2 ВЫЧИСЛИТЕЛЬНЫЕ УСТРОЙСТВА

Вычислительные устройства процессора выполняют обработку данных в алгоритмах ЦОС и общего управления. Процессор содержит шесть вычислительных устройств: включают два арифметико-логических устройства (АЛУ), два умножителя/накопителя (умножителя), устройство сдвига и набор видео АЛУ. Эти устройства получают данные из регистров регистрового файла данных. Вычислительные команды этих устройств обеспечивают выполнение операций с фиксированной точкой, каждая вычислительная команда может быть выполнена за один такт работы процессора.

Вычислительные устройства выполняют различные типы операций. АЛУ выполняют арифметические и логические операции. Умножители выполняют умножение и операции умножения/сложения и умножения/вычитания. Устройство сдвига выполняет операции логического и арифметического сдвига, упаковки и извлечения битов. Видео АЛУ выполняют логические SIMD (Single Instruction Multiple Data) операции над определёнными 8-разрядными операндами.

Вычислительные устройства принимают и передают данные через регистровый файл данных, состоящий из восьми 32-разрядных регистров. В операциях, требующих использования 16-разрядных операндов, регистры разбиваются на пары, обеспечивая возможность использования шестнадцати 16-разрядных регистров.

Доступ к регистровому файлу данных обеспечивается командами языка ассемблера процессора. Синтаксис ассемблера позволяет осуществлять перемещение данных с помощью этих регистров и одновременно определять формат данных, используемых при вычислениях.

Структура изложения материалов в других разделах этой главы следует из рис. 2-1. Сначала приводится описание каждого вычислительного устройства, которое содержит подробную информацию о принципах его работы и сопровождается обзором набора вычислительных команд. Детальное изучение организации вычислительных устройств, регистровых файлов и шин данных позволяет лучше понять процесс обработки данных при вычислениях. В заключение рассматриваются расширенные возможности параллельного выполнения команд процессором и раскрываются преимущества использования многофункциональных команд.

На рис. 2-1 показана взаимосвязь между регистровым файлом данных и вычислительными устройствами – умножителями, АЛУ и устройством сдвига.

