УПРАВЛЕНИЕ ПРОГРАММОЙ 3

3.1 ОБЗОР

В данной главе предлагается описание программного автомата процессоров семейства ADSP-2100. Программный автомат управляет последовательностью выполнения программы. В нем содержится контроллер прерываний и логическое устройство состояний.

3.2 ПРОГРАММНЫЙ АВТОМАТ

Программный автомат генерирует адреса команд и обеспечивает гибкое управление программой. Он позволяет осуществлять последовательное выполнение команд, обработку прерываний оригинальным образом и реализовать за один цикл команды вызова, условного и безусловного перехода.

На рис. 3.1 приведена блок-схема программного автомата. Каждый функциональный блок программного автомата будет детально рассмотрен далее.

В данной главе, кроме описания логических устройств программного автомата, приводится также информация о командах, которые задают последовательность выполнения программы в процессорах семейства ADSP-2100. К ним относятся следующие команды:

DO UNTIL
JUMP
CALL
RTS *(Return From Subroutine - возвращение в главную программу)*
RTI (*Return From Interrupt - возобновление работы после обслуживания прерывания)*
IDLE

Полное описание каждой команды дано в главе 15, "Набор команд".

3.2.1 Логическое устройство выбора адреса следующей команды

Пока процессор выполняет текущую команду, программный автомат осуществляет предварительную выборку следующей команды. Адрес этой команды выбирается в памяти программы логическим устройством выбора адреса следующей команды из одного из четырех источников:

- инкрементор счетчика команд
- стек счетчика команд
- регистр команд
- контроллер прерываний

3 УПРАВЛЕНИЕ ПРОГРАММОЙ

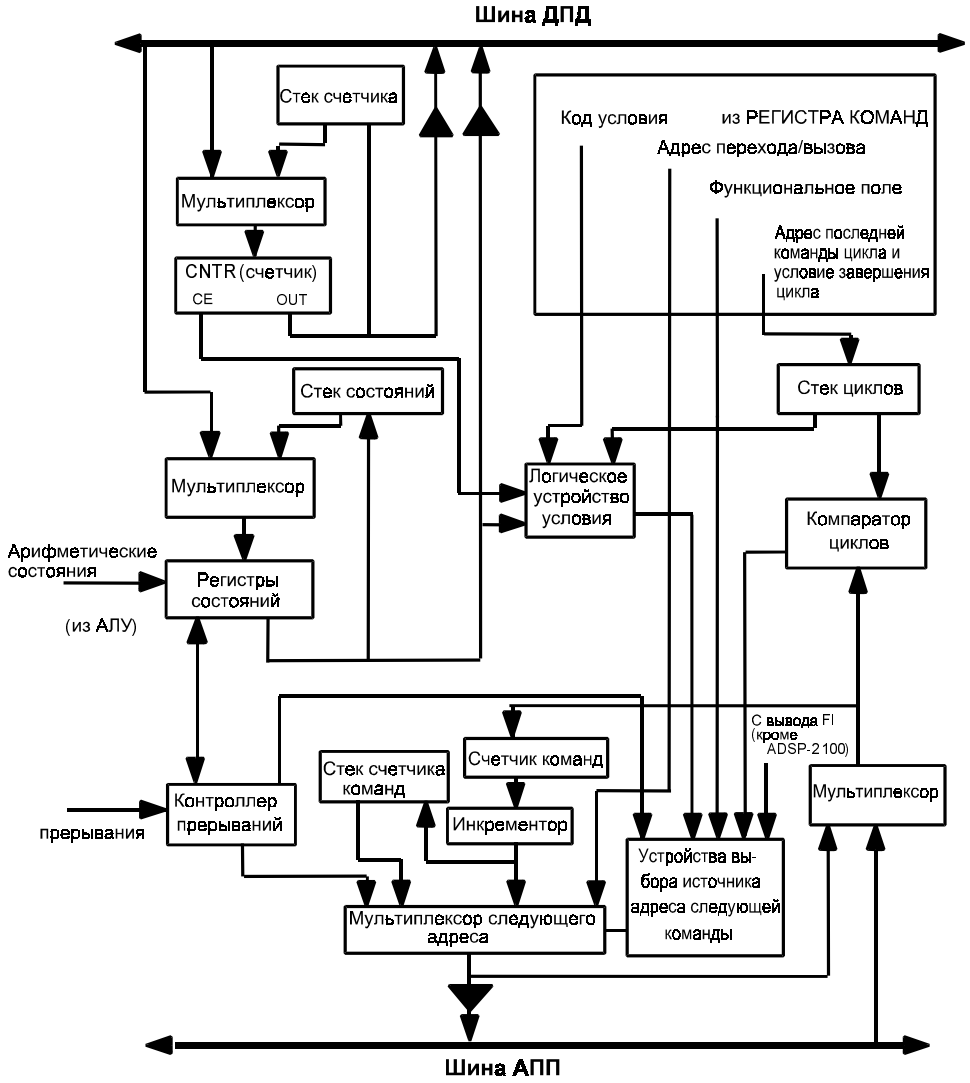

Рис. 3.1 Блок-схема программного автомата

Устройство выбора источника адреса следующей команды (рис. 3.1) позволяет выбрать один из приведенных выше источников, основываясь на данных, поступающих из регистра команд, логического устройства условия,

компаратора циклов и контроллера прерываний. Затем адрес этой команды выводится на шину адреса памяти программы (АПП).

В случае последовательного выполнения программы, а также когда не предпринимается переход или возвращение по условию, или когда цикл DO UNTIL заканчивается, в качестве источника адреса следующей команды выбирается инкрементор счетчика команд. Выходное значение инкрементора выводится на шину АПП и загружается обратно в счетчик команд для начала следующего цикла.

При возвращении из подпрограммы или после окончания обработки прерывания в качестве источника адреса следующей команды выбирается стек счетчика команд. Верхнее значение стека может также использоваться в качестве следующего адреса при возвращении к началу цикла DO UNTIL.

В случае прямой адресации адрес команды, к которой осуществляется переход, берется из регистра команд. Адрес перехода разрядностью 14 бит содержится в командном слове.

При обслуживании прерывания адрес следующей команды в памяти программы берется из контроллера прерываний. После обнаружения прерывания, процессор переходит к ячейке, содержащей вектор прерывания, соответствующий запросу прерывания.

Еще одним источником адреса следующей команды может служить один из индексных регистров (I4-I7) Генератора адреса данных 2 (DAG2). Это происходит при использовании косвенной адресации в такой команде перехода, как:

JUMP (I4);

В этом случае в счетчик команд загружается значение из DAG2 через шину АПП. (Генераторы адреса данных будут рассмотрены в главе 4).

3.2.2 Счетчик команд и стек счетчика команд

Счетчик команд представляет собой 14-битовый регистр, в котором постоянно содержится адрес текущей выполняемой команды. Это значение увеличивается на 1 в 14-разрядном инкременторе. Выходное значение инкрементора может выбираться мультиплексором следующего адреса для выборки следующей последовательной команды.

Счетчик команд связан со стеком, содержащим до шестнадцати 14-битовых слов, в который при выполнении команды CALL помещается значение инкрементора. Стек счетчика команд заполняется также при выполнении команды DO UNTIL и при обработке прерываний. Однако, в случае прерывания, инкрементор игнорируется, а в стек помещается текущее значение

счетчика команд (а не значение, на единицу большее). Таким образом можно возобновить выполнение преждевременно прерванной команды по окончанию обслуживания прерывания. Данные автоматически помещаются и извлекаются из стека счетчика команд во всех вышеназванных случаях. Возможно и непосредственное извлечение данных из стека посредством команды POP.

Для чтения (и непосредственного извлечения), а также записи (и помещения в стек) верхнего значения стека счетчика команд предусмотрена специальная команда, использующая псевдорегистр TOPPCSTACK, описание которого приводится в конце данной главы.

Выходное значение мультиплексора следующего адреса подается обратно в счетчик команд, который, как правило, перезагружает это значение в конце каждого цикла процессора. Однако, в случае косвенной адресации адрес следующей команды выводится на шину АПП Генератором адреса данных DAG2, и счетчик команд загружается непосредственно с шины АПП.

