

## 4 ПРОГРАММНЫЙ АВТОМАТ

Программный автомат процессора управляет процессом выполнения программы, формируя адрес следующей команды, выполняемый другими устройствами процессора. Процесс выполнения программ процессором, как правило, линейный: процессор выполняет команды программы последовательно.

Изменения в этом линейном потоке происходят, когда программа использует непоследовательные программные структуры, проиллюстрированные на рис. 4-1. Непоследовательные структуры приводят к выполнению процессором команды, которая не является следующей в линейной последовательности команд. Эти структуры включают:

- **Циклы.** Последовательность команд повторяется несколько раз без непроизводительных затрат.
- **Подпрограммы.** Процессор временно прерывает последовательное выполнение программы для выполнения команды из другой области памяти.
- **Переходы.** При выполнении программы происходит постоянный переход к другой части программы.
- **Прерывания и исключения.** При появлении в процессе выполнения программы события или команды, процессор переходит к выполнению подпрограммы.
- **Ожидание.** Специальная команда, по которой процессор прекращает работу и остаётся в таком состоянии до поступления прерывания. При поступлении прерывания процессор обслуживает его и продолжает обычное выполнение программы.

# Программный автомат

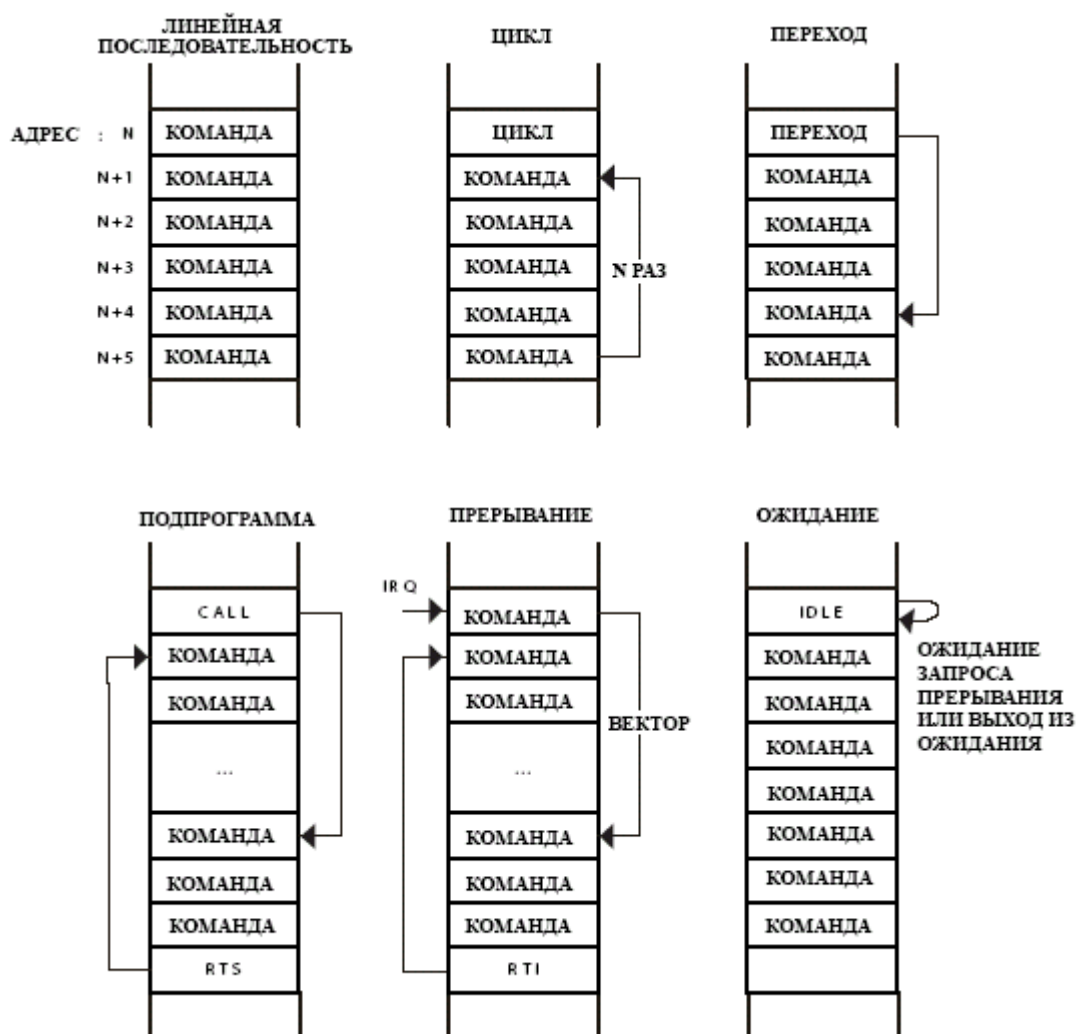


Рис. 4-1. Изменения процесса выполнения программы.

Программный автомат управляет выполнением перечисленных программных структур, выбирая адрес команды, выполняемой на следующем такте.

Выбранный адрес поступает в конвейер команд, заканчивающийся счётчиком команд (PC). Конвейер содержит 32-разрядные адреса выбираемой, декодируемой и исполняемой на данном такте команд. Счётчик команд связан с регистрами RETn, хранящими адреса возврата. Все адреса, формируемые программным автоматом, являются 32-разрядными адресами памяти команд.

Контроллер событий программного автомата управляет событиями, осуществляя обработку прерываний и событий, определяя, является ли прерывание маскируемым, и формируя соответствующий адрес вектора события.

Кроме формирования адресов данных, генераторы адреса данных могут формировать адреса команд при организации программным автоматом косвенных переходов.

# Программный автомат

Программный автомат обрабатывает условные команды и условия выхода из цикла. Регистры циклов поддерживают организацию вложенных циклов. Регистры, отображённые в карте памяти, хранят информацию, используемую при организации программ обслуживания прерываний.

## Регистры программного автомата

В таблице 4-1 перечислены регистры процессора, относящие к программному автомату. Все регистры программного автомата, за исключением регистров PC и SEQSTAT, могут быть записаны и прочитаны напрямую. Программное помещение регистров в стек и извлечение из него производится с использованием явных команд

- $[--SP] = R_n$  (для помещения регистров в стек);
- $R_n = [SP++]$  (для извлечения регистров из стека).

Таблица 4-1. Регистры программного автомата

Имя регистра	Описание
SEQSTAT	Регистр состояния программного автомата
RET RETN RETI RETE RETS	Регистры адреса возврата: см. раздел “События и выполнение программы”. Возврат из исключения Возврат из немаскируемого прерывания Возврат из прерывания Возврат из эмуляции Возврат из подпрограммы
LC0, LC1 LT0, LT1 LB0, LB1	Регистры циклов с нулевыми непроизводительными затратами: Счётчики циклов Регистры начала цикла Регистры конца цикла
FP, SP	Указатель кадра и указатель стека: см. раздел “Указатели кадра и стека” в главе 5.
SYSCFG	Регистр конфигурации системы
CYCLES, CYCLES2	Счётчики тактов: см. раздел “Регистры счётчиков тактов выполнения (CYCLES и CYCLES2)” в главе 19.
PC	Счётчик команд

## Регистр состояния программного автомата (SEQSTAT)

Регистр состояния программного автомата (SEQSTAT), содержит информацию о текущем состоянии программного автомата, а также диагностическую информацию о последнем событии. Регистр SEQSTAT доступен только для чтения и только в режиме Супервизора.

# Программный автомат

## Регистр состояния программного автомата (SEQSTAT)

Только для чтения.

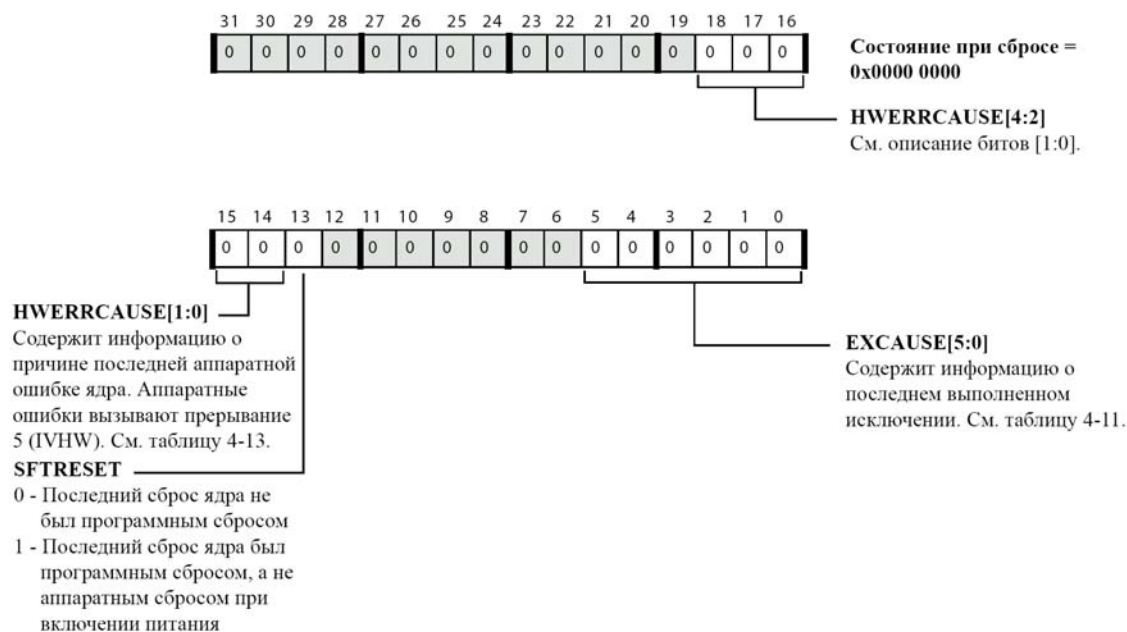


Рис. 4-2. Регистр состояния программного автомата

## Регистры циклов с нулевыми непроизводительными затратами (LC, LT, LB)

Два набора регистров циклов с нулевыми непроизводительными затратами обеспечивают организацию циклов, используя для оценки условий выхода из циклов аппаратные счётчики вместо команд программы. После оценки условия выполнение программы начинается с нового адреса. Оба набора регистров включают в себя регистры счётчика цикла (LC), начала цикла (LT) и конца цикла (LB).

Наборы 32-разрядных регистров циклов описаны в таблице 4-2.

Таблица 4-2 Регистры циклов

Регистры	Описание	Назначение
LC0, LC1	Счётчики циклов	Содержат количество оставшихся итераций цикла
LT0, LT1	Регистры начала цикла	Содержат адрес первой команды в теле цикла
LB0, LB1	Регистры конца цикла	Содержат адрес последней команды в теле цикла

## Регистр конфигурации системы (SYSCFG)

Регистр конфигурации системы (SYSCFG), управляет конфигурацией процессора. Данный регистр доступен только в режиме Супервизора.

# Программный автомат

## Регистр конфигурации системы (SYSCFG)

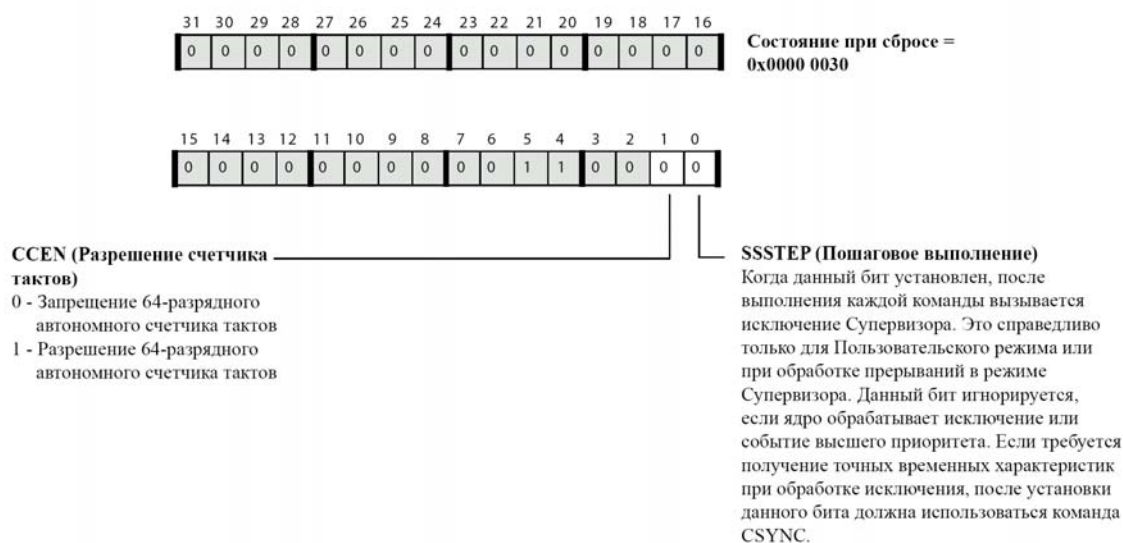


Рис. 4-3. Регистр конфигурации системы

## Конвейер команд

Программный автомат определяет адрес следующей команды на основании команды, исполняемой на данном такте, и текущего состояния процессора. Если условия не требуют иного поведения, процессор выполняет команды из памяти программы последовательно в порядке возрастания адреса.

Процессор имеет десятиуровневый конвейер команд, описываемый в таблице 4-3.

