5 ГЕНЕРАТОРЫ АДРЕСА ДАННЫХ

Генераторы адреса данных (DAGs, Data Address Generators) формируют адреса для пересылок данных в память и из памяти. Формируя адреса, генераторы адреса данных позволяют программам осуществлять косвенное обращение к памяти, используя вместо абсолютного адреса регистры DAG.

Архитектура DAG, показанная на рис. 5-1, поддерживает несколько функций, минимизирующих непроизводительные затраты в процедурах доступа к данным. Эти функции включают:

- Выдача адреса – обеспечивается адрес при доступе к данным.
- Выдача адреса и пост-модификация – обеспечивается адрес при пересылке данных и автоматическое инкрементирование/декрементирование адреса, сохраняемого в регистре и используемого при следующей пересылке.
- Выдача адреса со смещением – обеспечивается адрес, смещённый относительно базового, без инкрементирования исходного указателя адреса.
- Модификация адреса – осуществляется инкрементирование или декрементирование адреса, сохраняемого в регистр без выполнения пересылки данных.
- Бит-реверсная адресация – при пересылке данных обеспечивается бит-реверсный адрес; реверсирование адреса, сохраняемого в регистр, не выполняется.

Подсистема DAG включает два арифметических устройства DAG, девять регистров указателей, четыре индексных регистра и четыре полных набора соответствующих регистров модификации, базового адреса и длины. Эти регистры содержат значения, используемые генераторами адреса данных для формирования адресов. Ниже следует описание этих регистров.

- Индексные регистры, `I[3:0]`. Беззнаковые 32-разрядные индексные регистры являются указателями на адрес в памяти. Например, при выполнении команды `R3=[I0]` в регистр `R3` загружается значение, расположенное в ячейке памяти, на которую указывает регистр `I0`. Индексные регистры могут использоваться при 16- и 32-разрядных обращениях к памяти.
- Регистры модификации, `M[3:0]`. Знаковые 32-разрядные регистры модификации обеспечивают инкремент или шаг, с которым индексный регистр постмодифицируется при пересылке регистра. Например, команда `R0=[I0 ++ M1]` приводит к тому, что DAG:
  - выдает адрес в регистре `I0`;
  - загружает в `R0` содержимое ячейки памяти, на которую указывает регистр `I0`;
  - изменяет содержимое регистра `I0` на величину, содержащуюся в регистре `M0`.
- Регистры длины и базового адреса, `B[3:0]` и `L[3:0]`. Беззнаковые 32-разрядные регистры длины и базового адреса определяют начальный адрес и диапазон адресов циклического буфера. Каждая пара регистров `B` и `L` всегда

используется совместно с соответствующим I-регистром. Например, `I3`, `B3`, `L3`. Дополнительную информацию см. в разделе «Адресация циклических буферов».

- Регистры указателей, `P[5:0]`, `FP`, `USP` и `SP`. 32-разрядные регистры указателей являются указателями на адреса в памяти. Различные команды могут использовать поле `P[5:0]`, регистры `FP` (указатель кадра) и `SP/USP` (указатель стека/указатель пользовательского стека) и манипулировать ими. Например, при выполнении команды `R3=[P0]` в регистр `R3` загружается значение, содержащееся в ячейке памяти, на которую указывает регистр `P0`. Регистры указателей не влияют на адресацию циклических буферов. Они могут использоваться при 8-, 16- и 32-разрядных обращениях к памяти. В целях дополнительной защиты режима регистр `SP` доступен только в режиме Супервизора, а регистр `USP` доступен в Пользовательском режиме.

Не следует полагать, что L-регистры автоматически инициализируются нулевым значением при линейной адресации. После сброса процессора I-, L-, M- и B-регистры содержат случайные значения. Для каждого используемого I-регистра программа должна инициализировать соответствующий L-регистр нулевым значением при линейной адресации или значением длины буфера при адресации циклических буферов.

Следует отметить, что инициализация всех регистров DAG должна выполняться в индивидуальном порядке. Инициализация B-регистра не влечет за собой автоматическую инициализацию I-регистра.