3.2.3 Счетчик циклов и стек счетчика

Счетчик циклов и стек счетчика обеспечивают программный автомат мощным механизмом для организации циклов. Счетчик представляет собой 14-битовый регистр с возможностью автоматического постдекрементирования, управляющий последовательностью циклов программы, которые выполняются заданное число раз. Значения счетчика являются беззнаковыми величинами разрядностью 14 бит.

До начала цикла в счетчик (регистр CNTR) загружаются 14 младших бит с шины ДПД, являющиеся требуемым числом циклов. Действительному числу циклов N противопоставляется значение N - 1. Благодаря логической операции проверки состояния заполненности счетчика CE (счетчик пуст), которая вместе с автоматическим пост-декрементированием значения счетчика осуществляется в конце цикла DO UNTIL, степень заполненности счетчика используется в качестве условия прекращения цикла. Заполненность счетчика проверяется в начале каждого цикла процессора, и значение счетчика уменьшается на 1 в конце; таким образом, когда значение счетчика достигает 1, устанавливается, что счетчик пуст, а значит, цикл выполнен N раз.

Проверка условия CE и автоматическое уменьшение значения счетчика может также выполняться при команде условного перехода, которая проверяет заполненность счетчика. Когда заполненность счетчика проверяется как часть команды возврата по условию или условной арифметической команды, декрементирование счетчика не выполняется.

Содержимое счетчика может непосредственно считываться на шину ДПД в любое время. При этом два старших бита шины ДПД заполняются нулями.

Стек счетчика циклов, содержащий до 4-х слов по 14 бит, позволяет организовывать вложенные циклы за счет временного сохранения номеров циклов. При загрузке нового значения с шины ДПД в счетчик его текущее значение

автоматически помещается в стек счетчика. Каждый раз при выполнении условия CE (счетчик пуст) содержимое стека извлекается автоматически, предполагая, таким образом, выполнение внешнего цикла (если таковой имеется). Для случаев, когда требуется преждевременный выход из цикла, предусматривается возможность непосредственного извлечения содержимого стека счетчика.

При автоматическом помещении данных в стек счетчика существует два исключения. При загрузке счетчика с шины ДПД его содержимое не заносится в стек, если счетчик был пуст, так как в этом случае ячейка стека будет истрачена впустую. Такое состояние возникает после перезапуска системы и при выполнении условия CE, когда стек счетчика пуст. Бит состояния пустого стека счетчика в регистре SSTAT указывает на то, что стек пуст.

Второе исключение представлено в явной форме в специальном синтаксисе OWRCNTR (перезаписать счетчик). При выполнении этой команды в счетчик записывается новое значение, в то время как в стек ничего не помещается. OWRCNTR нельзя считывать (т.е. использовать как исходный регистр) и включать в последнюю команду цикла, организованного командой DO UNTIL.

3.2.4 Компаратор циклов и стек циклов

Команда DO UNTIL инициализирует цикл с нулевыми потерями ресурсов, используя компаратор (сравнивающее устройство) циклов и стек циклов программного автомата.

В каждом цикле компаратор сравнивает следующий адрес, сгенерированный программным автоматом, с адресом последней команды цикла (который содержится в команде DO UNTIL). Адрес первой команды в цикле устанавливается в вершине стека счетчика команд. Когда выполняется последняя команда цикла, процессор реализует операцию условного перехода на начало цикла без каких-либо издержек, которые могут возникнуть при организации цикла другим образом.

В стеке циклов, в котором может записываться до четырех уровней, содержатся адреса последних команд и условия окончания временно задержанных циклов. Единственный дополнительный цикл, связанный с организацией вложенных циклов DO UNTIL, возникает при выполнении самой команды DO UNTIL, так как помещение и извлечение данных из всех стеков производится автоматически. При использовании состояния заполненности счетчика в качестве

условия окончания цикла для начальной загрузки счетчика требуется другой цикл. В таблице 3.1 приводятся условия завершения цикла, которые могут использоваться совместно с командой DO UNTIL.

Таблица 3.1

Логические условия завершения цикла DO UNTIL

Синтаксис	*Условие состояния*	*Истинно, если:*
EQ	Равно нулю	AZ = 1
NE	Не равно нулю	AZ = 0
LT	Меньше нуля	AN.XOR.AV = 1
GE	Больше или равно нулю	AN.XOR.AV = 0
LE	Меньше или равно нулю	(AN.XOR.AV).OR.AZ = 1
GT	Больше нуля	(AN.XOR.AV).OR.AZ = 0
AC	Перенос в АЛУ	AC = 1
NOT AC	Нет переноса в АЛУ	AC = 0
AV	Переполнение в АЛУ	AV = 1
NOT AV	Нет переполнения в АЛУ	AV = 0
MV	Переполнение в умножителе-накопителе	MV = 1
NOT MV	Нет переполнения в умножителе-накопителе	MV = 0
NEG	Операнд X последней команды ABS был отрицателен	AS = 1
POS	Операнд X последней команды ABS был положителен	AS = 0
CE	Счетчик пуст	
FOREVER	Всегда	

При выполнении команды DO UNTIL 14-разрядный адрес последней команды и условие завершения цикла разрядностью 4 бита (и то, и другое содержатся в команде DO UNTIL) помещаются в стек циклов, содержащий 4 слова по 18 бит. Одновременно с этим, выходное значение инкрементора счетчика команд помещается в стек счетчика команд. Так как команда DO UNTIL находится непосредственно перед первой командой цикла, то в стеке счетчика команд содержится адрес первой команды цикла, а в стеке циклов - адрес последней команды цикла и условие его завершения. По заполнению стека циклов активизируется компаратор циклов, который сравнивает верхний адрес в стеке циклов с адресом следующей команды. Когда эти два адреса становятся равными, компаратор циклов уведомляет устройство выбора источника следующего адреса о том, что в следующем цикле будет выполнена последняя команда цикла.

На данном этапе, в зависимости от типа команды в конце цикла, имеется три возможных результата. В первом примере показана наиболеетипичная ситуация. Случаи в примерах 2 и 3 также не исключаются, но для их надлежащего выполнения требуется более сложная программа.

Пример 1

Если последняя команда в цикле не является командой перехода, вызова, возврата или ожидания-простоя, то следующий адрес будет выбран схемой выбора адреса следующей команды на основе условия завершения цикла, хранящегося в вершине стека циклов. Если это условие не выполняется, то выбирается верхний адрес стека счетчика команд, т.е. первая команда цикла. В случае выполнения условия завершения цикла выбирается инкрементор счетчика команд, в результате чего происходит выход из цикла. Затем извлекаются данные из стека циклов, стека счетчика команд и стека счетчика (если он используется).

(Обратите внимание, что условные арифметические команды выполняются на основе условия, выраженного в команде в явной форме, в то время как последовательность циклов управляется (неявным) условием завершения цикла, которое содержится в вершине стека).

Пример 2

Если последней в цикле стоит команда перехода, вызова или возврата, предпочтение отдается выраженной в явной форме команде, а не последовательности цикла, заданной в неявной форме. Если условие, заданное в команде, не выполняется, то имеет место нормальная последовательность выполнения цикла, представленная в примере 1.

При выполнении условия, выраженного в команде, управление программой переходит по адресу перехода/вызова/возврата. Никаких действий, которые обычно имеют место при обнаружении конца цикла, в данном случае не происходит, т.е. нет выбора первой команды цикла, выхода из цикла и помещения данных в стек цикла, счетчика команд и стек счетчика, а также уменьшения значения счетчика.

(Обратите внимание, что при получении команды возврата управление переходит обратно к верхним ячейкам счетчика циклов, так как стек счетчика команд содержит адрес первой команды цикла.)

Пример 3

Если последней командой цикла является команда IDLE (простой-ожидание), то последовательность программы в большей степени управляется командой IDLE, нежели заданным циклом. При выполнении команды IDLE процессор входит в состояние ожидания прерывания, сопровождающееся понижением потребляемой мощности. При получении сигнала

прерывания, выполнение цикла прекращается, а работа программы возобновляется с первой команды следующего цикла.