Таблица 4-3. Уровни конвейера команд

Уровень конвейера	Описание
Выборка команды 1 (IF1)	Начало доступа к памяти команд
Выборка команды 2 (IF2)	Промежуточный уровень доступа к памяти
Выборка команды 3 (IF3)	Окончание доступа к памяти команд
Декодирование команды (DEC)	Выравнивание команды, начало декодирования команды и доступ к регистровому файлу указателей
Вычисление адреса (AC)	Вычисление адресов данных и адреса перехода
Исполнение 1 (EX1)	Начало доступа к памяти данных
Исполнение 2 (EX2)	Чтение регистрового файла
Исполнение 3 (EX3)	Окончание доступа к памяти данных и начало выполнения двухтактных команд
Исполнение 4 (EX4)	Выполнение одноктактной команды
Запись результата (WB)	Запись в регистровые файлы данных и указателей и обработка событий

На рис. 4-4 показана диаграмма конвейера.

# Программный автомат

	Выборка команды 1	Выборка команды 2	Выборка команды 3	Декодирование команды	Вычисление адреса	Ex1	Ex2	Ex3	Ex4	WB
Выборка команды 1	Выборка команды 2	Выборка команды 3	Декодирование команды	Вычисление адреса	Ex1	Ex2	Ex3	Ex4	WB	

Рис. 4-4. Конвейер процессора

Программный автомат декодирует и распределяет операции между устройством памяти команд и устройством выравнивания команд. Он также управляет остановами и аннулированием команд конвейера. Программный автомат гарантирует полную замкнутость конвейера, что позволяет избавить программиста от необходимости управления конвейером.

Логика выборки команд и организации переходов формирует 32-разрядные адреса выборок для устройства памяти команд. Устройство выравнивания команд возвращает команды и информацию об их разрядности в начале уровня декодирования команды.

Для каждого типа команд (16-, 32- или 64-разрядных) устройство выравнивания гарантирует наличие достаточного количества достоверных данных в буферах выравнивания для формирования команды на каждом такте. Так как команды могут быть 16-, 32- или 64-разрядными, устройству выравнивания может не понадобиться совершать выборку данных из кэша в каждом такте. Например, при последовательном выполнении 16-разрядных команд, устройство выравнивания получает данные из устройства памяти команд один раз за четыре такта. Логика выравнивания запрашивает адрес следующей команды на основании состояния буферов выравнивания. Программный автомат отвечает за формирование адреса выборки в следующем такте при условии, что процесс выполнения программы не прерывается изменениями.

Программный автомат хранит адрес выборки до получения запроса от логики выравнивания или до изменения процесса выполнения программы. Он всегда инкрементирует предыдущий адрес выборки на 8 (следующие 8 байтов). Если происходит изменение процесса выполнения команды, вызванное, например прерыванием или переходом, программный автомат сообщает об этом устройству памяти команд, которое аннулирует данные, находящиеся в буфере выравнивания.

Устройство исполнения содержит два 16-разрядных умножителя, два 40-разрядных АЛУ, два 40-разрядных аккумулятора, одно 40-разрядное устройство сдвига, видеоустройство (добавляющее поддержку 8-разрядных операций АЛУ) и восьмиэлементный 32-разрядный регистровый файл данных.

Чтение регистрового файла происходит на уровне EX2 конвейера (для операндов). Запись происходит на уровне WB (для данных, записываемых в регистры). Умножители и видеоустройство активны на уровне EX3, а АЛУ и устройство сдвига – на уровне EX4. Запись в аккумуляторы производится в конце уровня EX4.

# Программный автомат

Любое изменение процесса последовательного выполнения программы может повлечь потенциальное снижение производительности процессора. Операции, нарушающие последовательное выполнение программы, включают:

- переходы;
- вызов подпрограммы и возврат из подпрограммы;
- прерывание и возврат из прерываний;
- циклы.

## Переходы и управление переходами

Одним из типов непоследовательного выполнения программы, поддерживаемых программным автоматом, являются переходы. Переход происходит, когда по команде JUMP или CALL процессор начинает выполнение команд с адреса, отличного от следующего адреса в линейной последовательности адресов. Описание использования команд JUMP и CALL см. в *Руководстве по набору команд процессора Blackfin ADSP-BF53x*. Ниже приводится краткое описание этих команд:

- При выполнении команд JUMP и CALL процесс выполнения программы переходит к другому адресу. Различие между командами JUMP и CALL заключается в том, что при выполнении команды CALL адрес возврата автоматически помещается в регистр RETS. Адресом возврата является адрес, следующий за адресом команды CALL. Таким образом, адрес возврата становится доступен команде возврата, соответствующей команде CALL, что упрощает организацию выхода из подпрограммы.
- По команде возврата программный автомат выполняет выборку команды по адресу возврата, содержащемуся в регистре RETS (при возврате из подпрограммы). К типам команд возврата относятся возврат из подпрограммы (RTS), возврат из прерывания (RTI), возврат из исключения (RTX), возврат из эмуляции (RTE) и возврат из немаскируемого прерывания (RTN). Каждому типу возврата соответствует отдельный регистр для хранения адреса возврата.
- Команды JUMP могут быть условными, зависящими от состояния бита CC в регистре ASTAT. Команды такого типа выполняются без задержки. Программный автомат может оценивать значение бита состояния CC для принятия решения о необходимости выполнения перехода. Если условие не определено, переход выполняется всегда.
- В условных командах JUMP для снижения задержки, вызванной конвейером, используется статическое предсказание переходов.

Переходы могут быть прямыми и косвенными. При прямом переходе адрес задаётся командой (например, JUMP 0x30), а при косвенном переходе адрес формируется генератором адреса данных (например, JUMP (P3)).

Все типы команд JUMP и CALL могут быть относительными (выполняются относительно счётчика команд). Косвенные переходы могут быть абсолютными или относительными.

# Программный автомат

## Прямые короткие и длинные переходы

Программный автомат поддерживает и короткие и длинные переходы. Адрес, по которому осуществляется переход, равен адресу команды из счётчика команд плюс смещение. Смещение относительно счётчика команд при коротком переходе выражается непосредственным 13-разрядным значением, которое должно делиться на 2 без остатка (бит 0 должен равняться нулю). Использование 13-разрядного значения даёт эффективный динамический диапазон смещения от –4096 до +4094 байтов.

Смещение относительно счётчика команд при длинном переходе выражается непосредственным 25-разрядным значением, которое должно делиться на 2 без остатка (бит 0 должен равняться нулю). Использование 25-разрядного значения даёт эффективный динамический диапазон смещения от –16777216 до +16777214 байтов.

Если на момент написания программы известно, что адрес, по которому будет осуществляться переход, будет смещён относительно текущего значения счётчика команд не более чем на 13-разрядную величину, можно использовать команду `JUMP.S 0xnxxx`. Если смещение, требуемое для представления адреса, по которому будет осуществляться переход, выражается значением, занимающим более 13 разрядов, необходимо использовать команду `JUMP.L 0xxxxxxxxxxx`. Если смещение, требуемое для представления адреса, по которому будет осуществляться переход, неизвестно и должно оцениваться средствами разработки, следует использовать команду `JUMP 0xxxxxxxxxxx`. При дизассемблировании данная команда заменяется соответствующей командой `JUMP.S` или `JUMP.L`.

## Прямые вызовы

Команда `CALL` представляет собой команду перехода, при которой адрес команды, следующей за ней, копируется в регистр `RETS`. В прямой команде `CALL` используется 25-разрядное смещение относительно счётчика команд, которое должно делиться на 2 без остатка (бит 0 должен равняться нулю). Применение 25-разрядного значения даёт эффективный динамический диапазон смещения от –16777216 до +16777214 байтов.

## Косвенные переходы и вызовы

В косвенных командах `CALL` и `JUMP` адреса перехода задаются значением `R`-регистра генератора адреса данных. При выполнении команды `CALL` в регистр `RETS` загружается адрес команды, следующей за командой `CALL`.

Например:

```
JUMP (R3);  
CALL (R0);
```



## Косвенные переходы и вызовы относительно счётчика команд

Команды косвенных переходов и вызовов относительно счётчика команд используют в качестве смещения адреса, по которому будет производиться переход, значение Р-регистра. При выполнении команды CALL в регистр RETS загружается адрес команды, следующей за командой CALL.

Например:

```
JUMP (PC + P3);
```

```
CALL (PC + P0);
```

## Флаг кода условия

Процессор поддерживает использование бита флага кода условия (CC), применяемого для принятия решения о необходимости перехода. Доступ к данному флагу может осуществляться в восьми случаях:

- Значение CC используется для оценки условного перехода.
- Значение регистра данных может быть скопировано в CC, и значение CC может быть скопировано в регистр данных.
- Флаг состояния может быть скопирован в CC, и значение CC может быть скопировано во флаг состояния.
- Флаг CC используется в команде BITTST.
- Бит CC может быть установлен в соответствии с результатом сравнения регистров указателей.
- Бит CC может быть установлен в соответствии с результатом сравнения регистров данных.
- В некоторых командах устройства сдвига (циклический сдвиг или VXOR) CC используется в качестве части операнда/результата.
- Бит CC может устанавливаться командами проверки и установки битов.

Перечисленные способы использования бита CC применяются при управлении процессом выполнения программы. Команды перехода не относятся к командам, устанавливающим флаги арифметического состояния. В коде команды присутствует отдельный бит, определяющий интерпретацию значения CC. Данный бит указывает на необходимость перехода по истинному или по ложному значению CC.

Операции сравнения имеют вид  $CC = expr$ , где *expr* (выражение) включает пару регистров одного типа (например, регистров данных, регистров указателей или комбинацию регистра и небольшой непосредственно заданной константы). Непосредственно задаваемая константа может быть 3-разрядным (от -4 до 3) знаковым числом при знаковом сравнении или 3-разрядным (от 0 до 7) беззнаковым числом при беззнаковом сравнении.

Бит CC может устанавливаться по результату операций “равно” (==), “меньше” (<) или “меньше или равно” (<=). Существуют также операции проверки битов,

# Программный автомат

определяющие, установлен ли бит в 32-разрядном регистре, и устанавливающие или сбрасывающие бит *CC*.

## Условные переходы

Программный автомат поддерживает организацию условных переходов. К ним относятся команды *JUMP*, при которых, в зависимости от значения бита *CC*, выполняется переход или продолжается последовательное выполнение программы. Адресом, по которому осуществляется переход, является адрес команды относительно счётчика команд плюс смещение. Смещение относительно счётчика команд представляет собой непосредственно задаваемое 11-разрядное значение, которое должно делиться на 2 без остатка (бит 0 должен быть равен нулю). Использование 11-разрядного значения даёт эффективный динамический диапазон смещения от  $-1024$  до  $+1022$  байтов.

Например, по следующей команде производится проверка флага *CC*, и при положительном результате проверки происходит переход по адресу, определяемому меткой *dest\_address*:

```
IF CC JUMP dest_address;
```

## Условные пересылки регистров

Пересылки регистров могут выполняться в зависимости от того, истинно или ложно (1 или 0) значение флага *CC*. В некоторых случаях использование подобной команды вместо перехода позволяет избежать потери тактов, возникающей при организации перехода. Эти условные пересылки могут осуществляться между любыми R- или P-регистрами (включая *SP* и *FP*).

Пример:

```
IF CC R0 = P0;
```

## Предсказание переходов

Программный автомат поддерживает статическое предсказание переходов для ускорения выполнения условных переходов. Данные переходы выполняются на основании состояния бита *CC*.

На уровне *EX4* программный автомат сравнивает действительное значение бита *CC* с предсказанным значением. Если значение бита предсказано неверно, переход корректируется; корректный адрес становится доступен на уровне *WB* конвейера.

Задержка перехода при условных переходах составляет:

- Если предсказано «не производить переход», и переход действительно не производится: 0 тактов *SSLK*.
- Если предсказано «не производить переход», а в действительности переход производится: 8 тактов *SSLK*.

# Программный автомат

- Если предсказано «произвести переход», и переход действительно производится: 4 такта ССЛК.
- Если предсказано «произвести переход», а в действительности переход не производится: 8 тактов ССЛК.

При всех безусловных переходах адрес перехода, вычисленный на уровне АС конвейера, посылается на шину адреса выборки команд в начале уровня EX1. Все безусловные переходы имеют задержку в 4 такта ССЛК.

Рассмотрим пример в таблице 4-4.

Таблица 4-4. Предсказание перехода.

Команда	Описание
If CC JUMP dest (bp)	По данной команде проверяется флаг CC, и, если он установлен, осуществляется переход по адресу, определяемому меткой dest. Если флаг CC установлен, то переход предсказывается верно, и задержка перехода уменьшается. В противном случае переход предсказан неверно и задержка перехода возрастает.

## Циклы и управление циклами

Программный автомат поддерживает механизм организации циклов с нулевыми непроизводительными затратами. Программный автомат содержит два устройства организации циклов, каждое из которых включает три регистра – регистр начала цикла (LT0, LT1), регистр конца цикла (LB0, LB1) и регистр счётчика цикла (LC0, LC1).

Когда выполняется команда по адресу X, и X совпадает с содержимым LB0, следующей выполняемой командой будет команда по адресу, содержащемуся в LT0. Другими словами, когда  $PC == LB0$ , выполняется явный переход к адресу, являющемуся содержимым LT0.

Возврат к началу цикла возможен только, если значение счётчика больше или равно 2. Если значение счётчика не равно нулю, счётчик декрементируется на 1. Например, рассмотрим случай цикла с двумя итерациями. В начале значение счётчика равняется 2. При достижении конца первой итерации цикла, значение счётчика декрементируется и становится равным 1, и процесс выполнения программы переходит к началу тела цикла (для выполнения второй итерации). Когда вновь достигается конец цикла, счётчик декрементируется и становится равным 0, но возврат к началу тела цикла не производится (так как команды в теле цикла были выполнены дважды).