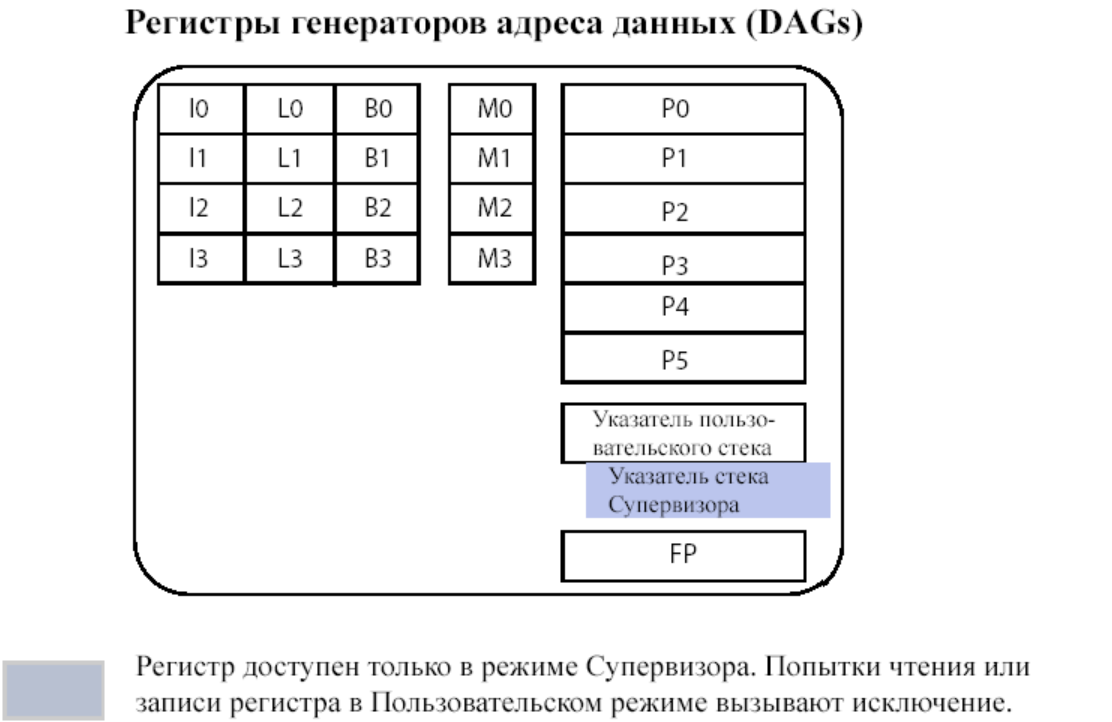

Рис. 5-1. Регистры генераторов адреса данных процессора.

Адресация с использованием генераторов адреса данных

Генераторы адреса данных могут формировать адрес, инкрементируемый непосредственно заданным значением или значением регистра. При адресации с постмодификацией DAG выдает значение I-регистра без изменения, а затем прибавляет к нему значение M-регистра или непосредственно заданное значение.

При индексной адресации DAG прибавляет к значению P-регистра небольшое смещение, но не обновляет P-регистр, обеспечивая, таким образом, доступ к определенной ячейке памяти по смещению. В процессоре реализована байтовая адресация. При любом доступе к данным адрес должен выравниваться в соответствии с разрядностью данных. Другими словами, при выборке 32-разрядного значения адрес должен быть выровнен по 32-разрядной границе, а при сохранении в память 8-разрядного значения адрес может выравниваться по границе любого байта. В зависимости от используемого типа данных значения инкремента и декремента регистров DAG могут составлять 1, 2 или 4 соответственно для 8-, 16- или 32-разрядных обращений к памяти.

Например, рассмотрим следующую команду:

```
R0 = [P3++];
```

При выполнении этой команды выбирается 32-разрядное слово, на которое указывает регистр `P3`, и помещается в регистр `R0`. Затем, для поддержания выравнивания, требуемого при 32-разрядном доступе, регистр `R3` пост-инкрементируется на *четыре*.

```
R0.L = W[I3++];
```

При выполнении этой команды выбирается 16-разрядное слово, на которое указывает регистр `I3`, и помещается в младшую половину регистра-приемника, `R0.L`. Затем, для поддержания выравнивания, требуемого при 16-разрядном доступе, регистр `I3` пост-инкрементируется на *два*.

```
R0 = B[P3++] (Z);
```

При выполнении этой команды выбирается 8-разрядное слово, на которое указывает регистр `P3`, и помещается в регистр `R0`. Затем регистр `P3` пост-инкрементируется на *единицу*. Перед передачей в 32-разрядный регистр данных значение байта может быть дополнено знаковыми разрядами или нулями (соответствует случаю, указанному в примере).