Примечание:

Повышенная осторожность необходима в случае преждевременного выхода из цикла, а также в случаях, когда цикл заканчивается командами JUMP, CALL, RETURN или IDLE. Так как ни один из механизмов организации циклов не работает во время выполнения команд перехода/вызова/возврата, то в стеках счетчика, циклов и счетчика программ сохраняется неизвлеченная информация, касающаяся организации цикла. Поэтому для восстановления правильного состояния процессора следует непосредственно извлечь эти данные из каждого задействованного стека. При обращении к подпрограмме подобная проблема возникает только тогда, когда команда вызова стоит последней в цикле; в таких случаях по команде возврата осуществляется переход программы к команде, следующей сразу после цикла. Команды вызова подпрограмм, находящиеся внутри цикла и не являющиеся последней командой цикла, обрабатываются согласно процедуре, рассмотренной в примере 1.

Единственным ограничением при организации циклов с помощью команды DO UNTIL является невозможность завершения вложенных циклов на одной и той же команде. Так как компаратор циклов может проверить только одно условие завершения цикла за один такт, то в случаях, когда внешний и вложенный цикл завершаются на одной и той же команде, при инкрементировании счетчика команд может произойти выход из внутреннего вложенного цикла за пределы конечного адреса внешнего цикла.

3.3 КОМАНДЫ УПРАВЛЕНИЯ ПРОГРАММОЙ

В данном разделе рассматриваются основные базовые команды, используемые для управления последовательностью выполнения программы.

3.3.1 Команда перехода (JUMP)

14-разрядный адрес команды, к которой осуществляется переход, содержится в командном слове JUMP. При декодировании команды JUMP этот адрес непосредственно вводится в мультиплексор следующего адреса программного автомата. Адрес выводится на шину АПП и загружается обратно в счетчик команд для следующего цикла. Например, при выполнении команды:

```
JUMP fir_start;
```

происходит переход по адресу с меткой `fir_start`.

3.3.1.1 Косвенная адресация по адресу, взятому из регистра

В случае таких переходов адрес перехода берется из одного из 4-х регистров генератора адреса данных DAG2 (I4, I5, I6 или I7). (Генераторы адреса данных рассматриваются в главе 4). Адрес выводится на шину АПП генератором адреса данных 2 и загружается в счетчик команд во время следующего цикла. Например при выполнении команды:

```
JUMP (I4);
```

выполняется переход по адресу, содержащемуся в регистре I4.

3.3.2 Команда вызова (CALL)

Команда CALL выполняется также, как и команда JUMP. Адрес подпрограммы содержится в командном слове CALL, и после извлечения этого адреса из регистра команд он загружается обратно в счетчик команд для следующего цикла. Кроме того, текущее значение счетчика команд увеличивается на 1 и помещается в стек счетчика команд. После возвращения из подпрограммы работа счетчика и выполнение программы возобновляется с команды, следующей за командой CALL.

3.3.3 Циклы, организованные с помощью команды DO UNTIL

В наиболее распространенной форме циклов DO UNTIL в качестве счетчика итераций используется регистр счетчика (CNTR). При использовании счетчика для управления итерациями цикла завершение цикла DO UNTIL происходит по выполнению условия CE (счетчик пуст). Ниже приведен простой пример цикла подобного типа:

```
L0 = 10;                {установка регистра длины циклического буфера}
I0 = ^data_buffer;      {загрузка указателя с первым адресом}
                        {циклического буфера}
M0 = 1;                 {установка модифицирующего регистра для}
                        {приращения указателя}
CNTR = 10;              {загрузка длины циклического буфера в}
                        {счетчик}
DO loop UNTIL CE;       {задание повторения цикла до обнуления}
                        {счетчика}
    DM(I0,M0) = 0;      {инициализация/очистка циклического буфера}

    ...любая команда...
loop:...любая команда...
```

Когда команда
```
C N T R = 1 0 ;
```

выполняется до начала цикла, счетчик загружается с шины ДПД. Любой имевшийся до этого в счетчике номер цикла одновременно помещается в стек; эта операция не выполняется, если счетчик пуст. Сама по себе команда
```
D O  l o o p  U N T I L  C E ;
```
только задает условия организации цикла. Во время выполнения этой команды не осуществляется никаких других операций. Это происходит только один раз, в начале первого прохождения цикла. При выполнении команды DO UNTIL адрес непосредственно следующей за ней команды помещается в стек счетчика команд (за счет помещения в стек инкрементированного значения счетчика команд). В течении того же цикла в стек циклов помещается адрес команды конца цикла и условие его завершения. По мере продолжения выполнения операций внутри цикла, компаратор сравнивает адрес каждой команды с адресом последней команды цикла. До достижения этого адреса программа выполняется последовательно.

Каждый раз по достижению конца цикла компаратор определяет, что выполняемая команда является последней в цикле и воздействует на логическое устройство выбора адреса следующей команды программного автомата. Вместо выбора в качестве адреса следующей команды инкрементированного значения счетчика команд, проверяется условие завершения цикла. Если это условие не выполняется, то выполнение программы возобновляется с первой команды цикла (в качестве следующего адреса берется верхнее значение стека счетчика команд). Обратите внимание, что значения стеков счетчика команд и циклов не извлекаются, а только считываются.

При последнем прохождении цикла выполняется условие его завершения. При этом извлекается значение, хранящееся в стеке счетчика команд, и выполнение программы возобновляется начиная с команды, следующей непосредственно за последней командой цикла. В последнем цикле данные выталкиваются также из стека циклов и стека счетчика.

3.3.4 Команда ожидания (IDLE)

По команде IDLE процессор входит в состояние неопределенного ожидания в режиме пониженной потребляемой мощности до получения сигнала прерывания. После обслуживания немаскируемого прерывания процессор переходит в нормальный режим работы, и программа выполняется дальше начиная с команды, следующей за командой IDLE.

3.3.4.1 Команда ожидания (IDLE) с замедлением внутренней тактовой частоты процессора

Усовершенствованная версия команды IDLE позволяет замедлять внутреннюю тактовую частоту процессора, уменьшая, таким образом, потребление энергоресурсов. Уменьшенная частота тактовых синхроимпульсов является программируемой дробной частью от нормальной тактовой частоты процессора и задается при помощи выбора соответствующего значения делителя в команде IDLE. Эта команда имеет следующий формат:

```
IDLE (n);
```

где n=16, 32, 64 или 128. При выполнении данной команды процессор остается полностью работоспособным, но работает с меньшей внутренней тактовой частотой. Когда процессор находится в описываемом состоянии, частота всех остальных внутренних тактовых сигналов, например SCLK, CLKOUT, тактовых синхроимпульсов таймера, уменьшается в той же степени, что и внутренняя тактовая частота процессора. При отсутствии значения делителя в команде IDLE по умолчанию выполняется стандартная версия этой команды.

При использовании команды IDLE (n) замедляется внутренняя тактовая частота процессора и, тем самым, увеличивается время ответа процессора на входящие прерывания. Необходимое при стандартном состоянии ожидания для ответа на прерывание время в один цикл увеличивается в *n* (значение делителя) раз. После получения сигнала разрешенного прерывания процессор остается в состоянии ожидания в течение, максимум, n циклов перед тем, как возобновить свою нормальную работу (*n=16, 32, 64 или 128*).

Когда команда IDLE (n) используется в системах, работающих с внешними последовательными тактовыми синхроимпульсами (SCLK), частота последних может превышать внутреннюю тактовую частоту процессора. При таких условиях прерывания не должны генерироваться быстрее, чем они могут быть обслужены с учетом того дополнительного времени, которое требуется процессору для выхода из состояния ожидания (максимум *n* циклов процессора).

3.4 КОНТРОЛЛЕР ПРЕРЫВАНИЙ

При получении сигнала прерывания контроллер прерываний программного автомата передает управление программой команде, расположенной по адресу соответствующего вектора прерывания. Прерывания и соответствующие адреса векторов прерывания для каждого процессора семейства ADSP-2100 приведены в таблицах 3.2-3.7. (Обратите внимание, что SPORT1 можно сконфигурировать либо как последовательный порт, либо как совокупность управляющих выводов, включая два входа внешних прерываний, $\overline{\text{IRQ0}}$ и $\overline{\text{IRQ1}}$. Более подробная информация о конфигурации SPORT1 дана в главе 5 "Последовательные порты".)