Так как имеется два устройства организации цикла, циклу 1 назначен больший приоритет, и он может использоваться в качестве внутреннего во вложенных циклах. Другими словами, при возврате к началу тела цикла 1 по определённой команде ( $PC == LB1, LC1 \geq 2$ ), даже при совпадении адресов, в цикле 0 по этой команде не произойдёт переход к началу тела цикла. Возврат к началу тела цикла 0 возможен только по истечению счётчика цикла 1.

# Программный автомат

Для одновременной загрузки всех трёх регистров устройства организации цикла может использоваться команда LSETUP. Каждый из регистров цикла может быть загружен индивидуально при помощи регистровых передач, однако, это может повлечь значительные непроизводительные затраты, если во время передачи счётчик цикла содержит ненулевое значение (цикл активен).

В примере программы показан цикл с 32 итерациями, содержащий 2 команды.

Листинг 4-1. Пример цикла

```
P5 = 0x20;
LSETUP ( lp_start, lp_end ) LCO = P5;
lp_start:
R5 = R0 + R1(ns) || R2 = [P2++] || R3 = [I1++];

lp_end: R5 = R5 + R2;
```

Два набора регистров цикла используются для управления двумя вложенными циклами:

- LC[1:0] - регистры счетчика цикла.
- LT[1:0] - регистры адреса начала цикла.
- LB[1:0] - регистры адреса конца цикла.

При выполнении команды LSETUP программный автомат загружает адрес последней команды цикла в LBx и адрес первой команды цикла в LTx. Адреса начала и конца цикла вычисляются относительно счетчика команд на основании адреса команды LSETUP и смещения. В каждом случае к положению команды LSETUP прибавляется значение смещения.

LC0 и LC1 являются беззнаковыми 32-разрядными регистрами, каждый из которых поддерживает до  $2^{32}-1$  итераций в цикле.



Когда LCx = 0, организация цикл не производится, и код выполняется однократно.

Таблица 4-5. Регистры цикла

Первый/последний адрес цикла	Смещение относительно счетчика команд, используемое для вычисления адреса начала/конца цикла	Эффективный диапазон адресов команд начала/конца цикла
Начало/первый адрес	5-разрядное знаковое число; должно делиться на 2 без остатка	От 0 до 30 байтов относительно команды LSETUP
Конец/последний адрес	11-разрядное знаковое число; должно делиться на 2 без остатка	От 0 до 2046 байтов относительно команды LSETUP (задаваемый цикл может занимать до 2046 байтов)

Процессор содержит четырехэлементный буфер команд цикла, который сокращает объём выборок команд при выполнении цикла. Если код цикла содержит четыре

# Программный автомат

команды или менее, то при любом количестве итераций цикла осуществление выборок из памяти команд не требуется, так как команды хранятся локально. Использование буфера цикла эффективно сокращает время, затрачиваемое на выборки команд, при организации цикла, содержащего более четырех команд, позволяя производить выборки во время выполнения команд, находящихся в буфере цикла.

Если в команде LSETUP задаётся ненулевое смещение начального адреса цикла (`lp_start`), при первом возврате к началу цикла возникает задержка в четыре такта. Поэтому, предпочтительно использование нулевого смещения начального адреса цикла.

Процессор не имеет ограничений на тип команд в последней строке цикла. Допускается использование переходов и вызовов подпрограмм в последней команде цикла.

## События и управление событиями

Контроллер событий процессора управляет пятью типами действий или событий:

- эмуляция;
- сброс;
- немаскируемые прерывания;
- исключения;
- прерывания.

Следует отметить, что термин “событие” описывает все пять типов. В целом, контроллер событий управляет пятнадцатью событиями: Эмуляция, Сброс, Немаскируемое прерывание, Исключение и одиннадцать прерываний.

Прерывание – это событие, которое изменяет нормальный процесс выполнения программы процессором. Оно происходит асинхронно с процессом выполнения программы. В отличие от прерывания, исключение является программно-иницилируемым событием, чьи воздействия синхронны с процессом выполнения программы.

Система событий основана на использовании вложений и назначении приоритетов. Как следствие, в любой момент времени могут быть активны несколько программ обслуживания, и обслуживание события с низким приоритетом может приостанавливаться событием с более высоким приоритетом.

В процессоре реализован двухуровневый механизм управления событиями. Контроллер прерываний системы (SIC) процессора совместно с контроллером событий ядра (SEC) назначает приоритеты и управляет всеми прерываниями системы. SIC обеспечивает отображение многих источников периферийных прерываний на упорядоченные по приоритетам входы прерываний общего назначения ядра. Это отображение программируемо; индивидуальные источники прерывания могут маскироваться в SIC.

# Программный автомат

В дополнение к выделенным прерываниям и исключениям, описанным в таблице 4-6, СЕС поддерживает девять прерываний общего назначения (IVG7 – IVG15). Два прерывания с низким приоритетом (IVG14 и IVG15) рекомендуется зарезервировать для обработчиков программных прерываний, оставляя для поддержки системы семь прерываний, упорядоченных по приоритетам, (IVG7 – IVG13). См. таблицу 4-6.

Таблица 4-6. Соответствие событий ядра и системы.

	Источник события	Имя события ядра
События ядра	Эмуляция (высший приоритет)	EMU
	Сброс	RST
	NMI	NMI
	Исключение	EVX
	Зарезервировано	–
	Аппаратная ошибка	IVHW
	Таймер ядра	IVTMR
Прерывания системы	Прерывания вывода PLL из ожидания Ошибка DMA (общего характера) Прерывание ошибки PPI Прерывание ошибки SPORT0 Прерывание ошибки SPORT1 Прерывание ошибки SPI Прерывание ошибки UART	IVG7
	Прерывания часов реального времени Прерывание DMA0 (PPI)	IVG8
	Прерывание DMA1 (приём SPORT0) Прерывание DMA2 (передача SPORT0) Прерывание DMA3 (приём SPORT1) Прерывание DMA4 (передача SPORT1)	IVG9
	Прерывание DMA5 (SPI) Прерывание DMA6 (приём UART) Прерывание DMA7 (передача UART)	IVG10
	Прерывания Таймера 0, Таймера 1, Таймера 2	IVG11
	Прерывание программируемых флагов A/B	IVG12
	Прерывание DMA8/9 (поток 0 DMA типа “память-память”) Прерывание DMA10/11 (поток 1 DMA типа “память-память”) Программный сторожевой таймер	IVG13
	Программное прерывание 1	IVG14
	Программное прерывание 2 (наименьший приоритет)	IVG15

Необходимо отметить, что показанное отображение прерываний системы в события ядра является используемым по умолчанию после сброса и может быть изменено программным способом.

## Обработка прерываний системы

Диаграмма процесса обработки прерываний изображена на рис. 4-5. Когда прерывание (прерывание А) генерируется периферией с разрешенными прерываниями:

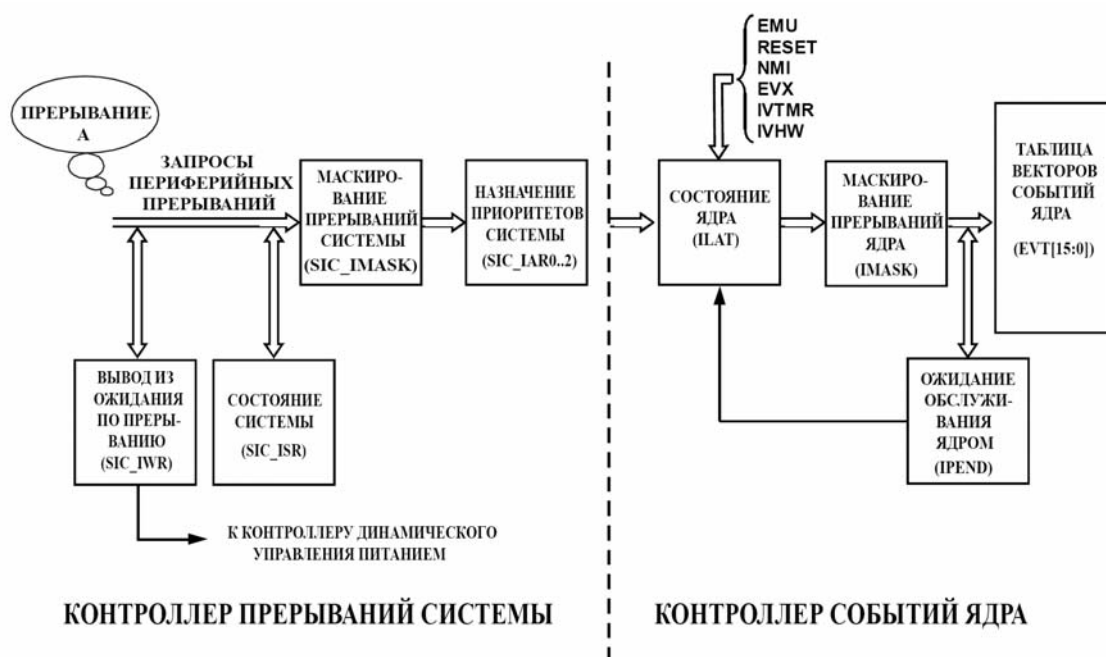
# Программный автомат

1. SIC\_ISR регистрирует запрос и отслеживает установленные, но еще не обслуженные прерывания системы (т.е. прерывания, ещё не сброшенные программой обслуживания).
2. SIC\_IWR производит проверку необходимости вывода процессора из состояния ожидания на основании запроса данного прерывания.
3. SIC\_IMASK маскирует или разрешает прерывания, поступающие от периферии, на уровне системы. Если прерывание A не маскировано, запрос поступает на обработку на шаг 4.
4. Регистры SIC\_IARx, которые отображают прерывания периферии на меньший набор прерываний ядра общего назначения (IVG7 – IVG15), определяют приоритет ядра прерывания A.
5. ILAT добавляет прерывание A в набор зарегистрированных прерываний, зафиксированных ядром, но ещё не обслуживаемых.
6. IMASK маскирует или разрешает события с различными приоритетами ядра. Если событие IVGx, соответствующее прерыванию A, не маскировано, процесс обработки продолжается на шаге 7.
7. Производится обращение к таблице векторов событий (EVT) в целях поиска соответствующего вектора программы обслуживания прерывания (ISR) для прерывания A.
8. Когда вектор события, соответствующий прерыванию A, поступает в конвейер ядра, устанавливается соответствующий бит регистра IPEND и сбрасывающий соответствующий бит регистра ILAT. Таким образом, регистр IPEND отслеживает все ожидающие обслуживания прерывания, а также все прерывания, обслуживаемые в данный момент.
9. При завершении выполнения программы обслуживания прерывания A, по команде RTI сбрасывается соответствующий бит регистра IPEND. Однако, соответствующий бит регистра SIC\_ISR не будет сброшен, пока программа обработки прерывания не сбросит механизм, вызвавший прерывание A, или если данный бит не сбрасывается в процессе обслуживания прерывания.

Следует отметить, что эмуляция, сброс, немаскируемое прерывание и исключения, а также запросы прерываний таймера ядра (IVTMR) и аппаратной ошибки (IVHW) входят в схему обработки прерывания на уровне ILAT. Регистры прерываний уровня системы (SIC\_IWR, SIC\_ISR, SIC\_IMASK, SIC\_IARx) не влияют на перечисленные выше события.

Если одно прерывание ядра одновременно обслуживает несколько источников прерываний, программа обслуживания прерывания должна идентифицировать периферийное устройство, вызвавшее прерывание. Для выполнения дальнейших действий в соответствии с конкретным источником прерывания, может потребоваться опрос периферии программой обслуживания прерывания.

# Программный автомат



Примечание: В скобках указаны названия регистров, отображённых в карте памяти

Рис. 4-5. Обработка прерывания

## Прерывания периферии системы

Система процессора имеет большое количество периферийных устройств, и, соответственно, требует поддержки большого количества прерываний. В таблице 4-7 перечислены:

- источник периферийного прерывания;
- ID периферийного прерывания, используемый в регистрах назначения прерываний системы ( $SIC\_IAR_x$ ). См. раздел “Регистры назначения прерываний системы ( $SIC\_IAR_x$ )”;
- прерывания ядра общего назначения, в которые отображаются прерывания системы при сбросе.
- ID прерывания ядра, используемый в регистрах назначения прерываний системы ( $SIC\_IAR_x$ ). См. раздел “Регистры назначения прерываний системы ( $SIC\_IAR_x$ )”.



# Программный автомат

Таблица 4-7. Источники периферийных прерываний при сбросе

Источник периферийного прерывания	ID периферийного прерывания	Прерывание общего назначения (назначенное при сбросе)	ID прерывания ядра
Прерывание вывода PLL из ожидания	0	IVG7	0
Ошибка DMA (общего характера)	1	IVG7	0
Прерывание ошибки PPI	2	IVG7	0
Прерывание ошибки SPORT0	3	IVG7	0
Прерывание ошибки SPORT1	4	IVG7	1
Прерывание ошибки SPI	5	IVG7	1
Прерывание ошибки UART	6	IVG7	1
Прерывание часов реального времени (по будильнику, секундам, минутам, часам, секундомеру)	7	IVG8	1
Прерывание DMA 0 (PPI)	8	IVG8	1
Прерывание DMA 1 (приём SPORT0)	9	IVG9	2
Прерывание DMA 2 (передача SPORT0)	10	IVG9	2
Прерывание DMA 3 (приём SPORT1)	11	IVG9	2
Прерывание DMA 4 (передача SPORT1)	12	IVG9	2
Прерывание DMA 5 (SPI)	13	IVG10	3
Прерывание DMA 6 (приём UART)	14	IVG10	3
Прерывание DMA 7 (передача UART)	15	IVG10	3
Прерывание таймера 0	16	IVG11	4
Прерывание таймера 1	17	IVG11	4
Прерывание таймера 2	18	IVG11	4
Прерывание программируемого флага A	19	IVG12	5
Прерывание программируемого флага B	20	IVG12	5
Прерывание DMA 8/9 (поток 0 DMA типа "память-память")	21	IVG13	6
Прерывание DMA 10/11 (поток 1 DMA типа "память-память")	22	IVG13	6
Прерывание программного сторожевого таймера	23	IVG13	6
Зарезервировано	24-31	–	–

Процессор имеет гибкую структуру периферийных прерываний. По умолчанию при сбросе одно прерывание ядра общего назначения обслуживает несколько периферийных прерываний, как показано в таблице 4-7.