В командах, использующих индексные регистры, в качестве модификаторов используется M-регистр или небольшое непосредственно заданное значение (±2 или 4). В командах, использующих регистры указателей, в качестве модификатора используется небольшое непосредственно заданное значение или другой P-регистр. Подробности см. в таблице 5-3 “Обзор команд DAG”.

Генераторы адреса данных

Указатели кадра и стека

Во многих отношениях применение регистров указателей стека и кадра похоже на применение других P-регистров, `P[5:0]`. Они могут выполнять роль обычных указателей в любых операциях загрузки/сохранения. Например, `R1 = B[SP] (Z)`. Однако, регистры `FP` и `SP` имеют дополнительное функциональное назначение.

Регистры указателей стека включают:

- Указатель пользовательского стека (`USP` при использовании в режиме супервизора, `SP` – в Пользовательском режиме).
- Указатель стека Супервизора (`SP` в режиме Супервизора).

Обращение к регистрам указателя пользовательского стека и указателя стека супервизора осуществляется с использованием альтернативного имени `SP`. В зависимости от текущего режима работы процессора только один из этих регистров активен и доступен по имени `SP`:

- При любом обращении к `SP` в Пользовательском режиме (например, извлечении из стека `R0 = [SP++];`) в качестве эффективного адреса явно используется значение `USP`.
- При таком же обращении к `SP` в режиме Супервизора (например, `R0 = [SP++];`) в качестве эффективного адреса явно используется значение указателя стека Супервизора.

Для манипуляции указателем пользовательского стека в программе, работающей в режиме Супервизора, следует использовать альтернативное имя `USP`. В режиме Супервизора операция пересылки регистра `USP` (например, `R0 = USP;`) передает текущий указатель пользовательского стека в регистр `R0`. Альтернативное имя `USP` может использоваться только в режиме Супервизора.

В некоторых командах загрузки/сохранения `SP` и `FP` используются явным образом. К этим командам относятся:

- Команды загрузки/сохранения с индексацией при помощи регистра `FP`, которые расширяют диапазон адресации операций, кодируемых 16-разрядными словами.
- Команды помещения в стек/извлечения из стека, включая их версии, работающие с набором регистров.
- Команды выделения/удаления памяти (`LINK`/`UNLINK`), управляющие пространством кадра стека (стекового фрейма) и регистром указателя кадра (`FP`) этого пространства.

Адресация циклических буферов

Генераторы адреса данных поддерживают адресацию циклических буферов. Циклические буферы представляют собой диапазон адресов данных, которые DAG проходит циклически, совершая цикл при достижении конечного/начального адреса диапазона.

Для адресации циклических буферов генераторы адреса данных используют четыре типа регистров DAG, работающих следующим образом:

- Индексный регистр (I) содержит значение, вызываемое генератором адреса данных на шину адреса.

- Регистр модификации (M) содержит величину постмодификации (отрицательную или положительную), которую DAG прибавляет к I-регистру в конце каждого обращения к памяти.

  Любой M-регистр может использоваться совместно с любым из I-регистров. Кроме того, в качестве величины постмодификации вместо M-регистра может использоваться непосредственно задаваемое значение. Размер величины модификации не должен превышать длину циклического буфера (значение L-регистра).

- Регистр длины (L) задает размер циклического буфера и диапазон адресов, в котором DAG циклически перемещает значение I-регистра.

  L имеет положительное значение, которое не должно превышать 2^{32}-1. Если значение L-регистра равно нулю, использование циклического буфера запрещается (буфер является линейным).

- Регистр базового адреса (B) или сумма B- и L-регистров задает значение, с которым DAG сравнивает модифицированное значение I-регистра после каждого обращения к памяти.