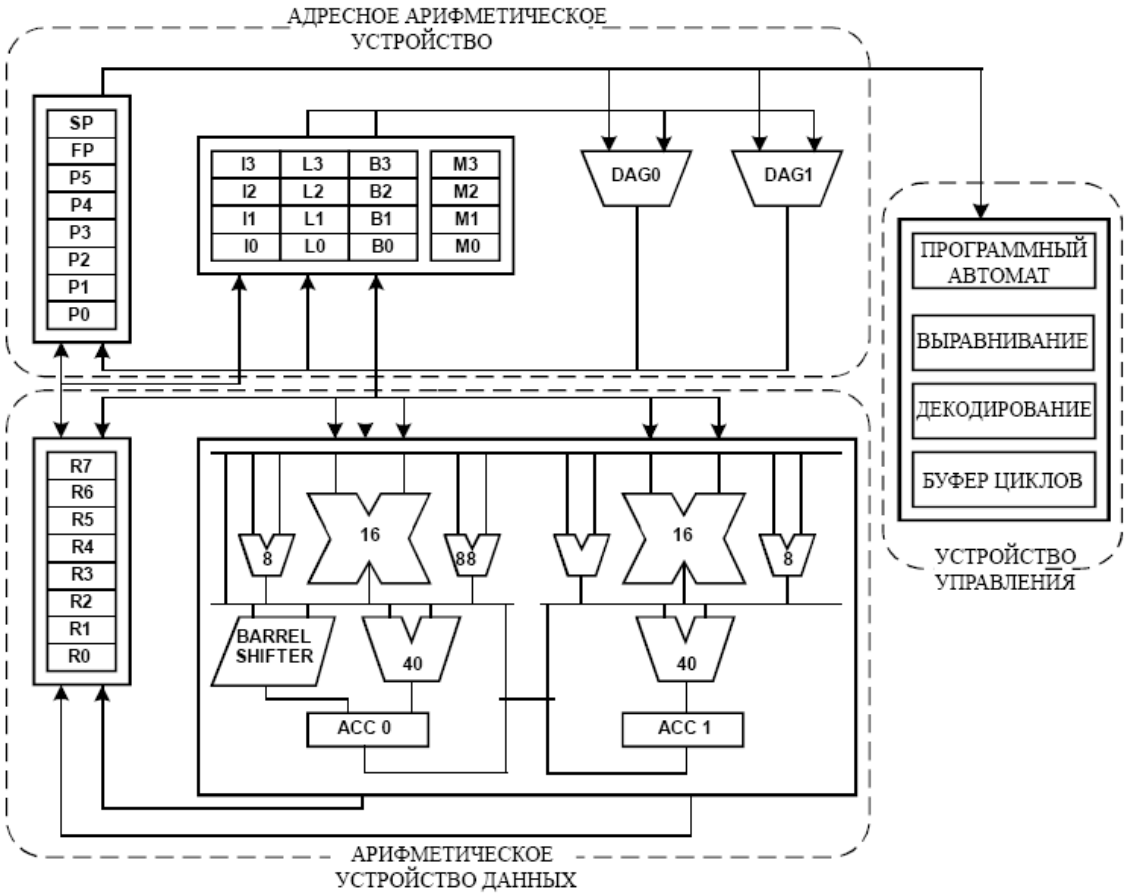

Рис. 2-1. Архитектура ядра процессора

Простые (однофункциональные) команды умножителя, АЛУ и устройства сдвига имеют неограниченный доступ к регистрам регистрового файла данных. Многофункциональные операции могут иметь ограничения, описываемые в разделе, посвящённом данному режиму вычислений.

Два дополнительных регистра, `A0` и `A1`, содержат 40-разрядные результаты аккумулятора. Эти регистры являются выделенными регистрами АЛУ и предназначены в основном для реализации умножения-накопления.

Традиционные режимы арифметических операций, такие как целочисленный и дробный, явно указываются в команде. Режимы округления устанавливаются в регистре `ASTAT`, в который также записываются флаги состояния/условия результатов вычислительных операций.

Использование форматов данных

Процессоры ADSP-BF53x являются 16-разрядными процессорами, выполняющими операции над данными с фиксированной точкой. В большинстве операций используется представление чисел в дополнительном коде, а также беззнаковые числа и простые двоичные строки. Кроме того, существуют команды,

поддерживающие выполнение арифметических операций с 32-разрядными целыми числами, а также специальные возможности работы с 8-разрядными числами и числами в формате с блочной плавающей точкой. Подробная информация об используемых форматах приведена в приложении D “Форматы представления чисел”.

В арифметических операциях процессоров семейства ADSP-BF53x знаковые числа всегда представлены в дополнительном коде. В этих процессорах не используются форматы величины со знаком, поразрядного дополнения, двоично-десятичные числа или избыточный код.

Двоичная строка

Формат двоичной строки является самой простой формой двоичной записи; в данном формате шестнадцать бит обрабатываются как битовая комбинация. Примерами вычислений с использованием этого формата являются логические операции: NOT, AND, OR, XOR. В этих операциях АЛУ операнды обрабатываются как двоичные строки без учёта информации о знаковом бите или размещении двоичной точки.

Беззнаковые числа

Беззнаковые двоичные числа могут рассматриваться как положительные числа, имеющие значение приблизительно равное удвоенному значению знакового числа той же длины. Процессор обрабатывает младшие слова (least significant words) чисел с повышенной точностью как беззнаковые числа.

Знаковые числа: дополнительный код

В арифметике процессора ADSP-BF53x термин *знаковый* относится к числам, представленным в дополнительном коде. Большинство операций процессоров семейства Blackfin предполагает или поддерживает арифметику в формате двоичного дополнения.

