы можем выполнять условные переходы и условные вызовы подпрограмм. Существует еще две разновидности команды перехода: «переход со сменой состояния» (BX) и «переход со сменой состояния и сохранением адреса» (BLX). Эти команды выполняют те же операции, что и предыдущие команды, но при этом еще и выполняют переключение с набора команд ARM на THUMB и обратно (**Рис. 1.11**).

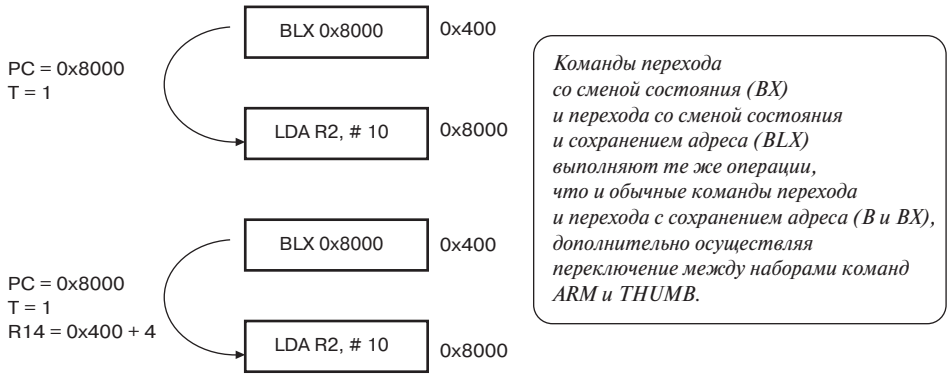


Рис. 1.11. Команды перехода BX и BLX.

Это единственный способ, который вы должны применять для изменения используемого набора команд, так как непосредственные манипуляции с флагом T регистра CPSR могут привести к непредсказуемым результатам.

1.6.2. Команды обработки данных

Обобщенный формат всех команд обработки данных приведен на **Рис. 1.12**. В каждой команде имеется регистр результата и два операнда. Первый операнд обязательно должен быть регистром, тогда как второй может быть и регистром, и константой.

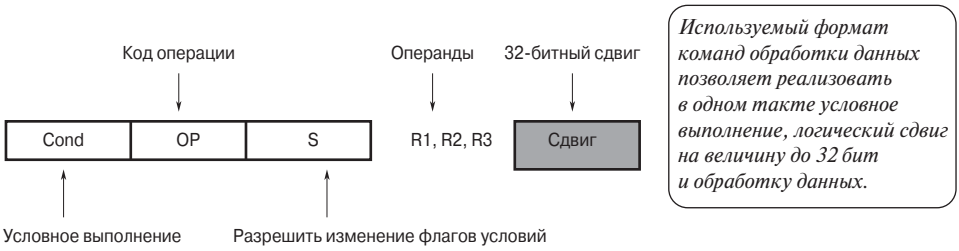


Рис. 1.12. Формат команд обработки данных.

Помимо всего прочего, в ЦПУ ARM7 имеется многорегистровое устройство циклического сдвига (barrel shifter), позволяющее при выполнении команды сдвигать значение 2-го операнда на величину до 32 бит. Бит S используется для управления флагами условий. Если этот бит установлен, флаги условий изменяются в соответствии с результатом выполнения команды. Если этот бит сброшен, состояние флагов условий не изменяется. Однако если при установленном бите S в качестве регистра результата указан счетчик команд (R15), производится копирование содержимого регистра SPSR текущего режима в регистр CPSR. Эта возможность используется для восстановления PC и переключения в исходный ре-