При адресации циклического буфера DAG перемещает индексный указатель (I-регистр) по элементам буфера, постмодифицируя и обновляя его при каждом обращении к памяти положительным или отрицательным значением, содержащемся в M-регистре.

Если индексный указатель попадает за пределы диапазона адресов буфера, DAG осуществляет его возврат в точку внутри буфера, вычитая из него или прибавляя к нему длину буфера (L-регистр).

Начальный адрес, относительно которого DAG выполняет обращение индексного указателя, называется базовым адресом буфера (B-регистр). При организации циклических буферов, содержащих 8-разрядные данные, ограничения на значение базового адреса отсутствуют. Циклические буферы, содержащие 16- и 32-разрядные данные, должны быть выровнены по 16- и 32-разрядной границе, соответственно. Исключения могут быть сделаны при видео-операциях. Дополнительную информацию см. в разделе «Выравнивание адресов памяти» в главе 5. При организации циклических буферов используется адресация с постмодификацией.

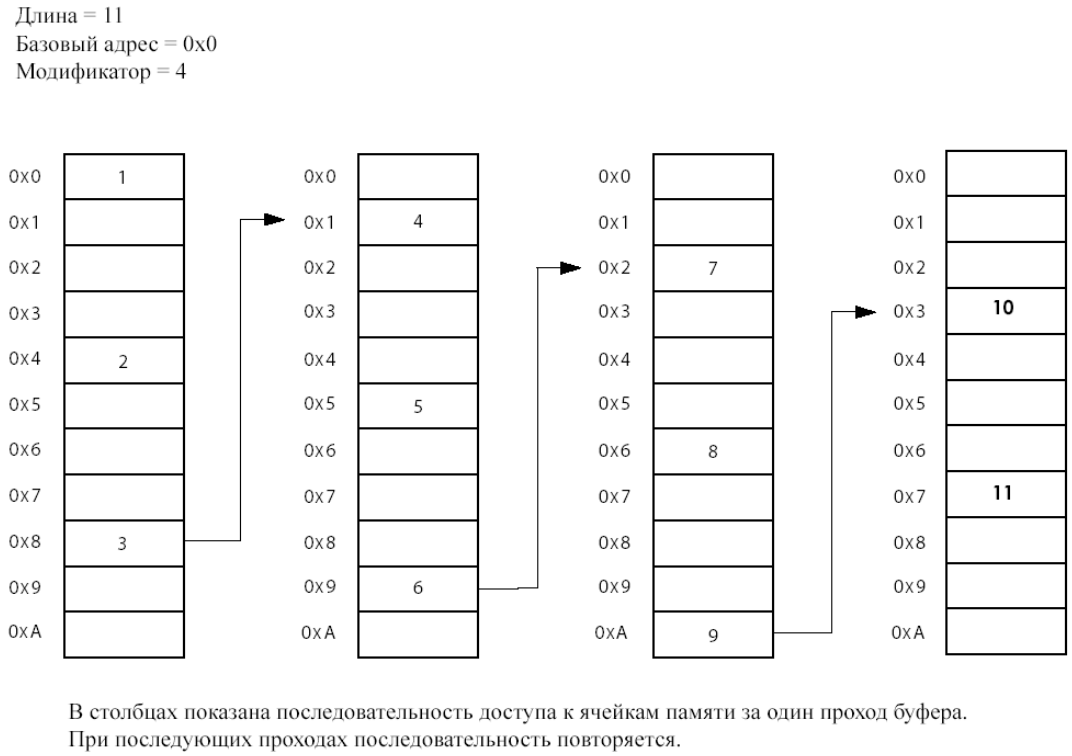

Рис. 5-2. Циклические буферы данных.

Как следует из рис. 5-2, при первом доступе с постмодификацией к буферу DAG выдает на шину адреса значение I-регистра, после чего модифицирует адрес, добавляя к нему величину модификации.

- Если обновленное значение индекса меньше длины буфера, DAG записывает его в I-регистр.
- Если обновленное значение индекса превышает длину буфера, DAG добавляет к нему (при отрицательном значении величины модификации) или вычитает из него (при положительном значении величины модификации) значение L-регистра перед записью в I-регистр.