Представление дробных чисел в формате 1.15

Арифметика процессора ADSP-BF53x оптимизирована для работы с числами в дробном двоичном формате, обозначаемом 1.15 (“один точка пятнадцать”). В формате 1.15 один знаковый бит (старший бит, MSB) и пятнадцать битов дробной части используются для представления диапазона значений от -1 до 0.999969.

На рис. 2-2 показаны веса битов для чисел в формате 1.15, а также несколько примеров записи чисел в формате 1.15 и их десятичные эквиваленты.

ЧИСЛО В ФОРМАТЕ 1.15 (ШЕСТНАДЦАТЕРИЧНЫЙ ФОРМАТ)	ДЕСЯТИЧНЫЙ ЭКВИВАЛЕНТ
0X0001	0.000031
0X7FFF	0.999969
0XFFFF	-0.000031
0X8000	-1.000000

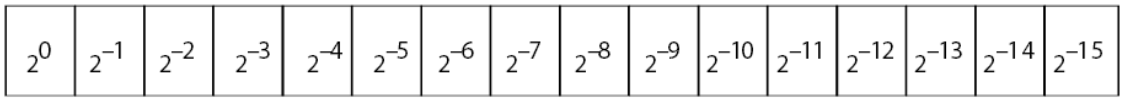

Рис. 2-2. Веса битов для чисел в формате 1.15.

Регистровые Файлы

Вычислительные устройства процессора имеют три категории групп регистров – регистровый файл данных, регистровый файл указателей и набор регистров генератора адреса данных (DAG):

- регистровый файл данных принимает операнды для вычислительных устройств по шине данных, и в него помещаются результаты вычислений;
- регистровый файл указателей содержит указатели, используемые в операциях адресации;
- регистры DAG представляют собой набор выделенных регистров, предназначенных для организации циклических буферов с нулевыми непроизводительными затратами в операциях ЦОС.

Дополнительную информацию см. в главе 5 “Генераторы Адреса Данных”.

Регистровые файлы процессора показаны на рис. 2-3.

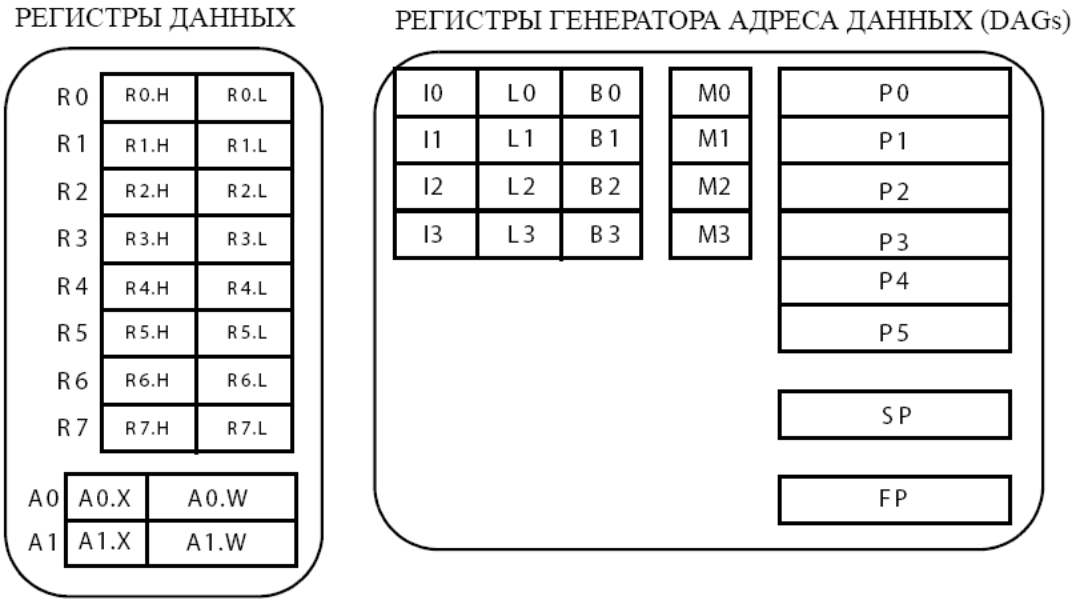

Рис. 2-3. Регистровые файлы ADSP-21535.