Программа обслуживания, поддерживающая несколько источников прерываний должна опрашивать соответствующие регистры, отображённые в памяти системы, для определения периферийного устройства, вызвавшего прерывание.

Если назначения по умолчанию, показанные в таблице 4-7, приемлемы, инициализация прерывания включает в себя только:

- инициализацию элементов (адресов векторов) таблицы векторов событий ядра;
- инициализацию регистра IMASK;
- снятие маскирования определённых периферийных прерываний, требуемых системой, в регистре SIC\_IMASK.

## Регистр разрешения вывода из ожидания по прерыванию системы (SIC\_IWR)

SIC обеспечивает взаимодействие между источником периферийного прерывания и контроллером динамического управления питанием (DPMC). Любое


## Программный автомат

периферийное устройство может быть сконфигурировано на вывод ядра из состояния ожидания для обработки прерывания. Это производится путем разрешения соответствующего бита в регистре разрешения вывода из ожидания по прерыванию системы (см. рис. 4-6). Если в регистре SIC\_IWR разрешено использование источника периферийного прерывания, и ядро находится в состоянии ожидания, по возникновении прерывания DPMC инициирует процедуру вывода ядра из ожидания для его обработки. Необходимо отметить, что данный режим работы может внести задержку в обработку прерывания, зависящую от состояния управления питанием. Дальнейшее обсуждение режимов питания и состояния ожидания ядра см. в главе 8, “Динамическое управление питанием”.

По умолчанию все прерывания формируют запросы вывода ядра из ожидания. Однако в некоторых приложениях может быть желательно запрещение данной функции для некоторых периферийных устройств, например для прерывания передачи SPORTx.

Регистр SIC\_IWR не влияет на работу процессора, когда ядро не находится в режиме ожидания. Биты данного регистра соответствуют битам регистров маскирования прерываний системы (SIC\_IMASK) и состояния прерываний (SIC\_ISR).

После сброса во все задействованные биты данного регистра записывается значение 1, разрешая функцию вывода из ожидания для всех немаскированных прерываний. Данный регистр следует конфигурировать в процедуре инициализации после сброса до разрешения прерываний. Запись или чтение регистра SIC\_IWR возможна в любой момент времени. Для предотвращения ложных срабатываний и пропуска прерываний запись в данный регистр должна производиться, только когда все периферийные прерывания запрещены.

-  Необходимо отметить, что функция вывода из ожидания не зависит от функции маскирования прерываний. Если источник прерывания разрешён в регистре SIC\_ISR, но маскирован в регистре SIC\_IMASK, ядро выводится из состояния ожидания, но прерывание не генерируется.

# Программный автомат



Рис. 4-6. Регистр разрешения вывода из ожидания по прерыванию системы

## Регистр состояния прерываний системы (SIC\_ISR)

В состав SIC входит регистр состояния, доступный только для чтения, – регистр состояния прерываний системы, показанный на рис. 4-7. Каждый задействованный бит данного регистра соответствует одному из источников периферийных прерываний. Бит устанавливается, когда SIC обнаруживает установление прерывания, и сбрасывается, когда SIC обнаруживает снятие периферийного прерывания. Необходимо отметить, что для некоторых периферийных прерываний, таких как прерывания асинхронных входов программируемых флагов, от момента времени, когда программа обслуживания прерывания инициирует сброс прерывания (обычно путём записи в регистр системы, отображённый в памяти) до момента времени, когда SIC обнаруживает снятие прерывания, может пройти большое количество тактов.

В зависимости от того, как источники прерываний отображаются во входы прерываний ядра общего назначения, для определения источника прерывания может потребоваться опрос программой обслуживания прерывания нескольких битов состояния прерываний. Одна из первых команд, выполняемых в программе обслуживания прерывания, должна производить чтение регистра SIC\_ISR для определения того, не вызвано ли прерывание одновременным воздействием более чем одного из периферийных устройств, обслуживаемых одним входом прерывания. Программа обслуживания должна полностью обработать все прерывания, ожидающие обработки, перед выполнением команды RTI, которая разрешает дальнейшую генерацию прерываний по данному входу прерывания.

# Программный автомат

- ⊘ По окончании выполнения программы обслуживания прерывания, команда RTI сбрасывает соответствующий бит в регистре IPEND. Однако бит SIC\_ISR, соответствующий прерыванию, не будет сброшен до тех пор, пока программа обслуживания не произведёт сброс механизма, вызвавшего прерывание.

Во многих системах требуется использование относительно небольшого количества периферийных устройств, генерирующих прерывания, что позволяет поставить каждому периферийному устройству в соответствие уникальный уровень приоритета ядра. В таких проектах опрос регистра SIC\_ISR потребуется редко, если вообще потребуется.

На регистр SIC\_ISR не влияет состояние регистра маскирования прерывания (SIC\_IMASK), он может быть прочитан в любой момент времени. Запись в регистр SIC\_ISR не влияет на его содержимое.

## Регистр состояния прерываний системы (SIC\_ISR)

Для всех битов: 0 - прерывание снято, 1 - прерывание установлено

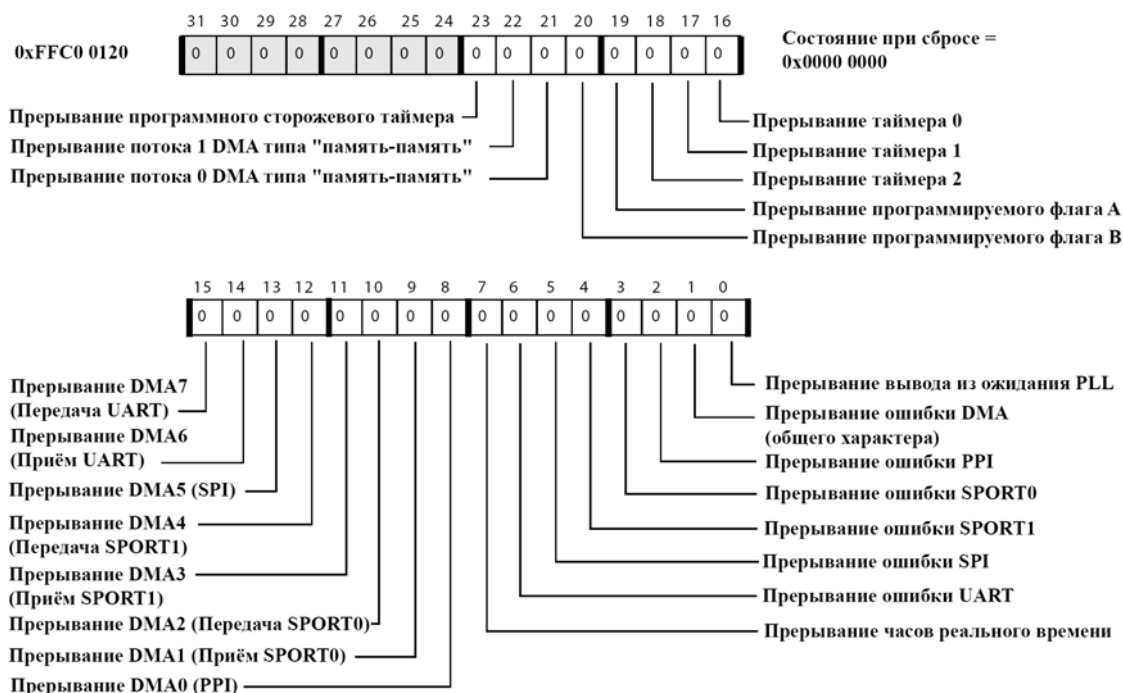


Рис. 4-7. Регистр состояния прерываний системы.

## Регистр маскирования прерываний системы (SIC\_IMASK)

Регистр маскирования прерываний системы, показанный на рис. 4-8, позволяет производить маскирование любого периферийного источника прерывания в SIC, независимо от того, разрешено ли формирование прерывания периферийным устройством.

# Программный автомат

При сбросе в регистре SIC\_IMASK все биты принудительно устанавливаются в 0 для маскирования всех периферийных прерываний. Запись 1 в определённый бит снимает маскирование и разрешает прерывание.

## Регистр маскирования прерываний системы (SIC\_IMASK)

Для всех битов: 0 - Прерывание маскировано, 1 - прерывание разрешено.

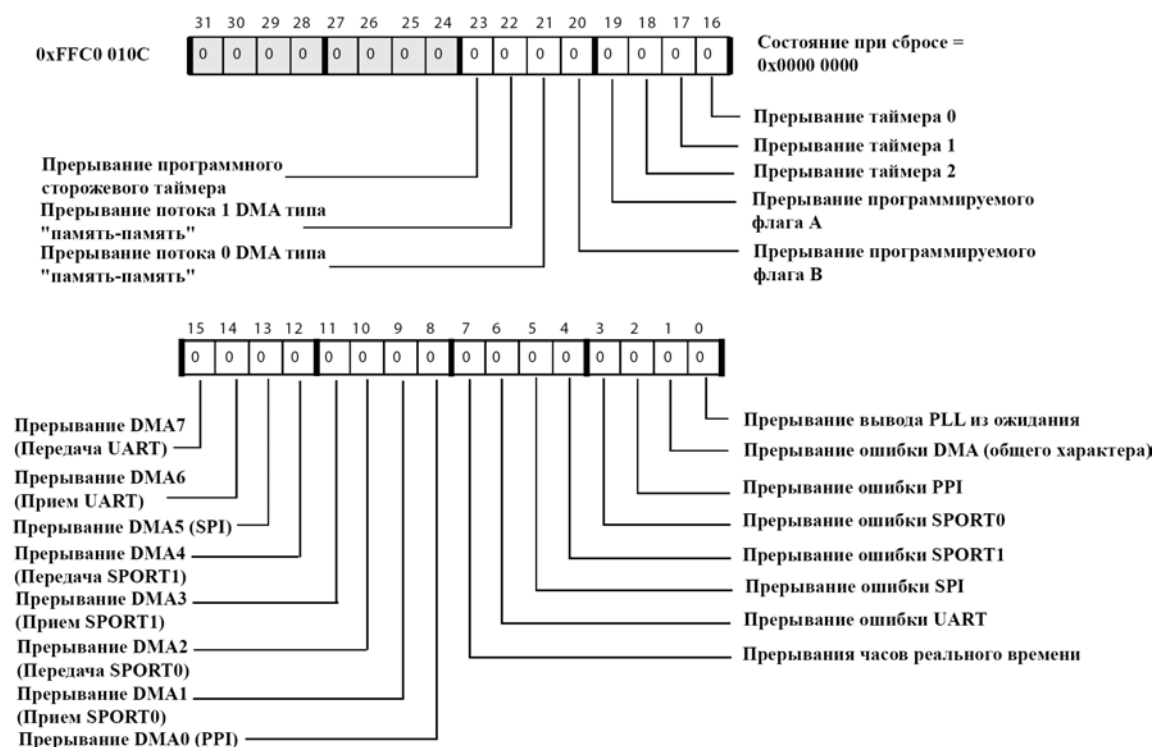


Рис. 4-8. Регистр маскирования прерываний системы

Хотя запись или чтение данного регистра может производиться в любой момент времени (в режиме Супервизора), он должен конфигурироваться до разрешения прерываний в процедуре инициализации при сбросе.

## Регистры назначения прерываний системы (SIC\_IARx)

Относительный приоритет периферийных прерываний может быть установлен путём назначения периферийному прерыванию соответствующего уровня прерывания ядра общего назначения. Данное назначение управляется настройками регистра назначения прерываний системы, как показано на рис. 4-9, 4-10 и 4-11. Если одному прерыванию поставлено в соответствие более одного источника, то запросы источников прерываний обрабатываются по схеме логического ИЛИ без аппаратного назначения приоритетов. Можно обеспечить назначение приоритетов при обработке прерываний программными методами, если это требуется приложением.

# Программный автомат

**и** При использовании прерываний общего назначения, которым назначены несколько периферийных прерываний, необходимо соблюдать особую предосторожность с целью гарантирования корректной обработки программой всех ожидающих обслуживания прерываний, обслуживаемых одним входом. За назначение приоритетов совместно обслуживаемых прерываний отвечает программное обеспечение.