Операции постмодификации и обращения указателя могут быть описаны следующими выражениями:

- Если значение M положительно:
  $I_{новое} = I_{старое} + M$
  если $I_{старое} + M <$ базовый адрес + длина буфера (конец буфера)
  $I_{новое} = I_{старое} + M - L$
  если $I_{старое} + M \geq$ базовый адрес + длина буфера (конец буфера)
- Если значение M отрицательно:
  $I_{новое} = I_{старое} + M$
  если $I_{старое} + M \geq$ базовый адрес буфера (начало буфера)
  $I_{новое} = I_{старое} + M + L$
  если $I_{старое} + M <$ базовый адрес буфера (начало буфера)

Адресация с бит-реверсией адреса

Для получения результатов в правильной последовательности в программах, реализующих некоторые алгоритмы (в частности, при вычислении БПФ), требуется применение адресации с бит-реверсией адреса. Для поддержки требований этих алгоритмов в DAG реализовано свойство адресации с бит-реверсией, позволяющее разделять последовательности данных на составные части в циклической манере и сохранять эти данные в порядке с бит-реверсией адресов. Более подробную информацию об адресации с бит-реверсией см. описание команды Modify-Increment в *Руководстве по набору команд процессора Blackfin ADSP-BF53x*.

Индексная адресация с использованием индексных регистров и регистров-указателей

При индексной адресации в качестве эффективного адреса используется значение индексного регистра или регистра указателя. Эти команды могут выполнять загрузку в регистр или сохранение в память 16- или 32-разрядных величин. По умолчанию используются 32-разрядные передачи. При необходимости 16-разрядной передачи, в записи команды загрузки или сохранения используется обозначение *W*.

Например:

```
R0 = [I2];
```
Загружается 32-разрядное значение из адреса памяти, на который указывает `I2` и помещается в регистр `R0`.

```
R0.H = W[I2];
```
Загружается 16-разрядное значение из адреса памяти, на который указывает `I2`, и помещается в 16-разрядный регистр, `R0.H`.

```
[P1] = R0;
```
Приведенная выше команда является примером операции сохранения в память 32-разрядного значения.

Регистры указателей могут использоваться в операциях загрузки или сохранения 8-разрядных значений.

Например:

```
B[P1++] = R0;
```
Выполняется помещение 8-разрядного значения, содержащегося в регистре `R0`, в ячейку памяти по адресу, на который указывает регистр `P1`. После этого выполняется инкрементирование `P1`.

Адресация с автоматическим инкрементированием и автоматическим декрементированием

При адресации с автоматическим инкрементированием значения индексных регистров и регистров указателей обновляются после выполнения обращения к памяти. Значение инкремента зависит от длины слова. Доступы с 32-разрядными словами приводят к обновлению указателя на 4. При доступах с 16-разрядными словами указатель обновляется на 2, при доступах с 8-разрядными словами – на 1. В 8- и 16-разрядных операциях чтения может указываться дополнение значения, загружаемого в регистр, либо знаковыми битами, либо нулями. Регистры указателей могут использоваться в 8-, 16- или 32-разрядных доступах, индексные регистры могут использоваться только в 16- и 32-разрядных доступах.

Например:

```
R0 = W[P1 ++] (Z);
```

В 32-разрядный регистр загружается 16-разрядное слово, содержащееся по адресу, на который указывает регистр указателя `P1`. Затем указатель инкрементируется на 2, и слово дополняется знаковыми битами до 32 разрядов.

Операция автоматического декрементирования работает аналогично, за исключением того, что после обращения к памяти выполняется декрементирование адреса.

Например:

```
R0 = [I2--];
```

В регистр `R0` загружается 32-разрядное значение, индексный регистр декрементируется на 4.

Адресация с предмодификацией указателя стека.

В единственной поддерживаемой команде предмодификации используется регистр указателя стека, `SP`. Адрес, содержащийся в регистре `SP`, декрементируется на 4 и используется, затем в качестве эффективного адреса в операции сохранения в память. Команда `[--SP] = R0;` используется в операциях извлечения из стека и поддерживает только передачи 32-разрядных слов.