В процессоре слово (word) имеет длину 32 разряда; H обозначает 16 старших разрядов 32-разрядного регистра; L обозначает 16 младших разрядов 32-разрядного регистра. Например, `A0.W` содержит младшие 32 разряда 40-разрядного регистра `A0`; `A0.L` содержит младшие 16 разрядов `A0.W`, `A0.H` содержит старшие 16 разрядов `A0.W`.

Регистровый файл данных

Регистровый файл данных состоит из восьми 32-разрядных регистров. Каждый регистр может быть представлен как пара независимых 16-разрядных регистров – старшей и младшей половин регистра. Таким образом, 32-разрядный регистр `R0` может рассматриваться как две независимые половины регистра, `R0.L` и `R0.H`.

Регистровый файл соединён с памятью данных L1 тремя отдельными шинами (две используются для чтения, одна – для записи), разрядностью по 32 разряда каждая. На каждом такте работы процессора между регистровым файлом данных и памятью данных может передаваться до четырёх 16-разрядных слов достоверных данных.

Регистры аккумуляторов

В дополнение к регистровому файлу данных в процессоре имеется два выделенных 40-разрядных регистра аккумулятора. К каждому из них можно обращаться как к его 16-разрядной младшей (`An.L`) или старшей (`An.H`) половине с 8-разрядным расширением (`An.X`). Кроме того, к каждому из этих регистров можно обращаться как к 32-разрядному регистру (`An.W`), содержащему младшие 32 разряда, или как к полному 40-разрядному регистру результата (`An`).

Регистровый Файл Указателей

Регистры указателей адреса общего назначения, также называемые P-регистрами, сгруппированы следующим образом:

- 6-элементный файл P-регистров `P[5:0]`;
- указатель кадра (`FP`), используемый в качестве указателя на запись активации текущей процедуры;
- регистр указателя стека (`SP`), являющийся указателем на последнюю задействованную позицию в стеке исполняемой программы. См. описание регистров, зависящих от режима, в главе 3 “Рабочие режимы и состояния”.

P-регистры имеют разрядность 32 разряда. Хотя P-регистры используются в первую очередь для вычислений адреса, они также могут использоваться в ограниченном наборе арифметических операций над целыми числами, например, для организации счётчиков. Однако, в отличие от регистров данных, арифметические операции с использованием P-регистров не влияют на флаги состояния в регистре арифметического состояния (`ASTAT`).

Набор регистров генератора адреса данных

При адресации команды ЦСП в первую очередь используют набор регистров генератора адреса данных (DAG). Набор регистров DAG состоит из следующих регистров:

- `I[3:0]` содержат адреса и выполняют функцию указателей;
- `M[3:0]` содержат значения модификации;
- `B[3:0]` содержат базовые адреса;
- `L[3:0]` содержат значения длины.

Разрядность всех регистров DAG – 32 разряда. Регистры I (индексные) и B (базового адреса) всегда содержат адреса 8-разрядных байтов в памяти. Индексный регистр содержит эффективный адрес. Регистры M (модификации) содержат величину смещения, которая прибавляется к содержимому одного из индексных регистров или вычитается из него.