3 УПРАВЛЕНИЕ ПРОГРАММОЙ

Вектора прерываний хранятся в памяти программы через четыре ячейки от остальных данных. - Это позволяет кодировать короткие подпрограммы обслуживания прерываний, не переходя к требуемой обслуживающей подпрограмме. В случаях когда подпрограммы обслуживания прерываний содержат более четырех команд, управление программой должно передаваться обслуживающей подпрограмме командой перехода, расположенной по адресу вектора прерывания.

После обслуживания прерывания управление возвращается главной программе при помощи команды RTI (возобновление работы после обслуживания прерывания), в ходе выполнения которой верхнее значение стека счетчика команд выталкивается в счетчик команд; содержимое стека состояний также извлекается для восстановления предыдущего состояния процессора.

Прерывания могут устанавливаться принудительно программными средствами; см. описание регистра IFC ниже.

Благодаря эффективному стеку и программному автомату, обработка немаскируемых прерываний происходит без задержки (кроме задержки, связанной с синхронизацией), даже при прерывании циклов DO UNTIL. Возможность использования вложенных прерываний позволяет высокоприоритетным прерываниям прерывать выполняемые в этот момент подпрограммы обслуживания низкоприоритетных прерываний без каких-либо дополнительных задержек.

В процессорах семейства ADSP-2100 имеется ряд теневых регистров, которые могут использоваться как "свежие" регистры АЛУ, умножителя-накопителя и устройства сдвига во время обслуживания прерываний. Благодаря этой особенности возможно контекстное переключение регистров в течении одного цикла. Использование теневых регистров описывается в разделе "Регистр состояния режима (MSTAT)" данной главы.

3.4.1 Последовательность обработки прерываний

При получении процессором запроса на прерывание, оно фиксируется на время, которое необходимо ему, чтобы закончить выполнение текущей команды. Затем контроллер прерываний сравнивает запрос на прерывание с регистром маски прерывания, IMASK.

В случае немаскируемого прерывания программный автомат помещает текущее значение счетчика команд (в котором содержится адрес следующей команды) в стек счетчика команд. За счет этого после обслуживания прерывания

Таблица 3.2

Прерывания и адреса векторов прерывания для ADSP-2101/2115

Источник прерывания	*Адрес вектора прерывания*
Запуск программы после $\overline{\text{RESET}}$	0x0000
$\overline{\text{IRQ2}}$	0x0004 (*высший приоритет)*
Передача SPORT0	0x0008
Прием SPORT0	0x000C
Передача SPORT1/ $\overline{\text{IRQ1}}$	0x0010
Прием SPORT1/ $\overline{\text{IRQ0}}$	0x0014
Таймер	0x0018 (*низший приоритет)*

Таблица 3.3

Прерывания и адреса векторов прерывания для ADSP-2105

Источник прерывания	*Адрес вектора прерывания*
Запуск программы после $\overline{\text{RESET}}$	0x0000
$\overline{\text{IRQ2}}$	0x0004 *(высший приоритет)*
Передача SPORT1/ $\overline{\text{IRQ1}}$	0x0010
Прием SPORT1/ $\overline{\text{IRQ0}}$	0x0014
Таймер	0x0018 *(низший приоритет)*

Таблица 3.4

Прерывания и адреса векторов прерывания для ADSP-2111

Источник прерывания	*Адрес вектора прерывания*
Запуск программы после $\overline{\text{RESET}}$	0x0000
$\overline{\text{IRQ2}}$	0x0004 *(высший приоритет)*
Запись в ХИП (из хост-машины)	0x0008
Считывание из ХИП (в хост-машину)	0x000C
Передача SPORT0	0x0010
Прием SPORT0	0x0014
Передача SPORT1/ $\overline{\text{IRQ1}}$	0x0018
Прием SPORT1/ $\overline{\text{IRQ0}}$	0x001C
Таймер	0x0020 *(низший приоритет)*

Таблица 3.5

Прерывания и адреса векторов прерывания для ADSP-2171

Источник прерывания	*Адрес вектора прерывания*
Запуск программы после $\overline{\text{RESET}}$ (или после выхода из режима пониженной мощности, когда PUCR=1)	0x0000 *(высший приоритет)*
Понижение потребляемой мощности (не маскируется)	0x002C
$\overline{\text{IRQ2}}$	0x0004
Запись в ХИП (из хост-машины)	0x0008
Считывание из ХИП (в хост-машину)	0x000C
Передача SPORT0	0x0010
Прием SPORT0	0x0014
программируемое прерывание 1	0x0018
Программируемое прерывание 2	0x001C
Передача SPORT1/ $\overline{\text{IRQ1}}$	0x0020
Прием SPORT1/ $\overline{\text{IRQ0}}$	0x0024
Таймер	0x0028 *(низший приоритет)*

Таблица 3.6

Прерывания и адреса векторов прерывания для ADSP-2181

Источник прерывания	*Адрес вектора прерывания*
Запуск программы после $\overline{\text{RESET}}$ (или после выхода из режима пониженной мощности, когда PUCR=1)	0x0000 *(высший приоритет)*
Понижение потребляемой мощности (не маскируется)	0x002C
$\overline{\text{IRQ2}}$	0x0004
$\overline{\text{IRQL1}}$ (по уровню)	0x0008
$\overline{\text{IRQL0}}$ (по уровню)	0x000C
Передача SPORT0	0x0010
Прием SPORT0	0x0014
$\overline{\text{IRQE}}$ (по фронту)	0x0018
Прерывание прямого побайтового доступа к памяти	0x001C
Передача SPORT1/ $\overline{\text{IRQ1}}$	0x0020
Прием SPORT1/ $\overline{\text{IRQ0}}$	0x0024
Таймер	0x0028 *(низший приоритет)*

Таблица 3.7

Прерывания и адреса векторов прерывания для ADSP-21msp58/59

Источник прерывания	*Адрес вектора прерывания*
Запуск программы после $\overline{\text{RESET}}$ (или после выхода из режима пониженной мощности, когда PUCR=1)	0x0000 *(высший приоритет)*
Понижение потребляемой мощности (не маскируется)	0x002C
$\overline{\text{IRQ2}}$	0x0004
Запись в ХИП (из хост-машины)	0x0008
Считывание из ХИП (в хост-машину)	0x000C
Передача SPORT0	0x0010
Прием SPORT0	0x0014
Передача данных в аналоговой форме (ЦАП)	0x0018
Прием данных в аналоговой форме (АЦП)	0x001C
Передача SPORT1/ $\overline{\text{IRQ1}}$	0x0020
Прием SPORT1/ $\overline{\text{IRQ0}}$	0x0024
Таймер	0x0028 *(низший приоритет)*

выполнение главной программы возобновляется, начиная со следующей команды. Программный автомат помещает также содержимое регистров ASTAT, MSTAT, IMASK в стек состояний. Содержимое регистров ASTAT, MSTAT и IMASK хранится в названном порядке. Первым помещается самый старший бит регистра ASTAT и так далее. После помещения содержимого регистра IMASK в стек в этот регистр автоматически загружается новое значение, которое определяет, разрешается ли использование вложенных прерываний (на основе значения бита разрешения использования вложенных прерываний в ICNTL).

Затем процессор выполняет команду NOP (нет операций), одновременно выбирая команду, расположенную по адресу вектора прерывания. По возвращению из подпрограммы обслуживания прерываний извлекается содержимое стеков счетчика команд и состояний, и выполнение программы возобновляется со следующей команды главной программы.

3.4.2 Выбор конфигурации прерываний

Для задания конфигурации прерываний задействованы следующие регистры:

• IMASK - Каждое отдельное прерывание (внешнее и внутреннее) маскируется или разрешается.