Индексная адресация с явно заданным смещением

Операция индексной адресации позволяет программам получать значения из таблиц данных по ссылке на базовый адрес таблицы. Регистр указателя модифицируется явно заданным полем, после чего используется в качестве эффективного адреса. Значение регистра указателя при этом не обновляется.

Когда полученное значение адреса не выровнено, в памяти вызывается исключение выравнивания.

Например, если `P1 = 0x13`, то адрес `[P1 + 0x11]` будет равен `[0x24]`. Это значение адреса корректно выровнено при любом типе доступа.

Адресация с постмодификацией

При адресации с постмодификацией в качестве эффективного адреса используется значение индексного регистра или регистра указателя, которое затем модифицируется значением другого регистра. Индексные регистры модифицируются регистрами модификации. При адресации с постмодификацией не поддерживается использование регистров указателей в качестве регистра-приемника, а также байтовая адресация.

Например:

```
R5 = [P1++P2];
```

В регистр `R5` загружается 32-разрядное значение, содержащееся в ячейке памяти, на которую указывает регистр `P1`. Затем к значению регистра `P1` добавляется значение регистра `P2`.

Например:

```
R2 = W[P4++P5] (Z);
```

В младшую половину регистра `R2` загружается 16-разрядное слово, дополняемое знаковыми битами до 32-х разрядов. Значение указателя `P4` инкрементируется на величину указателя `P5`.

Например:

```
R2 = [I2++M1];
```

В регистр `R2` загружается 32-разрядное слово. Значение индексного регистра `I2` обновляется значением регистра модификации `M1`.

Модификация регистров DAG и регистров указателей

Генераторы адреса данных поддерживают операции модификации значения адреса, содержащегося в индексном регистре, без выдачи адреса. Эта операция модификации адреса полезна при работе с указателями.

Операция модификации адреса модифицирует адреса, содержащиеся в любом индексном регистре DAG или регистре указателя (`I[3:0]`, `P[5:0]`, `FP`, `SP`), без обращения к памяти. Если соответствующие индексному регистру B- и L-

регистры настроены для адресации циклического буфера, операция модификации адреса выполняет (по необходимости) обращение в буфере.

Синтаксис этой операции схож с синтаксисом адресации с постмодификацией (индекс += модификатор). В качестве модификатора индексного регистра используется M-регистр, в качестве модификатора регистра указателя используется другой регистр указателя.

Рассмотрим следующий пример:

```
I1 += M2 ;
```

При выполнении этой команды к значению регистра `I1` прибавляется значение регистра `M2`, затем полученное значение записывается в регистр `I1`.

Выравнивание адресов памяти

Процессор требует осуществления соответствующего выравнивания памяти в зависимости от разрядности данных, обращение к которым производится. Нарушения выравнивания памяти вызывают исключения выравнивания, если они разрешены. Некоторые команды (например, многие команды видео АЛУ) автоматически запрещают исключения выравнивания, так как при их использовании возможно несоответствие выравнивания при помещении данных в память. Исключения выравнивания могут запрещаться вызовом команды `DISALGNEXPT` параллельно с операцией загрузки /сохранения.

В обычном режиме система памяти требует двух типов выравнивания адреса:
- Доступ в 32-разрядных командах загрузки/сохранения требует выравнивания по четырехбайтной границе, т.е. два младших бита адреса должны быть равны b#00.
- Доступ в 16-разрядных командах загрузки/сохранения требует выравнивания по двухбайтной границе, т.е. младший бит адреса должен быть равен b#0.

В таблице 5-1 приведен обзор типов и разрядностей передач, поддерживаемых в режимах адресации.

При использовании команды `DISALGNEXPT` следует соблюдать осторожность, так как она запрещает автоматическое определение ошибок выравнивания памяти. Команда `DISALGNEXPT` влияет только на ошибки выравнивания в операциях загрузки регистров при косвенной адресации с использованием I-регистров. Ошибки выравнивания в командах загрузки регистров при адресации с использованием P-регистров вызывают исключения.

Таблица 5-1. Типы и разрядности поддерживаемых передач.