Регистры B (базового адреса) и L (длины) задают параметры циклических буферов. B-регистр содержит начальный адрес буфера, L-регистр содержит его длину в байтах. Каждая пара L- и B-регистров используется совместно с соответствующим I-регистром. Например, `L0` и `B0` всегда соответствуют регистру `I0`. Однако любой M-регистр может использоваться совместно с любым I-регистром. Например, регистр `I0` может модифицироваться регистром `M3`. Дополнительную информацию см. в главе 5 “Генераторы Адреса Данных”.

Обзор команд Регистрового Файла

В таблице 2-1 приведены команды регистрового файла. Дополнительную информацию о синтаксисе языка ассемблера см. в *Справочном руководстве по набору команд процессора Blackfin ADSP-BF53x*.
В таблице 2-1 используются следующие обозначения:

- Allreg – `R[7:0]`, `P[5:0]`, `SP`, `FP`, `I[3:0]`, `M[3:0]`, `B[3:0]`, `L[3:0]`, `A0.X`, `A0.W`, `A1.X`, `A1.W`, `ASTAT`, `RETS`, `RETI`, `RETX`, `RETN`, `RETE`, `LC[1:0]`, `LT[1:0]`, `LB[1:0]`, `USP`, `SEQSTAT`, `SYSCFG`, `EMUDAT`, `CYCLES` и `CYCLES2`.
- A*n* – регистр результата любого АЛУ, `A0` или `A1`.
- Dreg – любой регистр Регистрового Файла Данных.
- Sysreg – регистры системы: `ASTAT`, `SEQSTAT`, `SYSCFG`, `RETI`, `RETX`, `RETN`, `RETE` или `RETS`, `LC[1:0]`, `LT[1:0]`, `LB[1:0]`, `EMUDAT`, `CYCLES` и `CYCLES2`.
- Preg – любой регистр Указателя, регистр `FP` или `SP`.
- Dreg_even – `R0`, `R2`, `R4` или `R6`.
- Dreg_odd – `R1`, `R3`, `R5` или `R7`.
- DPreg – любой регистр Регистрового Файла Данных или любой регистр Указателя, регистр `FP` или `SP`.
- Dreg_lo – младшие 16 разрядов любого регистра Регистрового Файла Данных.
- Dreg_hi – старшие 16 разрядов любого регистра Регистрового Файла Данных.

- A*n*.L – младшие 16 разрядов аккумулятора `A0.W` или `A1.W`.
- A*n*.H – старшие 16 разрядов аккумулятора `A0.W` или `A1.W`.
- Dreg_byte – младшие 8 разрядов каждого регистра Данных.
- Опция (X) – дополнение знаковыми разрядами.
- Опция (Z) – дополнение нулями.
- Символ “*” – флаг может быть установлен или сброшен в зависимости от результата выполнения команды
- Символ “**” – флаг сбрасывается.
- Символ “–” – операция не влияет на флаги.

Таблица 2-1. Обзор команд Регистрового Файла.

Команда	**Флаги статуса регистра ASTAT**						
	AZ	**AN**	**AC0 AC0_COPY AC1**	**AV0 AVS**	**AV1 AV1S**	**CC**	**V V_COPY VS**
Allreg = Allreg ;[1]	–	–	–	–	–	–	–
A*n* = A*n* ;	–	–	–	–	–	–	–
A*n* = Dreg ;	–	–	–	–	–	–	–
Dreg_even = A0 ;	*	*	–	–	–	–	*
Dreg_odd = A1 ;	*	*	–	–	–	–	*
Dreg_even = A0, Dreg_odd =A1 ;	*	*	–	–	–	–	*
Dreg_odd = A1, Dreg_even = A0 ;	*	*	–	–	–	–	*
IF CC DPreg = DPreg ;	–	–	–	–	–	–	–
IF ! CC DPreg = DPreg ;	–	–	–	–	–	–	–
Dreg = Dreg_lo (Z) ;	*	**	**	–	–	–	**/–
Dreg = Dreg_lo (X) ;	*	*	**	–	–	–	**/–
A*n*.X = Dreg_lo ;	–	–	–	–	–	–	–
A*n*.L = Dreg_lo ;	–	–	–	–	–	–	–
A*n*.H = Dreg_hi ;	–	–	–	–	–	–	–
Dreg_lo = A0 ;	*	*	–	–	–	–	*
Dreg_hi = A1 ;	*	*	–	–	–	–	*
Dreg_hi = A1 ; Dreg_lo = A0 ;	*	*	–	–	–	–	*
Dreg_lo = A0 ; Dreg_hi = A1 ;	*	*	–	–	–	–	*
Dreg = Dreg_byte (Z) ;	*	**	**	–	–	–	**/–
Dreg = Dreg_byte (X) ;	*	*	**	–	–	–	**/–