Регистр назначения прерываний системы 0 (SIC\_IAR0)

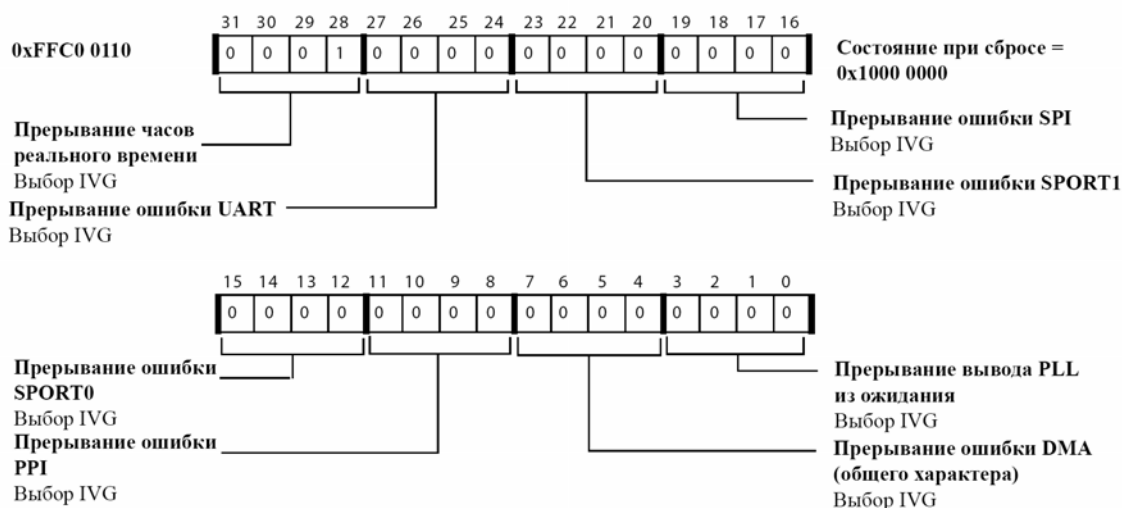


Рис. 4-9. Регистр назначения прерываний системы 0

Регистр назначения прерываний системы 1 (SIC\_IAR1)

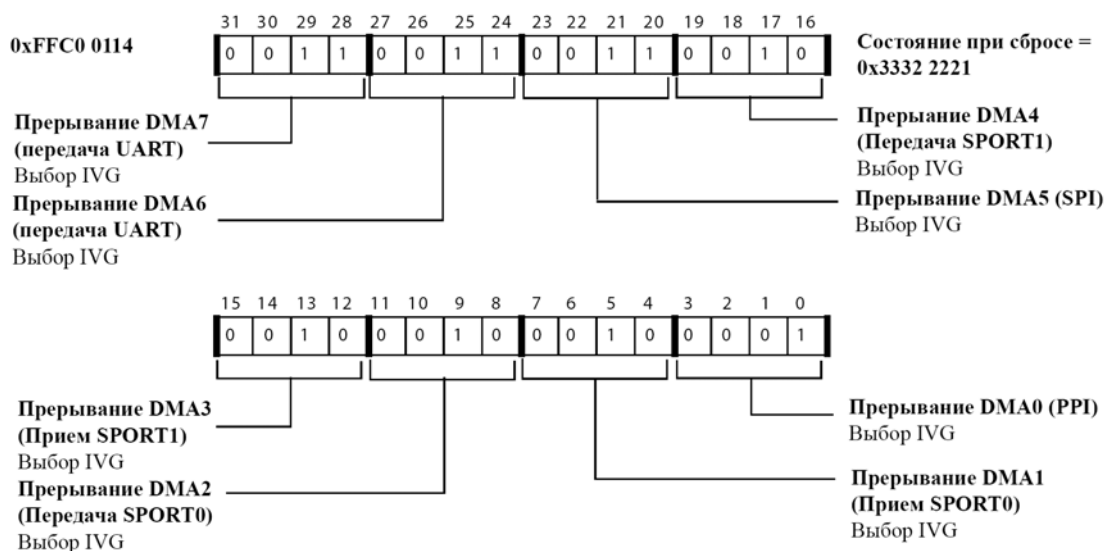


Рис. 4-10. Регистр назначения прерываний системы 1

# Программный автомат

## Регистр назначения прерываний системы 2 (SIC\_IAR2)

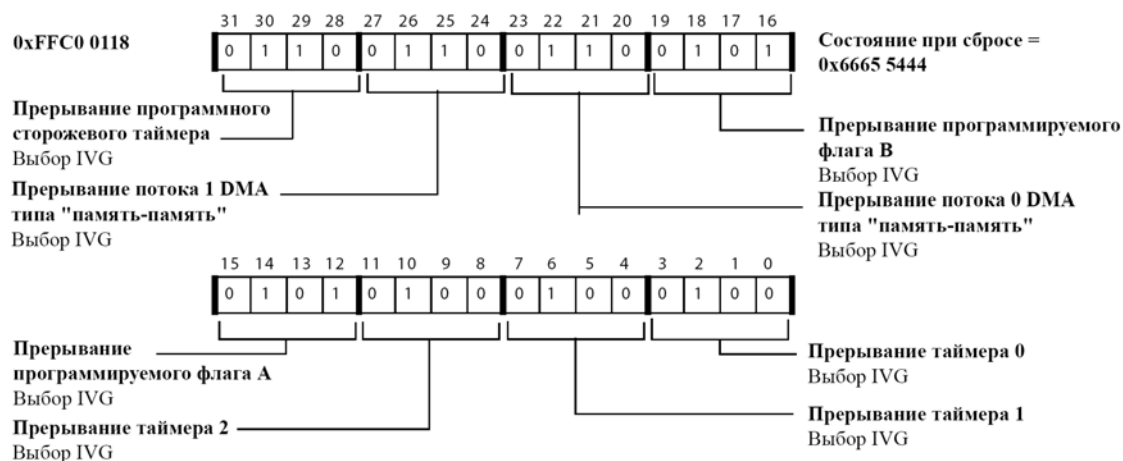


Рис. 4-11. Регистр назначения прерываний системы 2

Запись и чтение этих регистров может осуществляться в любой момент времени в режиме Супервизора. Однако их конфигурирование рекомендуется производить в программе обслуживания прерывания сброса до разрешения прерываний. Для предотвращения ложных срабатываний и пропуска прерываний запись в данный регистр следует производить, только когда запрещены все периферийные прерывания.

В таблице 4-8 приведены значения, которые необходимо записать в SIC\_IARx для конфигурирования периферийного устройства на определённый приоритет IVG.

Таблица 4-8. Выбор приоритета IVG

Прерывание общего назначения	Значение в SIC_IAR
IVG7	0
IVG8	1
IVG9	2
IVG10	3
IVG11	4
IVG12	5
IVG13	6
IVG14	7
IVG15	8

## Регистры контроллера событий

Контроллер событий использует три регистра, отображённых в карте памяти, для координирования запросов событий, ожидающих обслуживания. В каждом из этих регистров младшие 16 битов соответствуют 16 уровням событий (например, бит 0 соответствует режиму Эмуляции). К этим регистрам относятся:

- IMASK – регистр маскирования прерывания
- ILAT – регистр фиксации прерывания

# Программный автомат

- IPEND – регистр регистрации прерываний, ожидающих обслуживания.

Эти три регистра доступны только в режиме Супервизора.

## Регистр маскирования прерываний ядра (IMASK)

Этот регистр определяет, прерывания какого уровня приоритета могут обрабатываться. Запись и чтение регистра IMASK возможны в режиме Супервизора. Биты [15:5] имеют функциональное значение; биты [4:0] жёстко запрограммированы в 1, события этого уровня всегда разрешены. Если IMASK[N] == 1 и ILAT[N] == 1, прерывание N будет воспринято, при условии, что ранее не было распознано прерывание с более высоким приоритетом. Если IMASK[N] == 0 и ILAT[N] устанавливается прерыванием N, оно не будет воспринято, и ILAT[N] останется в установленном состоянии.

### Регистр маскирования прерываний ядра (IMASK)

Для всех битов: 0 - прерывание маскировано, 1 - прерывание разрешено.

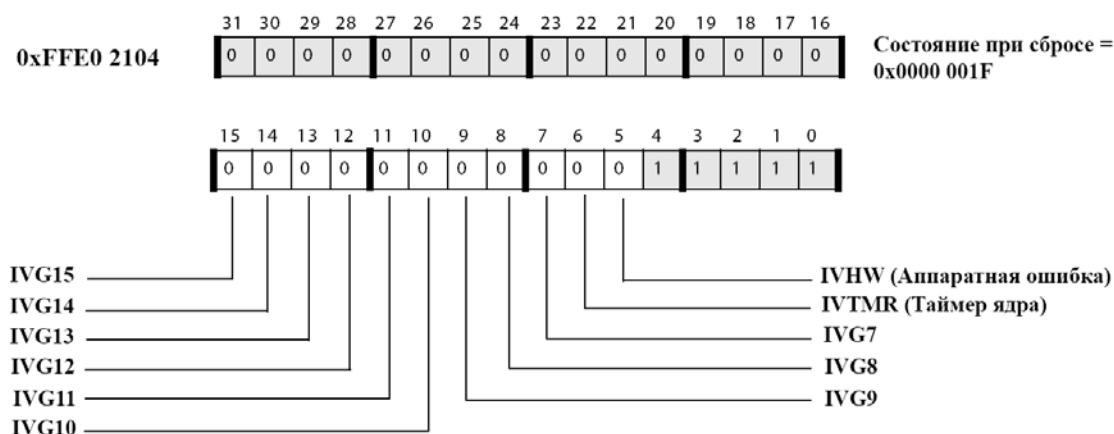


Рис. 4-12. Регистр маскирования прерываний ядра

## Регистр фиксирования прерываний ядра (ILAT)

Каждый бит регистра ILAT сигнализирует о том, что соответствующее прерывание зафиксировано, но ещё не воспринято процессором (см. рис. 4-13). Бит сбрасывается до того, как будет выполнена первая команда соответствующей программы обслуживания прерывания. Когда процессор воспринимает прерывание, ILAT[N] сбрасывается и одновременно устанавливается IPEND[N]. Чтение регистра ILAT возможно в режиме Супервизора. Запись в регистр ILAT используется только для сброса битов (в режиме Супервизора). Для сброса бита N регистра ILAT сначала необходимо убедиться в том, что IMASK[N] == 0, а затем записать в ILAT[N] единицу. Данная особенность записи в ILAT предназначена для случаев, когда необходимо сбросить (отменить) зафиксированный запрос прерывания вместо его обслуживания.



# Программный автомат

Для установки битов  $ILAT[15] \div ILAT[5]$ , а также  $ILAT[2]$  и  $ILAT[1]$ , может использоваться команда RAISE.

Бит  $ILAT[0]$  может быть сброшен только сигналом на выводе TRST интерфейса JTAG.

## Регистр фиксирования прерываний ядра (ILAT)

Значение бита 0 при сбросе зависит от эмулятора. для всех битов: 0 - прерывание не фиксировано, 1 - прерывание фиксировано.

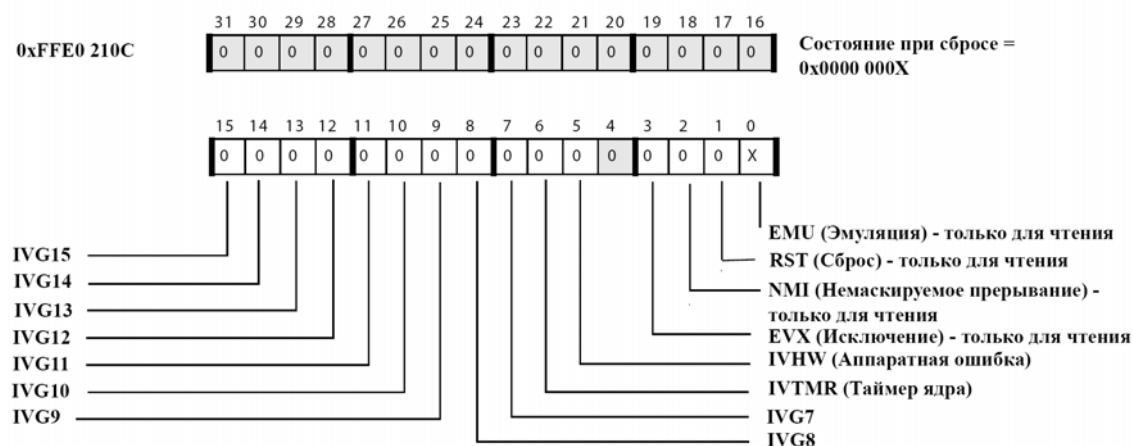


Рис. 4-13. Регистр фиксирования прерываний ядра

## Регистр регистрации прерываний ядра, ожидающих обслуживания (IPEND)

Регистр IPEND отслеживает все вложенные прерывания, задействованные в данный момент времени (см. рис. 4-14). Каждый бит регистра IPEND сигнализирует о том, что соответствующее прерывание в текущий момент времени активно или находится на некотором уровне вложения. Чтение данного регистра возможно в режиме Супервизора; запись данного регистра невозможна. Бит  $IPEND[4]$  используется контроллером событий для временного запрещения прерываний при входе в программу обслуживания прерываний и выходе из неё.

Во время обработки события, соответствующий бит регистра IPEND установлен. Младший бит из установленных битов регистра IPEND указывает на прерывание, обслуживаемое в данный момент времени. В любой момент времени регистр IPEND хранит текущее состояние всех вложенных событий.

# Программный автомат

## Регистр регистрации прерываний ядра, ожидающих обслуживания (IPEND)

Только для чтения. Для всех битов, за исключением бита 4: 0 - прерывание не ожидает обслуживания, 1 - прерывание ожидает обслуживания или активно.