Режим адресации	Типы поддерживаемых передач	Разрядности передач
С автоматическим инкрементированием С автоматическим декрементированием Косвенная Индексная	В регистры данных и из регистров данных	В командах загрузки: 32-разрядное слово 16-разрядное полуслово, дополняемое нулями 16-разрядное полуслово, дополняемое знаковыми битами 8-разрядный байт, дополняемый знаковыми битами 8-разрядный байт, дополняемый нулями В командах сохранения в память: 32-разрядное слово 16-разрядное полуслово 8-разрядный байт
	В регистры и из регистров указателей	В командах загрузки: 32-разрядное слово В командах сохранения в память: 32-разрядное слово
С постинкрементированием	В регистры данных и из регистров данных	В командах загрузки: 32-разрядное слово 16-разрядное полуслово, загружаемое в старшую половину регистра данных 16-разрядное полуслово, загружаемое в младшую половину регистра данных 16-разрядное полуслово, дополняемое нулями 16-разрядное полуслово, дополняемое знаковыми битами В командах сохранения в память: 32-разрядное слово 16-разрядное полуслово, извлекаемое из старшей половины регистра данных 16—разрядное полуслово, извлекаемое из младшей половины регистра данных

Таблица 5-2 представляет собой сводную таблицу режимов адресации. Символ (*) обозначает, что соответствующий режим адресации поддерживается процессором.

Таблица 5-2.

	32-разрядное слово	16-разрядное полуслово	8-разрядный байт	Дополнение знаковыми битами/нулями	Регистр данных	Регистр указателя	Половина регистра данных
Автоматическое инкрементирование Р-регистра [P0++]	*	*	*	*	*	*	
Автоматическое декрементирование Р-регистра [P0--]	*	*	*	*	*	*	
Косвенная адресация с использованием Р-регистра [P0]	*	*	*	*	*	*	*
Индексная адресация с использованием Р-регистра [P0+im]	*	*	*	*	*	*	
Индексная адресация с	*				*	*	

использованием регистра FP [FP+im]							
Пост-инкрементирование P-регистра [P0++P1]	*	*		*	*		*
Автоматическое инкрементирование I-регистра [I0++]	*	*			*		*
Автоматическое декрементирование I-регистра [I0--]	*	*			*		*
Косвенная адресация с использованием I-регистра [I0]	*	*			*		*
Пост-инкрементирование I-регистра [I0++M0]	*				*		

Обзор команд DAG

В таблице 5-3 перечислены команды DAG. Дополнительную информацию о синтаксисе языка ассемблера см. в *Руководстве по набору команд процессора Blackfin ADSP-BF53x*. В таблице 5-3 используются следующие обозначения:

- Dreg – любой регистр регистрового файла данных.
- Dreg_lo – младшие 16 разрядов любого регистра регистрового файла данных.
- Dreg_hi – старшие 16 разрядов любого регистра регистрового файла данных.
- Preg – любой регистр указателя, `FP` или `SP`.
- Ireg – любой индексный регистр DAG.
- Mreg – любой регистр модификации DAG.
- W – 16-разрядное значение.
- B – 8-разрядное значение.
- immA – знаковое A-разрядное непосредственно заданное значение.
- uimmAmB – беззнаковое A-разрядное непосредственно заданное значение, делящееся на B без остатка.
- Z – спецификатор дополнения нулями.
- X – спецификатор дополнения знаковыми разрядами.
- B – спецификатор бит-реверсии.

Опции, применимые совместно с этими командами, и размеры непосредственно задаваемых полей более подробно описываются в *Руководстве по набору команд процессора Blackfin ADSP-BF53x*.