3 УПРАВЛЕНИЕ ПРОГРАММОЙ

• ICNTL - Каждое внешнее прерывание конфигурируется по фронту либо по уровню; определяется возможность использования вложенных прерываний.
• IFC - Осуществляет принудительное прерывание или сброс задержанного чувствительного по фронту прерывания.

Прерывания $\overline{IRQ2}$, $\overline{IRQ1}$, $\overline{IRQ0}$ могут быть сконфигурированы по уровню или по фронту соответствующими установками в регистре ICNTL. Процессор ADSP-2181 имеет три дополнительных вывода для прерываний: $\overline{IRQE}$, $\overline{IRQL1}$ и $\overline{IRQL2}$. Вход $\overline{IRQE}$ сконфигурирован по фронту, в то время как входы $\overline{IRQL1}$ и $\overline{IRQL2}$ сконфигурированы по уровню.

При обработке сконфигурированных по фронту прерываний $\overline{IRQx}$ запрос на прерывание фиксируется каждый раз, когда уровень сигнала на выводе, на который приходят прерывания, изменяется от высокого к низкому. Фиксация запроса сохраняется до окончания обслуживания прерывания, а затем он автоматически сбрасывается. Задержанные прерывания по фронту могут также сбрасываться программными средствами путем установки соответствующего бита сброса в регистре IFC.

Сконфигурированные по фронту прерывания требуют, как правило, меньше внешних аппаратных средств, чем прерывания по уровню, что позволяет использовать в качестве прерываний такие сигналы, как синхроимпульсы от генератора частоты дискретизации.

Сконфигурированные по уровню прерывания должны оставаться подтвержденными до окончания обслуживания прерывания. Затем устройство, пославшее запрос на прерывание, должно прекратить подтверждать свой запрос, чтобы это прерывание не обрабатывалось повторно. Однако, если входные сигналы сконфигурированы по уровню, один и тот же входной сигнал может использоваться несколькими источниками прерывания за счет их логического комбини-рования, что позволяет получить единый запрос на прерывание. Сконфигурированные по уровню прерывания не фиксируются.

Программное обеспечение также может определять возможность использования вложенных прерываний. В режиме, не разрешающем использование вложенных прерываний, все запросы на прерывания автоматически маскируются и выводятся при входе в подпрограмму обслуживания прерываний. В режиме, разрешающем использование вложенных прерываний, процессор определяет и обслуживает прерывания с высшим приоритетом.

Прерывания, сгенерированные ХИП (портом интерфейса хост-машины) процессоров ADSP-2111, ADSP-2171 и ADSP-21msp58/59 имеют два уровня

маскирования. Отображенный в карте памяти регистр HMASK конфигурирует маскирование сгенерированных прерываний записи или считывания для каждого регистра данных ХИП. Регистр IMASK может быть установлен на маскирование или разрешение обслуживания всех прерываний считывания с ХИП или записи ХИП. Как регистр IMASK, так и регистр HMASK должны быть установлены для обслуживания прерываний, генерируемых регистрами HDR ХИП. Более подробная информация дается в главе 7 "Порт интерфейса хост-машины".

3.4.2.1 Регистр управления прерываниями (ICNTL)

ICNTL представляет собой 5-разрядный регистр, который конфигурирует внешние запросы на прерывание ($\overline{IRQx}$) каждого процессора. После перезапуска процессора все биты в ICNTL находятся в неопределенном состоянии. Описание установки бит регистра ICNTL для каждого процессора семейства приводится в Приложении E "Управляющие регистры и регистры состояния".

В регистре ICNTL имеется бит, определяющий, срабатывает ли внешнее прерывание $\overline{IRQx}$ по фронту или по уровню (0 = по уровню, 1 = по фронту). Для внутренне сгенерированных прерываний такого бита не предусмотрено.

Бит 4 в регистре ICNTL определяет, разрешается ли использование вложенных прерываний.

При изменении значения регистра ICNTL конфигурация прерывания изменяется с задержкой в один цикл.

3.4.2.2 Регистр маскирования прерываний (IMASK)

Каждый бит регистра IMASK разрешает или блокирует обслуживание определенного прерывания. Описание отдельных битов в регистре IMASK каждого процессора рассматриваемого семейства можно найти в Приложении E "Регистры управления/состояния". Биты маскирования несут положительную информацию (0 = маскирование, 1 = разрешение). После перезапуска процессора регистр IMASK обнуляется.

В процессорах ADSP-2171, ADSP-2181 и ADSP-21msp58/59 все прерывания автоматически запрещаются на время одного командного цикла после выполнения команды, которая модифицирует значение IMASK. Это не влияет на автобуферизацию последовательного порта и передачу данных при прямом доступе к памяти.

Если запрос на прерывание по фронту приходит, когда прерывание является маскируемым, этот запрос фиксируется, но не обслуживается; затем это прерывание может быть опознано программными средствами и позднее обслужено.

При переходе к подпрограмме обслуживания прерываний содержимое регистра IMASK автоматически помещается в стек состояний, из которого оно извлекается после возврата в главную программу. Конфигурация регистра IMASK после входа в подпрограмму обслуживания прерываний определяется битом разрешения вложенности прерываний (бит 4) регистра ICNTL; хотя, она может изменяться как часть самой подпрограммы обслуживания прерываний.

Когда вложенные прерывания заблокированы, прерывания всех уровней автоматически маскируются (IMASK устанавливается на 0) при входе в подпрограмму обслуживания прерываний.

При разрешении вложенных прерываний регистр IMASK устанавливается таким образом, что маскируются только прерывания с равным и более низким приоритетом. Прерывания с высшим приоритетом сохраняют свою конфигурацию. Состояние регистра IMASK представлено графически в Табл. 3.8 на примере ADSP-2101.

3.4.2.3 Глобальное разрешение/блокирование прерываний

В процессорах ADSP-2171, ADSP-2181 и ADSP-21msp58/59 предусмотрены глобальные команды разрешения и блокирования прерываний:

```
ENA INTS;
DIS INTS;
```

После перезапуска процессора прерывания разрешены по умолчанию. По команде DIS INTS маскируются все прерывания (включая понижение потребляемой мощности) независимо от содержимого регистра IMASK. По команде ENA INTS все немаскируемые прерывания могут быть снова обслужены.

Блокирование прерываний не влияет на автобуферизацию последовательного порта.

3.4.2.4 Регистр принудительной установки/сброса прерываний (IFC)

IFC является регистром только для записи и позволяет принудительно устанавливать и сбрасывать срабатывающие по фронту прерывания с помощью программных средств. Принудительная установка прерывания осуществляется

средствами управления программой за счет установки бита принудительного прерывания, соответствующего требуемому прерыванию.

Таблица 3.8

Состояние регистра IMASK при входе в подпрограмму обслуживания прерываний (ADSP-2101)

Бит разрешения вложенных прерываний (в регистре ICNTL) = 0 (вложенные прерывания не разрешены).

Обслуживаемый уровень прерыва-ния	*Содержимое регистра IMASK до входа в подпрограмму об-служивания прерывания (поме-щается в стек)*	*Содержимое регистра IMASK после входа в подпрограмму обслуживания прерывания*
0 (низший)	i j k l m n	0 0 0 0 0 0
1	i j k l m n	0 0 0 0 0 0
2	i j k l m n	0 0 0 0 0 0
3	i j k l m n	0 0 0 0 0 0
4	i j k l m n	0 0 0 0 0 0
5 (высший)	i j k l m n	0 0 0 0 0 0

Бит разрешения вложенных прерываний (в регистре ICNTL) = 1 (вложенные прерывания разрешены).

Обслуживаемый уровень преры-вания	*Содержимое регистра IMASK до входа в подпрограмму обслуживания прерывания (помещается в стек)*	*Содержимое регистра IMASK после входа в подпрограмму обслуживания прерывания*
0 (низший)	i j k l m n	i j k l m 0
1	i j k l m n	i j k l 0 0
2	i j k l m n	i j k 0 0 0
3	i j k l m n	i j 0 0 0 0
4	i j k l m n	i 0 0 0 0 0
5 (высший)	i j k l m n	0 0 0 0 0 0

(i j k l m n схематически обозначает любую комбинацию нулей и единиц)

Принудительная установка срабатывающих по фронту прерываний производится путем установки соответствующего бита в регистре IFC, что вызывает однократное обслуживание прерывания, если оно не маскируемое. Чтобы принудительно установить внешнее прерывание, оно должно быть сконфигурировано (установкой соответствующего бита в регистре ICNTL) по фронту. Прерывания

таймера, последовательного порта и аналоговые прерывания АЦП/ЦАП также срабатывают по фронту и могут маскироваться, сбрасываться и принудительно устанавливаться.