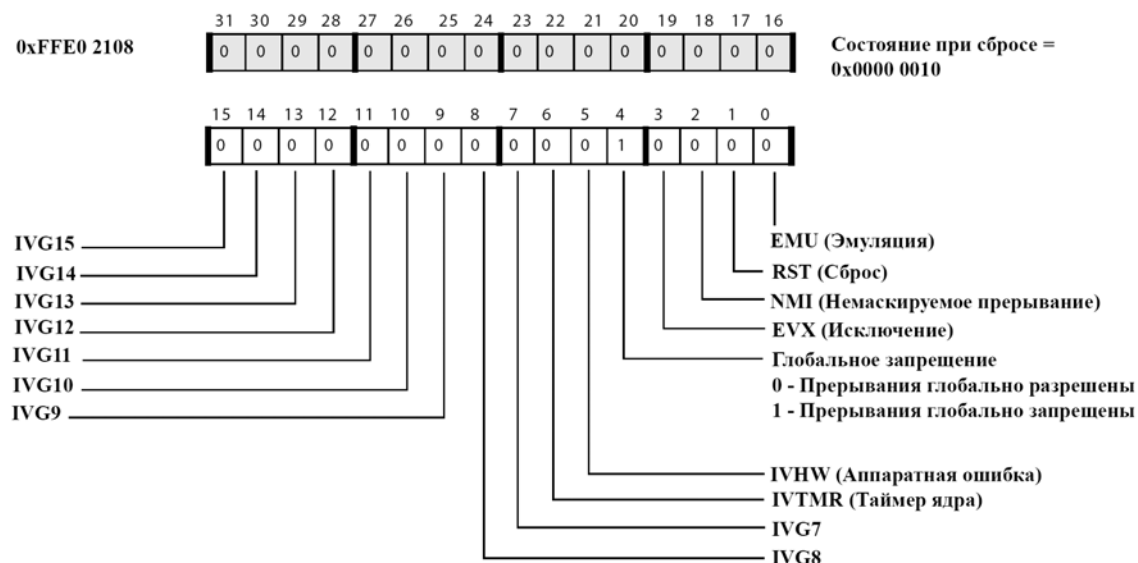


Рис. 4-14. Регистр регистрации прерываний ядра, ожидающих обслуживания

## Глобальное разрешение/запрещение прерываний

Прерывания общего назначения могут быть глобально запрещены при помощи команды `CLI Dreg` и разрешены при помощи команды `STI Dreg`; обе команды доступны только в режиме Супервизора. Сброс, немаскируемое прерывание, эмуляция и исключения не могут быть глобально запрещены. Глобальное запрещение прерываний сбрасывает биты `IMASK[15:5]` после сохранения текущего состояния регистра `IMASK`. Сммотри разделы “Разрешение прерываний” и “Запрещение прерываний” в главе “Управление внешними событиями” *Руководства по набору команд процессора ADSP-BF53x Blackfin*.

Когда код команды слишком критичен ко времени для задержки его выполнения прерыванием, запретите прерывания общего назначения, но убедитесь в их разрешении по завершении выполнения кода.

## Таблица векторов событий

Таблица векторов событий (EVT, Event Vector Table) представляет собой аппаратную таблицу с шестнадцатью 32-разрядными элементами. EVT содержит элементы, соответствующие любому возможному событию ядра. Доступ к элементам EVT осуществляется как к регистрам, отображенным в памяти; в каждый элемент таблицы при сбросе может быть записан адрес соответствующего вектора программы обслуживания прерывания. Когда происходит событие, выборка команд начинается по адресу, содержащемуся в элементе EVT, соответствующем данному событию.

# Программный автомат

Архитектура процессора позволяет назначить каждому из векторов прерываний уникальный адрес; таким образом, вектора прерываний не определяются фиксированным смещением относительного базового адреса таблицы векторов прерываний. Данный подход минимизирует задержку обслуживания, так как при этом нет необходимости совершать длинный переход от таблицы векторов к действительному адресу кода команды обслуживания прерывания.

В таблице 4-9 перечислены события в порядке приоритета. Каждому событию соответствует бит в регистрах состояния событий – ILAT, IMASK и IPEND.

Таблица 4-9. Таблица векторов событий ядра

Номер события	Класс события	Название	Адрес регистра, отображённого в памяти	Примечания
EVT0	Эмуляция	EMU	0xFFE0 2000	Высший приоритет. Адрес вектора обеспечивается интерфейсом JTAG.
EVT1	Сброс	RST	0xFFE0 2004	
EVT2	Немаскируемое прерывание	NMI	0xFFE0 2008	
EVT3	Исключение	EVX	0xFFE0 200C	
EVT4	Зарезервировано	Зарезервировано	0xFFE0 2010	Зарезервированный вектор
EVT5	Аппаратная ошибка	IVHW	0xFFE0 2014	
EVT6	Таймер ядра	IVTMR	0xFFE0 2018	
EVT7	Прерывание 7	IVG7	0xFFE0 201C	
EVT8	Прерывание 8	IVG8	0xFFE0 2020	
EVT9	Прерывания 9	IVG9	0xFFE0 2024	
EVT10	Прерывание 10	IVG10	0xFFE0 2028	
EVT11	Прерывание 11	IVG11	0xFFE0 202C	
EVT12	Прерывание 12	IVG12	0xFFE0 2030	
EVT13	Прерывание 13	IVG13	0xFFE0 2034	
EVT14	Прерывание 14	IVG14	0xFFE0 2038	
EVT15	Прерывание 15	IVG15	0xFFE0 203C	Низший приоритет

## Эмуляция

По событию эмуляции процессор входит в режим Эмуляции, в котором чтение команд происходит по интерфейсу JTAG. Это событие имеет наивысший приоритет среди прерываний ядра.

Более подробную информацию относительно процедуры эмуляции см. в главе 19, “Отладка процессора Blackfin”.

## Сброс

Прерывание сброса (RST) может инициироваться при помощи вывода  $\overline{\text{RESET}}$  или по истечении сторожевого таймера. Данный адрес отличается от адресов, соответствующих другим прерываниям, тем, что его содержимое доступно только

# Программный автомат

для чтения. Запись по этому адресу изменяет значение регистра, но не изменяет адрес, переход к которому процессор выполняет при сбросе. При сбросе процессор всегда выполняет переход по адресу вектора сброса. Дополнительную информацию см. в разделах “Состояние сброса” и “Методы загрузки” в главе 3.

Ядро имеет выходной сигнал, сигнализирующий о том, что произошла двойная ошибка. Состояние двойной ошибки является невозстанавливаемым. Система может быть запрограммирована (при помощи регистра SWRST) на выполнение запроса сброса при обнаружении двойной ошибки. Запрос сброса вызывает системный сброс ядра и периферии.

Вектор сброса определяется системой процессора. Он указывает на начало внутреннего загрузочного ПЗУ или на начало внешней асинхронной памяти, в зависимости от состояния выводов BMODE [1:0]. См. таблицу 4-10.

Таблица 4-10. Адреса вектора сброса.

Источник загрузки	BMODE [1:0]	Начальный адрес выполняемой программы
Загрузочное ПЗУ не используется; команды выполнение из 16-разрядной внешней памяти (банк асинхронной памяти 0)	00	0x2000 0000
Использование загрузочного ПЗУ для выполнения загрузки из 8-разрядной флэш-памяти	01	0xEF00 0000
Зарезервировано	10	0xEF000 0000
Использование загрузочного ПЗУ для конфигурации и выполнение загрузки загрузочной программы из последовательного ПЗУ по SPI (8-, 16- или 24-разрядный диапазон адресов)	11	0xEF00 0000

Если состояние выводов BMODE [1:0] сигнализирует о необходимости загрузки из флэш-памяти или последовательного ПЗУ, вектор сброса указывает на начало внутреннего загрузочного ПЗУ, в котором располагается ядро начальной загрузки небольшого объема. Программа начальной загрузки производит чтение регистра конфигурации системы при сбросе (SYSCR) с целью определения значения выводов BMODE [1:0], задающих соответствующую последовательность загрузки. Информацию о загрузочном ПЗУ процессора см. в разделе «Методы загрузки» в главе 3.

Если контакты BMODE [1:0] сигнализируют о том, что загрузочное ПЗУ не используется, вектор сброса указывает на начало области внешней асинхронной памяти. В данном случае внутреннее загрузочное ПЗУ не используется. Для поддержки чтения из данной области памяти устройство интерфейса внешней шины (EBIU) использует конфигурацию внешней памяти, по умолчанию задаваемую при аппаратном сбросе.

## NMI (Немаскируемое прерывание)

Элемент NMI таблицы векторов событий зарезервирован для немаскируемого прерывания, которое может генерироваться сторожевым таймером или входным сигналом NMI процессора. Примером события, требующего немедленной

# Программный автомат

обработки процессором и соответствующего, таким образом, предназначению немаскируемого прерывания, является предупреждение об отключении питания.

- ⊘ Если в момент обслуживания исключения, немаскируемого прерывания, сброса или эмуляции возникает исключение, вызывается состояние двойной ошибки, и в регистр RETX записывается адрес команды, вызвавшей исключение.

## Исключения

Исключения возникают синхронно с процессом выполнения программы. Другими словами, отдельная команда вызывает исключение при попытке завершения её исполнения. До того, как завершит работу обработчик исключения, команды, следующие за командой, вызвавшей исключение, выполняться не будут.

Многие исключения вызваны обращениями к памяти. Например, исключение возникает при попытке доступа с неверным выравниванием, или когда происходит нарушение защиты или промах в ассоциативном буфере защиты и кэширования. Исключения также возникают при выполнении запрещённых команд или использовании запрещённых комбинаций регистров.

Команда, вызывающая исключение, может либо выполняться, либо не выполняться, перед тем как исключение будет воспринято процессором. Это зависит от того, является ли исключение исключением ошибки или исключением обслуживания.

Команда, вызывающая исключение обслуживания, будет выполнена; в регистр RETX при этом записывается адрес команды, следующей за командой, вызвавшей исключение. Примером исключения обслуживания является однократный шаг при пошаговом выполнении программы.

Команда, вызывающая исключение ошибки не может быть выполнена; в регистр RETX при этом записывается адрес команды, вызвавшей исключение. Примером исключения ошибки является промах в CPLB.

- ⓘ Обычно, регистр RETX содержит корректный адрес возврата. При пропуске выполнения команды, вызвавшей исключение, следует соблюдать осторожность в случаях, когда адрес, следующей за ней команды, не является следующим в линейной последовательности адресов. Это может произойти, когда команда, вызывающая исключение, является последней командой цикла. В этом случае истинным адресом следующей команды будет адрес первой команды цикла.

Каждый раз, когда процессор воспринимает исключение, выполняется запись в поле EXCAUSE[5:0] регистра состояния программного автомата (SEQSTAT), которое указывает на тип произошедшего исключения. Список событий, вызывающих исключения, см. в таблице 4-11.

# Программный автомат

- ⊘ Если в момент обслуживания исключения, немаскируемого прерывания, сброса или эмуляции возникает исключение, вызывается состояние двойной ошибки, и в регистр RETX записывается адрес команды, вызвавшей исключение.

Таблица 4-11. События, вызывающие исключения.

Исключение	EXCAUSE[5:0]	Тип: (E) ошибка (S) обслуживание См. примечание 1.	Примечания/ Примеры
Команда принудительного вызова исключения EXCPT с 4-разрядным полем m	m-поле	S	Команда возвращает 4 разряда EXCAUSE.
Пошаговое выполнение	0x10	S	Когда процессор находится в режиме пошагового выполнения, каждая команда генерирует исключение. В первую очередь используется при отладке.
Исключение, вызванное заполнением буфера трассировки эмуляции	0x11	S	Процессор воспринимает данное исключение при переполнении буфера трассировки (только при разрешенном в регистре управления устройством трассировки переполнении).
Неопределенная команда	0x21	E	Может использоваться для эмуляции команд, не определенных для данной версии процессора.
Запрещенная комбинация команд	0x22	E	См. раздел, описывающий правила использования многофункциональных команд, в <i>Руководстве по набору команд процессора ADSP-BF53x Blackfin</i> .
Нарушение атрибута защиты в CPLB при доступе к данным	0x23	E	Попытка чтения или записи ресурса Супервизора или запрещенный доступ к памяти данных. К ресурсам Супервизора относятся регистры и команды, зарезервированные для использования в режиме Супервизора: регистры, доступные только в режиме Супервизора, все регистры, отображенные в памяти, и команды, доступные только в режиме Супервизора. (Данное исключение вызывается, например, одновременным 32-разрядным обращением к двум регистрам, отображенным в памяти, с использованием двух генераторов адреса данных). Кроме того, данное исключение используется для сигнализирования о нарушении защиты, вызванном недозволённым доступом к памяти, и определяется ассоциативным буфером защиты и кэширования (CPLB) устройства управления памятью (MMU).
Нарушение выравнивания адреса при доступе к данным	0x24	E	Попытка невыровненного доступа к памяти данных или кэшу данных.
Невосстанавливаемое событие	0x25	E	Возникает, например, при генерации исключения во время

# Программный автомат

			обработки предыдущего исключения.
Несовпадение в CPLB при доступе к данным	0x26	E	Используется MMU для сигнализации о несовпадении в CPLB при доступе к данным.
Множественное совпадение в CPLB при доступе к данным	0x27	E	Более чем один элемент CPLB совпадает с адресом выборки данных.
Исключение, вызываемое совпадением с точкой остановки при эмуляции	0x28	E	Произошло совпадение с точкой остановки, когда установлен один из битов EMUSW регистра управления адреса команды точки остановки (WPIACTL).
Нарушение выравнивания адреса при выборке команды	0x29	E	Ошибка выборки команды; например, ошибка чётности шины команд.
Нарушение неправильно выровненного адреса при выборке команды	0x2A	E	Попытка выборки из кэша команд с неправильным выравниванием. При этом исключением адресом возврата, содержащимся в RETx, является неправильно выровненный адрес, а не адрес команды, вызвавшей сбой. Например, если производится попытка косвенного перехода по неправильно выровненному адресу, указываемому в P0, адрес возврата в RETx равен P0, а не адресу команды перехода. (Необходимо отметить, что данное исключение никогда не может быть вызвано переходом относительно счетчика команд; оно может быть вызвано только косвенным переходом.)
Нарушение атрибута защиты в CPLB при выборке команды	0x2B	E	Запрещённый доступ выборки команды (нарушение защиты памяти).
Промех в CPLB при выборке команды	0x2C	E	Промех в CPLB при выборке команды.
Множественное совпадение в CPLB при выборке команды	0x2D	E	Более чем один элемент CPLB совпадает с адресом выбираемой команды.
Запрещённое использование ресурса супервизора	0x2E	E	Попытка использования регистра или команды Супервизора в Пользовательском режиме. К ресурсам Супервизора относятся регистры и команды, зарезервированные для использования Супервизором: регистры, доступные только в режиме супервизора, все регистры, отображенные в памяти, и команды, доступные только в режиме Супервизора.