Таблица 5-3. Обзор команд DAG

Команда
Preg = [Preg] ;
Preg = [Preg ++] ;
Preg = [Preg --] ;
Preg = [Preg + uimm6m4] ;
Preg = [Preg + uimm17m4] ;
Preg = [Preg – uimm17m4] ;
Preg = [FP – uimm7m4] ;
Dreg = [Preg] ;
Dreg = [Preg ++] ;
Dreg = [Preg --] ;
Dreg = [Preg + uimm6m4] ;
Dreg = [Preg + uimm17m4] ;
Dreg = [Preg – uimm17m4] ;
Dreg = [Preg ++ Preg] ;
Dreg = [FP – uimm7m4] ;
Dreg = [Ireg];
Dreg = [Ireg ++];
Dreg = [Ireg --];
Dreg = [Ireg ++ Mreg];
Dreg = W [Preg] (Z) ;
Dreg = W [Preg ++] (Z) ;
Dreg = W [Preg --] (Z) ;
Dreg = W [Preg + uimm5m2] (Z) ;
Dreg = W [Preg + uimm16m2] (Z) ;
Dreg = W [Preg – uimm16m2] (Z) ;
Dreg = W [Preg ++ Preg] (Z) ;
Dreg = W [Preg] (X) ;
Dreg = W [Preg ++] (X) ;
Dreg = W [Preg --] (X) ;
Dreg = W [Preg + uimm5m2] (X) ;
Dreg = W [Preg + uimm16m2] (X) ;
Dreg = W [Preg – uimm16m2] (X) ;
Dreg = W [Preg ++ Preg] (X) ;
Dreg_hi = W [Ireg] ;
Dreg_hi = W [Ireg ++] ;
Dreg_hi = W [Ireg --] ;
Dreg_hi = W [Preg] ;
Dreg_hi = W [Preg ++ Preg] ;
Dreg_lo = W [Ireg] ;
Dreg_lo = W [Ireg ++] ;
Dreg_lo = W [Ireg --] ;
Dreg_lo = W [Preg] ;
Dreg_lo = W [Preg ++ Preg] ;
Dreg =B [Preg] (Z) ;
Dreg =B [Preg ++] (Z) ;
Dreg =B [Preg --] (Z) ;
Dreg =B [Preg + uimm15] (Z) ;
Dreg =B [Preg – uimm15] (Z) ;
Dreg = B [Preg] (X) ;
Dreg = B [Preg ++] (X) ;
Dreg = B [Preg --] (X) ;
Dreg = B [Preg + uimm15] (X) ;
Dreg = B [Preg – uimm15] (X) ;
[Preg] = Preg ;
[Preg ++] = Preg ;
[Preg --] = Preg ;

Генераторы адреса данных

[Preg + uimm6m4] = Preg ;
[Preg + uimm17m4] = Preg ;
[Preg – uimm17m4] = Preg ;
[FP – uimm7m4] = Preg ;
[Preg] = Dreg ;
[Preg ++] = Dreg ;
[Preg --] = Dreg ;
[Preg + uimm6m4] = Dreg ;
[Preg + uimm17m4] = Dreg ;
[Preg – uimm17m4] = Dreg ;
[Preg ++ Preg] = Dreg ;
[FP – uimm7m4] = Dreg ;
[Ireg] = Dreg ;
[Ireg ++] = Dreg ;
[Ireg --] = Dreg ;
[Ireg ++ Mreg] = Dreg ;
W [Ireg] = Dreg_hi ;
W [Ireg ++] = Dreg_hi ;
W [Ireg --] = Dreg_hi ;
W [Preg] = Dreg_hi ;
W [Preg ++ Preg] = Dreg_hi ;
W [Ireg] = Dreg_lo ;
W [Ireg ++] = Dreg_lo ;
W [Ireg --] = Dreg_lo ;
W [Preg] = Dreg_lo ;
W [Preg] = Dreg ;
W [Preg ++] = Dreg ;
W [Preg --] = Dreg ;
W [Preg + uimm5m2] = Dreg ;
W [Preg + uimm16m2] = Dreg ;
W [Preg -- uimm16m2] = Dreg ;
W [Preg + + Preg] = Dreg_lo ;
B [Preg] = Dreg ;
B [Preg ++] = Dreg ;
B [Preg --] = Dreg ;
B [Preg + uimm15] = Dreg ;
B [Preg -- uimm15] = Dreg ;
Preg = imm7 (X) ;
Preg = imm16 (X) ;
Preg += Preg (BREV) ;
Ireg += Mreg (BREV) ;
Preg = Preg << 2 ;
Preg = Preg >> 2 ;
Preg = Preg >> 1 ;
Preg = Preg + Preg << 1 ;
Preg = Preg + Preg << 2 ;
Preg -= Preg ;
Ireg -= Mreg ;