Задержанные прерывания по фронту могут сбрасываться путем установки соответствующего бита сброса в регистре IFC. Срабатывающие по фронту прерывания автоматически сбрасываются при вызове соответствующей подпрограммы обслуживания прерывания.

Описание отдельных бит регистра IFC для каждого процессора семейства ADSP-2100 приводятся в приложении E "Регистры управления/состояния". Регистры IFC процессоров ADSP-2111, ADSP-2171 и ADSP-21msp58 не имеют бит принудительной установки/сброса для прерываний, приходящих с порта интерфейса хост-машины; прерывания ХИП не могут принудительно устанавливаться и сбрасываться за счет программных средств.

3.4.3 Задержка прерываний

Для прерываний таймера, $\overline{IRQx}$, последовательного порта, ХИП и АЦП/ЦАП задержка с момента, когда происходит прерывание, до выполнения первой команды подпрограммы обслуживания прерываний составляет не менее трех полных циклов. Это положение иллюстрируется на рис. 3.2. При допущении, что время установки и фиксации согласованы, для внутренней синхронизации прерывания требуется два цикла (для выводов прерываний $\overline{IRQx}$).

Команды, выполняемые в течение этих двух циклов, должны быть полностью завершены перед продолжением выполнения программы, включая любые дополнительные циклы, которые могут требоваться в случаях запроса/предоставления шины или ожидания доступа к памяти. Третий цикл задержки необходим для выборки первой команды, хранящейся в ячейке вектора прерывания. Во время этого цикла процессор вместо команды, которая должна была бы выполняться при нормальной работе, выполняет команду NOP. В следующем цикле работа возобновляется с первой команды подпрограммы обслуживания прерывания. Адрес прерванной команды помещается в стек счетчика команд, из которого он будет выбран после завершения подпрограммы обслуживания прерывания.

(Обратите внимание, что такая задержка прерываний таймера относится только к процессорам ADSP-2171, ADSP-2181 и ADSP-21msp58/59. Описание задержки прерываний таймера для процессоров ADSP-2101, ADSP-2105, ADSP-2115, ADSP-2111 дается в следующем разделе).

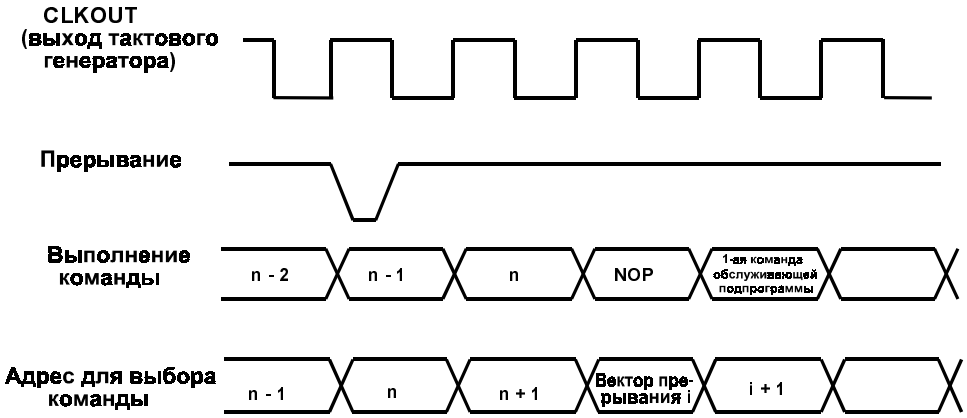

Рис. 3.2 Задержка прерываний (таймера, $\overline{\text{IRQx}}$, последовательного порта, ХИП, АЦП/ЦАП).

Для отложенного маскируемого прерывания задержка с момента выполнения команды, демаскирующей это прерывание (в регистре IMASK), до первой команды подпрограммы обслуживания прерывания составляет один цикл. Эта задержка аналогична задержке, показанной на рис. 3.3 для прерывания таймера процессоров ADSP-2101/2105/2111/2115, причем команда "n" является командой записи данных в регистр IMASK (для демаскирования прерывания).

3.4.3.1 Задержка прерывания таймера в процессорах ADSP-2101, ADSP-2105, ADSP-2115, ADSP-2111

Задержка с момента прихода прерывания до выполнения первой команды обслуживающей программы составляет для прерываний таймера вышеназванных процессоров один цикл, что показано на рис.3.3. Один цикл задержки необходим для выборки команды, хранящейся по адресу вектора прерывания.

3.5 РЕГИСТРЫ СОСТОЯНИЙ И СТЕК СОСТОЯНИЙ

Биты состояния и режима процессора находятся во внутренних регистрах и могут считываться с шины ДПД и записываться на нее. Это регистры:
ASTAT Регистр арифметических состояний
SSTAT Регистр состояния стеков (*может только считываться)*
MSTAT Регистр состояния режима
ICNTL Регистр управления прерываниями

3 УПРАВЛЕНИЕ ПРОГРАММОЙ

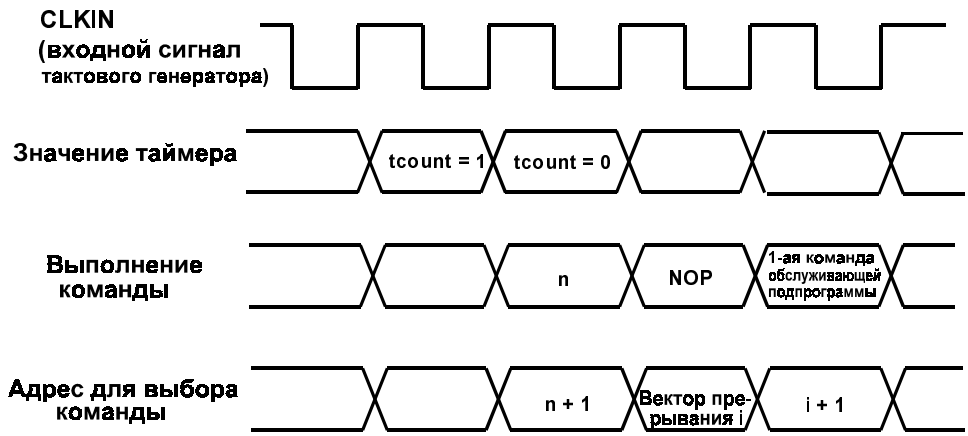

Рис. 3.3 Задержка прерываний таймера для процессоров ADSP-2101, ADSP-2105, ADSP-2115, ADSP-2111.

IMASK Регистр маскирования прерываний

IFC Регистр принудительной установки/сброса прерываний (*может только записываться)*

Регистры описания конфигурации прерываний были рассмотрены в предыдущем разделе. Регистры ASTAT, SSTAT и MSTAT будут представлены в следующих разделах.

Текущие значения регистров ASTAT, MSTAT и IMASK при обработке прерываний помещаются в стек состояний и извлекаются из него по возвращению в главную программу (по команде RTI). Глубина стека различается в различных процессорах. Во всех случаях обеспечивается достаточная глубина стека для организации всех вложенных прерываний.

3.5.1 Регистр состояния арифметических устройств (ASTAT)

Регистр ASTAT имеет разрядность 8 бит и содержит информацию о состоянии вычислительных устройств процессора. Биты регистра ASTAT показаны на рис. 3.4. Биты, определяющие условия AZ, AN, AV, AC, MV несут положительную информацию (1 = истинно, 0 = неистинно).

Каждый из показанных битов автоматически обновляется при генерировании арифметической командой нового состояния. Каждый бит изменяется только под влиянием своего поднабора арифметических операций, приведенных в таблице на следующей странице.