[1] Предупреждение: Не все комбинации регистров допустимы. Подробную информацию см. в функциональном описании команды Move Register в *Справочном руководстве по набору команд процессора Blackfin ADSP-BF53x*.

Типы данных

Процессор поддерживает 32-разрядные слова, 16-разрядные полуслова и байты. 32- и 16-разрядные слова могут представлять целые или дробные числа, байты всегда представляют целые числа. Целые числа могут быть знаковыми или беззнаковыми, дробные числа всегда являются знаковыми.
В таблице 2-2 приведены форматы данных, находящихся в памяти, регистровом файле и аккумуляторах. Буква "*d*" в таблице соответствует одному биту, буква "*s*" – одному знаковому биту.

Некоторые команды манипулируют данными в регистрах, дополняя их знаковыми разрядами или нулями до 32 разрядов, следующим образом:
- команды дополняют нулями беззнаковые данные;
- команды дополняют знаковыми разрядами знаковые 16-разрядные полуслова и 8-разрядные байты.

Остальные команды обращаются с данными как с 32-разрядными числами. Кроме того, двумя 16-разрядными полусловами или четырьмя 8-разрядными байтами можно манипулировать как 32-разрядными величинами. Более подробную информацию см. в описании команд в *Справочном руководстве по набору команд процессора Blackfin ADSP-BF53x*.

В таблице 2-2 используются следующие обозначения:
- s = знаковый бит (биты)
- d = бит (биты) данных
- "." = условная позиция десятичной запятой; в явном виде десятичная запятая в записи числа не присутствует.
- наклонным шрифтом выделяются биты, источник которых отличается от источника смежных с ними битов.

Таблица 2-2. Форматы данных

Формат	**Представление в памяти**	**Представление в 32-разрядном регистре**
Беззнаковое слово в формате 32.0	dddd dddd dddd dddd dddd dddd dddd dddd	dddd dddd dddd dddd dddd dddd dddd dddd
Знаковое слово в формате 32.0	sddd dddd dddd dddd dddd dddd dddd dddd	sddd dddd dddd dddd dddd dddd dddd dddd
Беззнаковое полуслово в формате 16.0	dddd dddd dddd dddd	0000 0000 0000 0000 dddd dddd dddd dddd
Знаковое полуслово в формате 16.0	sddd dddd dddd dddd	ssss ssss ssss ssss sddd dddd dddd dddd
Беззнаковый байт в формате 8.0	dddd dddd	0000 0000 0000 0000 0000 0000 dddd dddd
Знаковый байт в формате 8.0	sddd dddd	ssss ssss ssss ssss ssss ssss sddd dddd