Примечание 1: Для исключений обслуживания (S) адресом возврата является адрес команды, следующей за исключением. Для исключений ошибок (E) адресом возврата является адрес команды, вызвавшей исключение.

Если выполнение команды приводит к возникновению нескольких исключений, воспринимается только исключение с наивысшим приоритетом. В таблице 4-12 перечислены исключения, упорядоченные в порядке убывания приоритета.

# Программный автомат

Таблица 4-12. Исключения в порядке убывания приоритета.

Приоритет	Исключение	EXCAUSE
1	Невосстанавливаемое событие	0x25
2	Множественное совпадение в CPLB при выборке команды	0x2D
3	Доступ с неправильным выравниванием при выборке команды	0x2A
4	Нарушение защиты при выборке команды	0x2B
5	Несовпадение в CPLB при выборке команды	0x2C
6	Исключение доступа при выборке команды	0x29
7	Совпадение с точкой остановки	0x28
8	Неопределенная команда	0x21
9	Запрещенная комбинация	0x22
10	Запрещенное использование защищенного ресурса	0x2E
11	Множественное совпадение в CPLB при использовании DAG0	0x27
12	Неправильное выравнивание при использовании DAG0	0x24
13	Нарушение защиты при использовании DAG0	0x23
14	Промех в CPLB при использовании DAG0	0x26
15	Множественное совпадение в CPLB при использовании DAG1	0x27
16	Неправильное выравнивание при доступе с использованием DAG1	0x24
17	Нарушение защиты при использовании DAG1	0x23
18	Несовпадение в CPLB при использовании DAG1	0x26
19	Команда EXCPT	m-поле
20	Пошаговое выполнение	0x10
21	Заполнение буфера трассировки	0x11

## Обработка исключений, возникающих при выполнении обработчика исключения

При выполнении обработчика исключения избегайте вызова команд, генерирующих другое исключение. Если исключение вызвано выполнением обработчика исключения, немаскируемого прерывания, программы обслуживания сброса или происходит в режиме эмулятора, выполняются следующие действия:

- Команда, вызвавшая исключение, не выполняется. Не допускается запись результатов команд.
- Вызванное исключение не воспринимается процессором.
- Поле EXCAUSE регистра SEQSTAT обновляется записью кода невосстанавливаемого события.
- В регистре RETX сохраняется адрес команды, вызвавшей сбой. Следует отметить, что если, например, процессор выполнял обработку немаскируемого прерывания, регистр RETN не будет обновлен; адрес команды, вызвавшей исключение, всегда сохраняется в RETX.

Чтобы определить, произошло ли исключение во время работы обработчика исключения, необходимо после её завершения проверить регистр SEQSTAT на наличие кода, указывающего на “невосстанавливаемое событие” (EXCAUSE = 0x25). Если произошло “невосстанавливаемое событие”, регистр RETX содержит адрес последней команды, вызвавшей исключение. Этот механизм предназначен для обнаружения ошибок, а не для их исправления.



## Прерывание аппаратной ошибки

Прерывание аппаратной ошибки сигнализирует об аппаратной ошибке или сбое системы. Аппаратные ошибки происходят, когда логика, внешняя по отношению к ядру, (например, контроллер шины памяти) не может завершить передачу данных (чтение или запись) и устанавливает активный входной сигнал ошибки ядра. Подобные аппаратные ошибки вызывают возникновение прерывания аппаратной ошибки (прерывание IVHW в таблице векторов событий (EVT) и регистрах ILAT, IMASK и IPEND). Программа обслуживания прерывания аппаратной ошибки может определить причину возникновения ошибки путём чтения 5-разрядного поля HWERRCAUSE регистра состояния программного автомата (SEQSTAT) и действовать далее в соответствии с типом ошибки.

Прерывание аппаратной ошибки генерируется следующими событиями:

- ошибка чётности шины;
- условия внутренних ошибок ядра (например, переполнение монитора выполнения);
- прерывание компаратора шины доступа DMA (попытка записи в активный регистр DMA);
- ошибки периферийных устройств;
- ошибки тайм-аута шины

Список поддерживаемых условий возникновения аппаратных ошибок и соответствующих им кодов HWERRCAUSE приведён в таблице 4-13. В поле HWERRCAUSE возвращается код, соответствующий самой последней аппаратной ошибке. Если происходит одновременное возникновение нескольких аппаратных ошибок, только последняя из них может быть распознана и обслужена. Ядро не поддерживает назначение приоритетов, конвейерное обслуживание или организацию очередей для ошибок с разными кодами. Прерывание аппаратной ошибки остаётся активным до тех пор, пока остаётся активным любое из условий возникновения ошибки.

Таблица 4-13. Условия прерываний аппаратной ошибки

Условие исключения	HWERRCAUSE (5 bits)	Примечания / Примеры
Источник исключения – компаратор шины DMA	0b00001	Выход совпадения компаратора напрямую соединён с входом прерывания аппаратной ошибки. Прерывание совпадения компаратора маскируется записью в регистр компаратора управления шиной DMA (DB_CCOMP).
Ошибка доступа к регистру системы, отображённому в карте памяти	0b00010	Ошибка может возникать при обращении к несуществующему адресу в пространстве регистров системы, отображённых в карте памяти, при обращении к 32-разрядному регистру при помощи 16-разрядной команды или при обращении к 16-разрядному регистру при помощи 32-разрядной команды.
Ошибка адресации внешней памяти	0b00011	
Переполнение монитора выполнения	0b10010	
Команда RAISE 5	0b11000	Программа вызывает команду RAISE 5 для вызова прерывания аппаратной ошибки (IVHW).
Зарезервировано	Все остальные битовые комбинации.	

# Программный автомат


## Таймер ядра

Прерывание таймера ядра (IVTMR) возникает, когда значение таймера ядра достигает нуля. См. главу 15, “Таймеры”.

## Прерывания общего назначения (IVG7 – IVG15)

Прерывания общего назначения используются любым событием, требующим внимания процессора. Например, они могут использоваться контроллером DMA для сигнализирования об окончании передачи данных или устройством последовательной передачи данных для сигнализирования об ошибках передачи.

Прерывания общего назначения также могут вызываться программой при использовании команды RAISE. Команда RAISE может принудительно вызывать прерывания IVG15 – IVG7, IVTMR, IVHW, NMI и RST, но не может вызывать исключение и эмуляцию (EVX и EMU, соответственно).

 Рекомендуется зарезервировать два прерывания с самым низким приоритетом (IVG15 и IVG14) для обработчиков программных прерываний.

## Обслуживание прерываний

В СЕС каждому событию соответствует отдельный элемент организации очередности прерываний в виде бита регистра ILAT. Соответствующий бит регистра ILAT устанавливается при обнаружении переднего фронта прерывания (этот процесс занимает два такта тактового сигнала ядра) и сбрасывается при установке соответствующего бита регистра IPEND. Бит IPEND сигнализирует о том, что вектор прерывания поступил в конвейер ядра. На данном этапе СЕС распознаёт и ставит в очередь новое событие для следующего переднего фронта, появляющегося на соответствующем входе прерывания. Минимальная задержка от возникновения переднего фронта прерывания общего назначения до установления выхода IPEND составляет три такта тактового сигнала ядра. Однако, в зависимости от уровня активности ядра и его состояния задержка может составлять намного большее значение.

Для определения необходимости обслуживания прерывания контроллер производит операцию логического И над тремя величинами – ILAT, IMASK и текущим уровнем приоритета процессора.

Обслуживание прерывания с высшим приоритетом включает следующие действия:

1. Вектор прерывания в таблице векторов событий становится адресом следующей выборки.

# Программный автомат

По прерыванию выполнение большинства команд, находящихся в конвейере в данный момент времени, прекращается. При исключении обслуживания прекращается выполнение всех команд, следующих за командой, вызвавшей исключение. При исключении ошибки прекращается выполнение команды, вызвавшей исключение, и всех следующих за ней команд.

2. Адрес возврата сохраняется в соответствующем регистре возврата.

Регистрами возврата являются: RETI для прерываний, RETX для исключений, RETN для немаскируемых прерываний и RETE для отладки в режиме эмуляции. Адресом возврата является адрес команды, следующей за последней выполненной в процессе нормального выполнения программы командой.

3. В соответствии с типом события устанавливается режим процессора.

Если событие является немаскируемым прерыванием, исключением или прерыванием, процессор входит в режим Супервизора. Если событие является исключением эмуляции, процессор входит в режим Эмуляции.

4. Перед началом выполнения первой команды сбрасывается соответствующий бит прерывания в регистре ILAT и устанавливается соответствующий бит в регистре IPEND.

Также, для запрещения всех прерываний до сохранения адреса возврата в регистре RETI, устанавливается бит IPEND [ 4 ].

## Обработка прерываний с вложением и без вложения

Прерывания могут обрабатываться с использованием или без использования вложения.

### Невложенные прерывания

Если вложение прерываний не требуется, обработка прерываний, поступающих в процессе выполнения программы обслуживания прерывания, запрещается. Однако, эмуляция, немаскируемые прерывания и исключения по-прежнему воспринимаются системой.

Когда необходимость поддержки системой вложенных прерываний отсутствует, сохранение в стек адреса возврата, содержащегося в регистре RETI, не требуется. В стеке Супервизора необходимо сохранить только ту часть состояния процессора, которая используется программой обслуживания прерывания. Для возврата из программы обслуживания невложенного прерывания необходимо

# Программный автомат

выполнить только команду RTI, так как адрес возврата уже содержится в регистре RETI.

На рис. 4-15 показан пример обработки прерывания, при котором прерывания глобально запрещаются в течение всего времени выполнения программы обслуживания прерывания.



Рис. 4-15. Обработка невложенного прерывания

## Вложенные прерывания

Если требуется использование вложенных прерываний, адрес возврата к прерванной команде исходной программы обслуживания прерывания, должен быть явно сохранён и затем, по завершении выполнения вложенной программы обслуживания прерывания, восстановлен. Первая команда программы обслуживания прерывания, поддерживающей вложения, должна сохранять адрес, содержащийся в текущий момент времени в регистре RETI, помещая его в стек Супервизора ( $[--SP] = RETI$ ). При этом сбрасывается бит глобального запрещения прерываний IPEND[4], разрешая прерывания. Затем в стек Супервизора сохраняются все регистры, модифицируемые программой обслуживания прерывания. Состояние процессора помещается в стек Супервизора, а не в Пользовательский стек. Таким образом, команды помещения регистра RETI в стек ( $[--SP] = RETI$ ) и извлечения регистра RETI из стека ( $RETI = [SP++]$ ) используют стек Супервизора.

# Программный автомат

На рис. 4-16 иллюстрируется тот факт, что при помещении регистра RETI в стек использование прерываний во время выполнения программы обслуживания снова может быть разрешено. Это приводит лишь к наличию небольшого промежутка времени, в течение которого прерывания будут глобально запрещены.



Рис. 4-16. Обработка вложенного прерывания

## Пример начала программы обслуживания вложенного прерывания

Листинг 4-2. Начало программы обслуживания вложенного прерывания

```

/* Начало программы обслуживания вложенного прерывания.
Выполняется помещение адреса возврата, содержащегося в
регистре RETI в стек Супервизора. Это гарантирует
возобновление обслуживания прерывания с более низким
приоритетом. До данного момента обслуживание прерывания
приостанавливается */
ISR:
[--SP] = RETI; /* Разрешение прерываний и сохранение адреса
возврата в стек */
[--SP] = ASTAT;
[--SP] = FP;
[-- SP] = (R7:0, P5:0);
/* Тело программы обслуживания. Необходимо отметить, что
состояние ни одного из ресурсов процессора (аккумуляторы,
DAG, счётчики и границы циклов) не сохраняется.
  
```

# Программный автомат

Предполагается, что они не используются программой обслуживания прерывания \*/

## Пример окончания программы обслуживания вложенного прерывания

Листинг 4-3. Окончание программы обслуживания вложенного прерывания

```
/*Окончание программы обслуживания вложенного прерывания*/  
/* Восстановление регистра ASTAT, регистров данных и  
указателей. Извлечение RETI из стека Супервизора  
гарантирует, что обработка прерываний, происходящих между  
загрузкой адреса возврата и выполнением команды RTI, будет  
приостановлена */  
(R7:0, P5:0) = [SP++];  
FP = [SP++];  
ASTAT = [SP++];  
RETI = [SP++];  
/* Выполняется команда RTI, по которой производится переход  
по адресу возврата, разрешаются прерывания, и  
осуществляется переход в Пользовательский режим, если  
данное прерывание является последним из обслуживаемых  
вложенных прерываний */  
RTI;
```

По команде RTI выполняется возврат из прерывания. Адрес возврата извлекается из стека и помещается в регистр RETI, что приводит к приостановлению воздействия прерываний от момента восстановления RETI до окончания выполнения RTI. Приостановление воздействия прерываний предотвращает повреждение содержимого регистра RETI следующим прерыванием.