Бит состояния	*Обновляется*
AZ, AN, AV, AC	Любой операцией АЛУ, кроме DIVS, DIVQ
AS	Операцией нахождения абсолютного значения АЛУ (ABS)
AQ	Операциями деления АЛУ (DIVS, DIVQ)
MV	Любой операцией умножителя, кроме насыщения MR (SAT MR)
SS	Операцией нахождения порядка (EXP) устройства сдвига

Арифметическое состояние фиксируется в регистре ASTAT в конце цикла, в котором оно было сгенерировано, и не может использоваться до начала следующего цикла.

Загрузка любого регистра ввода или вывода АЛУ, умножителя-накопителя или устройства сдвига с шины ДПД не влияет ни на один из битов арифметических состояний. При выполнении арифметической команды АЛУ PASS биты AZ и AN устанавливаются для данного операнда X или Y, а содержимое AC сбрасывается.

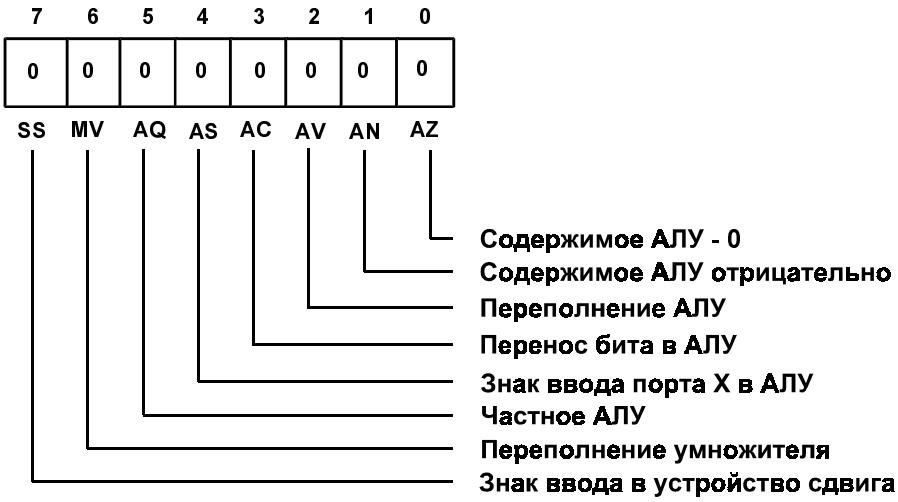

Рис. 3.4 Регистр ASTAT

3.5.2 Регистр состояния стеков (SSTAT)

Регистр SSTAT имеет разрядность 8 бит и содержит информацию о четырех стеках процессора. На рис. 3.5 показаны биты регистра SSTAT. Все биты несут положительную информацию (1 = истинно, 0 = неистинно).

Биты, содержащие информацию об обнулении стеков, указывают на то, что с момента последнего (пере)запуска процессора число операций по извлечению данных из определенного стека превышает или равно числу операций

по помещению информации в этот стек. Биты, содержащие информацию о переполнении стеков, указывают на то, что число операций по помещению данных в стек превысило число операций по извлечению данных из этого стека на количество, большее, чем общая глубина стека. В таких случаях, последние помещенные в стек значения будут утеряны.

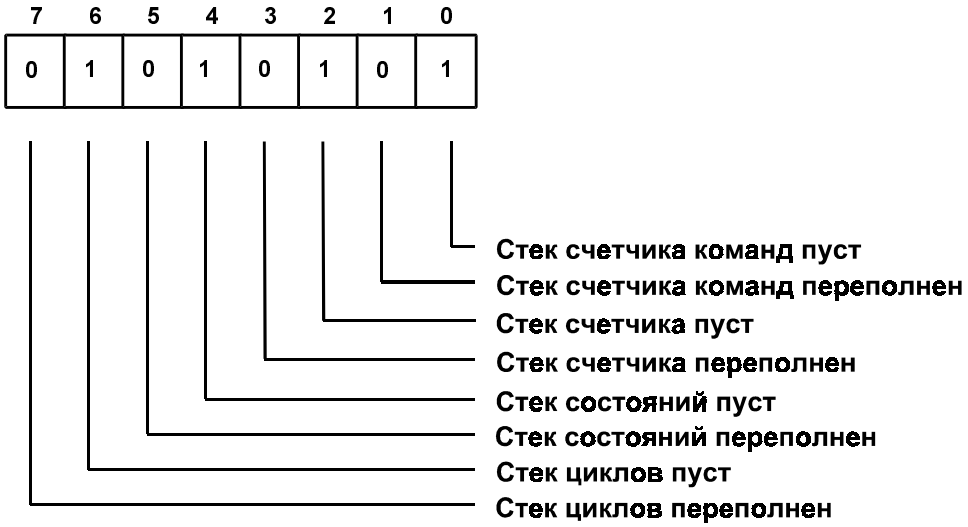

Рис. 3.5 Регистр SSTAT (только считывание)

Так как переполнение стека означает постоянную потерю информации, то биты, указывающие на состояние переполнения стека, "застывают" после своей установки и сохраняются независимо от последующих операций по извлечению информации из стека. В таких случаях возможно положение, когда для данного стека установлены как бит переполнения стека, так и бит, указывающий, что этот стек пуст.

Предположим, например, что стек счетчика, в котором имеется четыре ячейки, переполняется при пяти последовательных операциях по помещению данных в этот стек. После пяти последовательных операций по извлечению данных из стека стек снова станет пуст, но условие переполнения при этом сохранится. Для сброса состояния переполнения стека следует перезапустить процессор.

3.5.3 Регистр состояния режима (MSTAT)

Регистр MSTAT определяет операционный режим процессора. Описание бит регистра MSTAT приводится на рис. 3.6 на следующей странице.

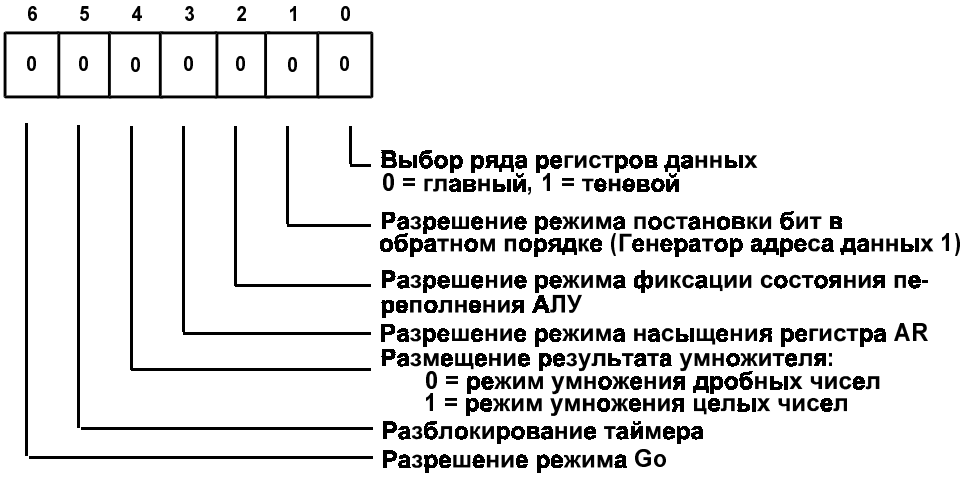

Рис. 3.6 Регистр MSTAT

Содержимое регистра MSTAT модифицируется при записи в него нового значения при помощи команды MOVE. В отличие от других регистров состояний, содержимое MSTAT может также изменяться командой управления режимом (ENA, DIS). Команды управления режимом представляют собой высокоуровневый метод задания операционных режимов процессора, который обладает возможностью самодокументирования. Более подробную информацию можно найти в описании команды управления режимом в главе 15, "Набор команд".

Например, для разрешения режима постановки бит в обратном порядке, может использоваться следующая команда:

```
ENA BIT_REV;
```

При разрешении режима постановки бит в обратном порядке все адреса, генерируемые Генератором адреса данных DAG1, поразрядно ставятся в обратном порядке. Такая перестановка оказывается полезной при вводе или выводе данных для реализации алгоритма БПФ.

В процессорах семейства ADSP-2100 предусмотрен теневой набор регистров, которые могут в любой момент использоваться как "свежие" регистры АЛУ, умножителя-накопителя и устройства сдвига, например, во время выполнения подпрограммы. Бит выбора набора регистров данных в регистре MSTAT определяет активный набор регистров (0 = основной, 1 = теневой). Набор теневых регистров дублирует все регистры ввода и результатов вычислительных устройств:

```
AX0   MX0   SI
AX1   MX1   SE
AY0   MY0   SB
```

3 УПРАВЛЕНИЕ ПРОГРАММОЙ

AY1	MY1	SR1
AF	MF	SR0
AR	MR0	
	MR1	
	MR2	

Например, следующая команда управления режимом переключает процессор с главного набора регистров на работу теневого набора регистров.

```
E N A  S E C _ R E G ;
```

а команда ниже переключает процессор обратно в режим работы основного ряда регистров:

```
D I S  S E C _ R E G ;
```

В режиме фиксации состояния переполнения в АЛУ бит состояния AV "застывает", если он раз установился. В этом режиме бит AV устанавливается в случае переполнения и "удерживается" даже тогда, когда в ходе последующих операций АЛУ переполнение не генерируется. Бит AV можно очистить только записав в него 0.

При разрешении режима насыщения регистра AR этот регистр насыщается до максимально положительного (0x7FFF) или отрицательного (0x8000) значения каждый раз, когда происходит переполнение в АЛУ.

Режим размещения результата умножителя определяет формат его работы: дробный или целочисленный, - которые подробно рассматривались в главе 2, "Вычислительные устройства".

После установки бита активизации таймера тот начинает операцию декрементирования. Сброс этого бита останавливает работу таймера.

При разрешенном режиме GO процессор может продолжать выполнение команд из внутренней памяти программы во время предоставления шины. Процессор может остановить работу, ожидая, когда освободятся шины, только в случае, когда требуется доступ к внешней памяти. При запрещении режима GO процессор всегда будет останавливать работу во время предоставления шины

3.6 УСЛОВНЫЕ КОМАНДЫ

Логическая схема программного автомата определяет, выполняется ли условная команда, например, команда условного перехода или вызова, или какая-либо арифметическая операция. Эта схема также управляет скрытой последовательностью циклов на основе условия продолжения цикла наверху стека циклов.

Устройство условной логики получает "сырую" информацию о состоянии вычислительных устройств из регистра ASTAT и вычитающего счетчика и выделяет из этой информации набор из шестнадцати составных условий состояния. В таблице 3.9 приводятся условия состояний и соответствующий им синтаксис языка ассемблер. Эти условия состояния используются вместе с оператором *IF условие,* который может входить в некоторые команды. Кроме того, в качестве условия команд JUMP и CALL может использоваться состояние на выводе FI (Flag In).

Таблица 3.9

Логика условных команд, заданных оператором IF

Синтаксис	*Условие состояния*	*Истинно, если:*
EQ	Равно нулю	AZ = 1
NE	Не равно нулю	AZ = 0
LT	Меньше нуля	AN.XOR.AV = 1
GE	Больше или равно нулю	AN.XOR.AV = 0
LE	Меньше или равно нулю	(AN.XOR.AV)OR.AZ = 1
GT	Больше нуля	(AN.XOR.AV).OR.AZ = 0
AC	Перенос в АЛУ	AC = 1
NOT AC	Нет переноса в АЛУ	AC = 0
AV	Переполнение в АЛУ	AV = 1
NOT AV	Нет переполнения в АЛУ	AV = 0
MV	Переполнение в умножителе-накопителе	MV = 1
NOT MV	Нет переполнения в умножителе-накопителе	MV = 0
NEG	Операнд X последней команды ABS был отрицателен	AS = 1
POS	Операнд X последней команды ABS был положителен	AS = 0
NOT CE	Счетчик не пуст	----
FLAG_IN*	Значение на выводе FI	Последний значение на выводе FI = 1
NOT FLAG_IN*	Значение на выводе FI	Последний значение на выводе FI = 0

3.7 Псевдорегистр TOPPCSTACK

Для чтения (и извлечения из стека) и записи (и помещения в стек) верхнего значения стека счетчика команд используется специальная версия ко-

* Доступны только с командами JUMP и CALL.

манды пересылки данных между регистрами. При выполнении обычной команды извлечения данных из стека счетчика команд, POP PC, извлеченное из стека значение не сохраняется; чтобы сохранить это значение в каком-либо регистре, следует использовать следующую команду:

```
reg=TOPPCSTACK;   {извлекает данные из стека}
                  {счетчика команд в регистр}
                  {"toppcstack" может быть на}
                  {брано малыми буквами}
```

Извлечение данных из стека счетчика команд производится по данной команде после задержки в один цикл. Обычно, для того чтобы извлечение данных было произведено правильно, после этой особой команды помещается команда NOP:

```
reg=TOPPCSTACK;
NOP;              {позволяет правильно произ}
                  {вести извлечение данных из}
                  {стека}
```

Какой-либо особой команды для помещения данных в стек счетчика команд не существует. Для того, чтобы поместить определенное значение в стек счетчика команд, используется следующая команда:

```
TOPPCSTACK=reg;   {содержимое регистра помещается}
                  {в стек счетчика команд}
```

Значение незамедлительно помещается в стек в течение того же цикла.

Примеры:

```
AX0=TOPPCSTACK;   {данные из стека счетчика}
                  {помещаются в регистр AX0}
NOP;

TOPPCSTACK=I7;    {содержимое регистра I7 помеща}
                  {ется в стек счетчика команд}
```

Описание команды пересылки данных между регистрами, Тип 17, приводится в главе 15, "Набор команд". *Обратите особое внимание, что TOPPCSTACK не может использоваться как регистр во всех других типах команд!*

В специальных командах TOPPCSTACK могут использоваться только следующие регистры:

Регистры АЛУ, умножителя-накопителя и устройства сдвига	*Регистры генератора адреса данных*	
AX0	I0	I4
AX1	I1	I5
MX0	I2	I6
MX1	I3	I7
AY0	M0	M4
AY1	M1	M5
MY0	M2	M6
MY1	M3	M7
AR	L0	L4
MR0	L1	L5
MR1	L2	L6
MR	L3	L7
SI		
SE		
SR0		
SR1		

3.7.1 Ограничения на использование псевдорегистра TOPPCSTACK

Ниже приводятся имеющиеся ограничения на использование специальных команд с псевдорегистром TOPPCSTACK.

1) Команды извлечения и помещения данных в псевдорегистр TOPPCSTACK не могут располагаться непосредственно после команды RTI (возвращения из подпрограммы обслуживания прерывания). Между этой командой и командой с псевдорегистром TOPPCSTACK должна стоять команда NOP:

```
reg=TOPPCSTACK;
NOP;                    {позволяет правильно выполнить}
                        {извлечение данных из стека}
RTI;                    {следующее значение извлекает-}
                        {ся из стека автоматически}
```

2) Команда чтения и извлечения данных из стека в псевдорегистр TOPPCSTACK не может быть последней или предпоследней командой в цикле Do Until. Ни команда 1, ни команда 2 в нижеследующем примере не может быть командой чтения/извлечения данных из стека в псевдорегистр TOPPCSTACK:

```
        Do loop UNTIL CE;
                AX0=DM(I5,M5);
                ...
                команда 1;
loop:                   команда 2;
```

3) Внутри любого цикла, организованного командой Do Until, должно быть равное количество операций помещения и извлечения данных из стека, включая как обычные команды извлечения данных из стека счетчика команд, так и специальные команды чтения/извлечения и записи/помещения данных с использованием псевдорегистра TOPPCSTACK.

4) Имеется несколько ограничений на совместное использование команд RTS (возвращения из подпрограммы), RTI (возвращения из подпрограммы обслуживания прерывания) и обычной команды извлечения данных из стека счетчика команд POP PC. Если команда 3 в нижеследующем примере является командой RTS, RTI или POP PC, то

```
команда 1;
команда 2;
команда 3;      {если это RTS, RTI или POP PC...}
```