Затем при выполнении команды RTI сбрасывается бит наивысшего приоритета среди установленных битов регистра IPEND. После этого процессор выполняет переход по адресу, на который указывает регистр RETI, и разрешает прерывания, сбрасывая бит IPEND [4].

## Регистрация запросов вложенных прерываний

SIC обнаруживает запросы прерываний периферийных устройств, срабатывающих по уровню. СЕС обеспечивает обнаружение прерываний общего назначения (IVG7 – IVG15) по фронту. Таким образом, SIC формирует синхронный импульс прерывания для СЕС и ожидает поступления от него подтверждения прерывания. Когда прерывание подтверждается ядром (путём установления соответствующего выхода IPEND), SIC генерирует другой синхронный импульс прерывания для СЕС, если периферийное прерывание остаётся активным. При использовании данного подхода система не теряет запросы периферийных прерываний, которые происходят в процессе обслуживания другого прерывания.

# Программный автомат

Так как несколько источников прерываний могут отображаться в одно прерывание общего назначения ядра процессора, до начала или в течение обработки уже обнаруженного прерывания, от SIC на данный вход могут одновременно поступать несколько активных импульсов. В случае использования прерывания, обслуживающего несколько источников, механизм подтверждения прерываний регистром IPEND, описанный ранее, разрешает возобновление обработки всех прерываний источников. Если любой из источников прерывания остаётся активным, SIC генерирует, по меньшей мере, ещё один импульс. Текущее состояние источников совместно обслуживаемых прерываний отображается в регистре состояния прерываний.

## Обработка исключений

Прерывания и исключения по-разному ведут себя по отношению к командам, находящимся в конвейере:

- При возникновении прерывания прекращается выполнение всех команд, находящихся в конвейере.
- При возникновении исключения прекращается выполнение всех команд в конвейере, следующих за командой, вызвавшей исключение. При возникновении исключений ошибки также прекращается выполнение команды, вызвавшей исключение.

Так как исключения, немаскируемые прерывания и эмуляция имеют выделенные регистры возврата, процедура сохранения адреса возврата не является обязательной. Следовательно, команды PUSH и POP при исключениях, немаскируемых прерываниях и эмуляции не влияют на систему прерываний.

Необходимо, однако, отметить, что команды возврата из исключений (RTX, RTN и RTE) сбрасывают младший из установленных битов регистра IPEND.

## Откладывание обработки исключения

Обработчики исключений обычно являются большими программами, так как они должны производить определение причины исключения среди нескольких возможных и предпринимать соответствующие действия по исправлению ошибок. Большие размеры программы обработки исключения могут привести к появлению значительных периодов времени, в течение которых обработка прерываний откладывается.

Во избежание длительного откладывания обработки прерываний, следует создать обработчик исключения таким образом, чтобы он идентифицировал причину возникновения исключения, после чего передавал обработку программе обслуживания прерывания с низким приоритетом. Для вызова обработчика прерывания с низким приоритетом следует использовать команду принудительного вызова прерывания/сброса (RAISE).



# Программный автомат

При передаче обработки исключения прерыванию с низким приоритетом IVG<sub>x</sub>, система должна гарантировать, что вход в программу обслуживания IVG<sub>x</sub> произойдёт раньше, чем возврат к коду прикладного уровня, вызвавшему исключение. При появлении прерывания, ожидающего обслуживания, приоритет которого выше, чем приоритет IVG<sub>x</sub>, допускается вход в программу обслуживания прерывания с высоким приоритетом до входа в программу обслуживания IVG<sub>x</sub>.

## Пример кода обработчика исключения

Листинг 4-4. Программа обработчика исключения с использованием передачи обработки

```
/* Определяется причина исключения при помощи анализа поля
EXCAUSE регистра SEQSTAT (предварительно в стек Супервизора
сохраняется содержимое R0, P0, P1 and ASTAT) */
[--SP] = R0;
[--SP] = P0;
[--SP] = P1;
[--SP] = ASTAT;
R0 = SEQSTAT;
/* Содержимое регистра SEQSTAT маскируется и в R0
оставляется только поле EXCAUSE */
R0 <<= 26;
R0 >>= 26;
/* Производится переход к событию, на которое указывает R0,
используя таблицу переходов EVTABLE */
P0 = R0;
P1 = _EVTABLE;
P0 = P1 + (P0 << 1);
R0 = W[P0] (Z);
P1 = R0;
JUMP (PC + P1);
/* Точка входа события. Здесь обработка передаётся
прерыванию с низким приоритетом IVG15. На данном этапе
также, обычно производится передача параметров. */
_EVENT1:
RAISE 15;
JUMP.S _EXIT;
/* Точка входа события прерывания IVG14 */
_EVENT2:
RAISE 14;
JUMP.S _EXIT;
/* Остальные события закомментированы */
/* В конце работы обработчика происходит восстановление R0,
P0, P1 и ASTAT, и возврат */
_EXIT:
ASTAT = [SP++];
P1 = [SP++];
```



# Программный автомат

```
P0 = [SP++];
R0 = [SP++];
RTX;
_EVTABLE:
.byte2 addr_event1;
.byte2 addr_event2;
...
.byte2 addr_eventN;
/* Таблица переходов EVTABLE содержит 16-разрядные смещения
адресов для каждого события. При использовании смещений код
становится независимым от абсолютного положения в памяти.
При этом также уменьшается объем таблицы
+-----+
| addr_event1 | _EVTABLE
+-----+
| addr_event2 | _EVTABLE + 2
+-----+
| . . .      |
+-----+
| addr_eventN | _EVTABLE + 2N
+-----+
*/
```

## Пример кода программы исключения

В листинге 4-5 представлен примерный шаблон программы исключения, переход к которой может осуществляться из обработчика исключения, описанного ранее.

### Листинг 4-5. Программа исключения

```
[--SP] = RETX; /* Помещение адреса возврата в стек */

/* Здесь следует поместить тело программы исключения. */

RETX = [SP++]; /* Извлечение из стека адреса возврата и
осуществление перехода. */

RTX ; /* Возврат из прерывания. */
```

## Пример использования аппаратных циклов в программе обслуживания прерывания

В следующем примере показан оптимальный метод сохранения и восстановления состояния процессора при использовании в программе обслуживания прерывания аппаратных циклов.

### Листинг 4-6. Сохранение и восстановление при использовании аппаратных циклов

# Программный автомат

lhandler:

<Здесь следует сохранить другие регистры>

```
[--SP] = LC0; /* сохранение информации цикла 0 */
```

```
[--SP] = LB0;
```

```
[--SP] = LT0;
```

<Здесь должен располагаться код обработчика>

```
/* Если в обработчике используется цикл 0, логичным является оставить в конце его работы значение LC0 равным нулю. В обычном режиме это происходит естественным образом при полном выполнении цикла. Если LC0 == 0 восстановление LT0 и LB0 не займёт дополнительных тактов. Если LC0 != 0, при его извлечении каждая из таких операций повлечёт 10 тактов задержки, требуемой на повторную выборку. Извлечение или запись LC0 всегда вызывает задержку. */
```

```
LT0 = [SP++];
```

```
LB0 = [SP++];
```

```
LC0 = [SP++]; /* Эта операция вызовет задержку в 10 тактов на повторную выборку */
```

<Здесь восстанавливаются другие регистры>

```
RTI;
```

## Дополнительные аспекты использования программного автомата

В следующих разделах описываются дополнительные аспекты использования программного автомата.

### Выполнение команд RTX, RTN и RTE событием с низким приоритетом

Команды RTX, RTN и RTE предназначены для осуществления возврата из исключения, немаскируемого прерывания или события эмуляции, соответственно. Перечисленные команды не следует использовать для возврата из события с низким приоритетом. Для возврата из прерывания следует использовать команду RTI. Сбои при использовании несоответствующей команды могут привести к нежелательным результатам.

В случае выполнения RTX, сбрасывается бит IPEND[3]. В случае выполнения RTI сбрасывается бит в IPEND с наивысшим приоритетом.

### Рекомендации по размещению стека системы

Программная модель стека при обработке исключений предполагает невозможность генерации исключения стеком Супервизора одновременно с

# Программный автомат

сохранением состояния процессора в программе обработчика исключения. Однако если размеры стека Супервизора разрастаются до выхода за пределы элемента CPLB или блока SRAM, может возникнуть исключение.

Чтобы гарантировать невозможность генерации исключения стеком Супервизора (невозможности выхода стека за границы элемента CPLB или блока SRAM при выполнении обработчика исключения), необходимо вычислить максимальный размер пространства памяти, занимаемого всеми программами обслуживания прерываний и обработчиком исключения в активном состоянии, и затем выделить необходимое количество памяти SRAM.

## **Задержки при обслуживании событий**

Если при выполнении команд из внешней памяти во время начала операции выборки команды возникает прерывание, его обслуживание в некоторых архитектурах процессоров задерживается до завершения текущей операции выборки. Рассмотрим случай, когда процессор работает с частотой 300 МГц и выполняет код из внешней памяти, время доступа к которой составляет 100 нс. В зависимости от момента возникновения прерывания по отношению к операции выборки команды, задержка выполнения программы обслуживания прерывания может составлять до 30 командных тактов. Если при выборке команды требуется выполнение операции заполнения строки кэша, задержка выполнения программы обслуживания прерывания может достигнуть многих сотен тактов.

Для организации обслуживания высокоприоритетных прерываний с наименьшей возможной задержкой, процессор позволяет проводить завершение операции заполнения строки, вносящей большую задержку, на системном уровне при одновременном выполнении программы обслуживания прерывания из памяти L1. См. рис. 4-17.

Если операция загрузки команды вызывает промах в кэше команд L1 и приводит к выполнению операции заполнения строки, вносящей большую задержку, то при возникновении прерывания задержка его обслуживания до завершения заполнения строки не производится. Вместо этого, процессор выполняет программу обслуживания прерывания в новом контексте, завершая операцию заполнения строки кэша в фоновом режиме.

Необходимо отметить, что программа обслуживания прерывания должна располагаться в кэше или SRAM памяти L1 и не должна генерировать промах в кэше, обращение к памяти L2 или периферии, так как процессор уже занят завершением операции заполнения строки кэша. Если в программе обслуживания прерывания выполняется операция загрузки или сохранения, требующая осуществления одного из перечисленных типов доступа, выполнение программы обслуживания прерывания задерживается до завершения исходного внешнего доступа.

Если выполнение программы обслуживания прерывания завершается до завершения операции загрузки, процессор выполняет останов, ожидая завершения заполнения строки.

# Программный автомат

Процессор действует аналогично при остановках, вызванных чтением из медленной памяти данных или периферийного устройства.

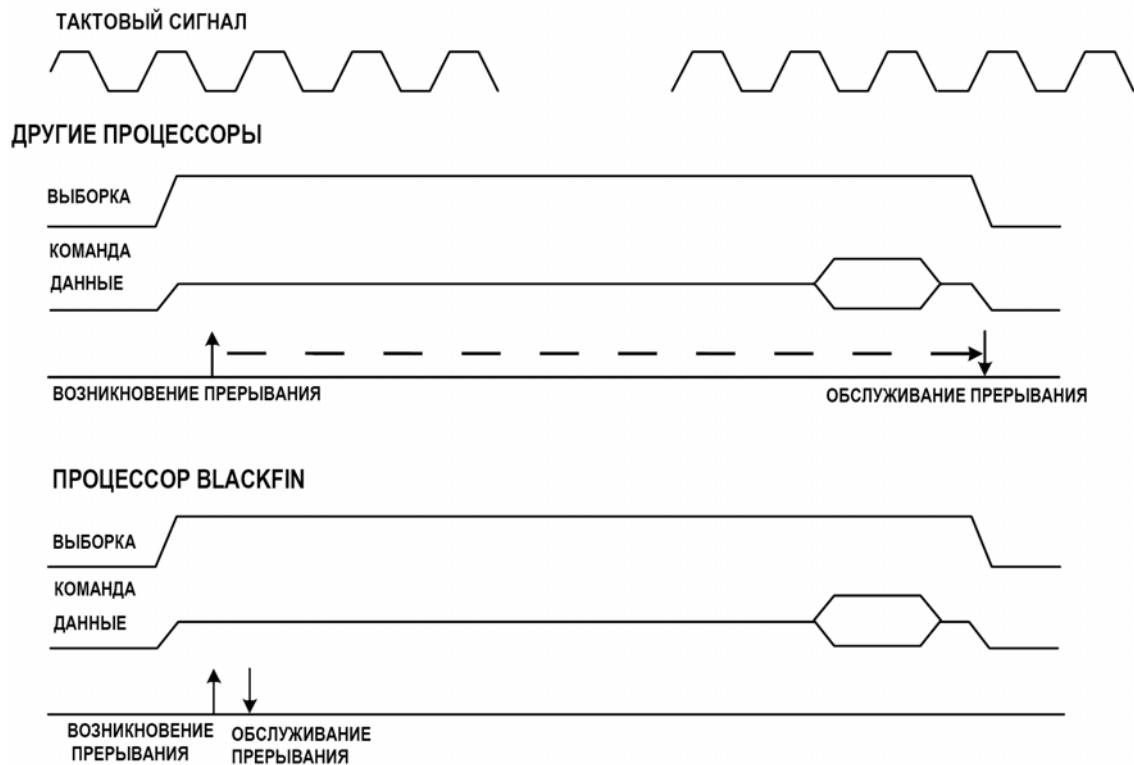


Рис. 4-17. Минимизация задержки при обслуживании прерывания

При записи в медленную память процессор обычно не использует описанную выше процедуру, так как предполагается, что запись происходит за один такт, и данные незамедлительно передаются в буфер записи для последующего выполнения.

Более подробную информацию о структуре памяти и кэша см. в главе 6 "Память".