Беззнаковое дробное число в формате 0.16	.dddd dddd dddd dddd	0000 0000 0000 0000 .dddd dddd dddd dddd
Знаковое дробное число в формате 1.15	s.ddd dddd dddd dddd	ssss ssss ssss ssss s.ddd dddd dddd dddd
Беззнаковое дробное число в формате 0.32	.dddd dddd dddd dddd dddd dddd dddd dddd	.dddd dddd dddd dddd dddd dddd dddd dddd
Знаковое дробное число в формате 1.31	s.ddd dddd dddd dddd dddd dddd dddd dddd	s.ddd dddd dddd dddd dddd dddd dddd dddd
Упакованный беззнаковый байт в формате 8.0	dddd dddd *dddd dddd* dddd dddd *dddd dddd*	dddd dddd *dddd dddd* dddd dddd *dddd dddd*
Упакованное беззнаковое дробное число в формате 0.16	.dddd dddd dddd dddd *.dddd dddd dddd dddd*	.dddd dddd dddd dddd *.dddd dddd dddd dddd*
Упакованное знаковое дробное число в формате 1.15	s.ddd dddd dddd dddd *s.ddd dddd dddd dddd*	s.ddd dddd dddd dddd *s.ddd dddd dddd dddd*

Порядок следования байтов

Доступ к внутренней и внешней памяти производится в формате с порядком следования байтов, начиная с младшего (little-endian byte order). Дополнительную информацию см. в разделе “Модель транзакций памяти” в главе 6.

Типы данных АЛУ

Операнды и результаты операций каждого АЛУ, за исключением примитива знакового деления (`DIVS`), обрабатываются либо как 16- либо как 32-разрядные двоичные строки. При формировании битов состояния результата АЛУ результаты обрабатываются как знаковые числа; состояние указывается флагами переполнения (`AV0`, `AV1`) и флагом отрицательного результата (`AN`). Каждое АЛУ имеет собственный “залипающий” флаг состояния переполнения, `AV0S` и `AV1S`. Если эти биты устанавливаются, их значение остаётся неизменным до осуществления сброса непосредственной записью в регистр `ASTAT`. Дополнительный флаг `V` устанавливается или сбрасывается в зависимости от передачи результата из обоих аккумуляторов в регистровый файл. Кроме того, одновременно с битом `V` устанавливается “залипающий” бит `VS`, который остаётся в установленном состоянии до принудительного сброса.

Логика формирования битов переполнения (`V`, `VS`, `AV0`, `AV0S`, `AV1`, `AV1S`) основана на правилах арифметики чисел в дополнительном коде. Бит или набор битов устанавливается в случае, когда получаемое значение старшего разряда отличается от значения, ожидаемого на основании знаков операндов и типа операции. Например, при сложении двух положительных чисел должен формироваться положительный результат; изменение в знаковом бите означает переполнение и приводит к установке соответствующих флагов переполнения,

AVn. Сложение отрицательного и положительного числа может дать как положительный, так и отрицательный результат, но не может привести к переполнению.

Логика формирования битов переноса (AC0, AC1) основана на правилах арифметики беззнаковых чисел. Бит устанавливается, когда происходит перенос из 16-го разряда (MSB). Биты переноса (AC0, AC1) наиболее полезны при их формировании в результате операций над младшими частями составных слов.

На основании результатов АЛУ генерируется информация о состоянии. Дополнительную информацию об использовании состояния АЛУ см. в разделе “Обзор команд АЛУ”.

Типы данных умножителя

Результатами операций каждого умножителя являются двоичные строки. Тип входных данных интерпретируется в соответствии с информацией, задаваемой в теле самой команды (умножение знакового числа на знаковое, беззнакового на беззнаковое, знакового на беззнаковое, или округление). Предполагается, что 32-разрядный результат операций умножителей является знаковым; он дополняется знаковыми разрядами до 40-разрядного значения (полной разрядности регистров A0 или A1).

Процессор поддерживает два режима преобразования формата: режим умножения дробных чисел для дробных операндов (формат 1.15 с 1 знаковым битом и 15 битами дробной части) и режим умножения целых чисел для целочисленных операндов (формат 16.0).

Когда процессор выполняет умножение двух операндов в формате 1.15, результатом является число в формате 2.30 (2 знаковых бита и 30 битов дробной части). В режиме умножения дробных чисел умножитель автоматически сдвигает произведение на один разряд влево перед передачей результата в регистр результата умножителя (A0, A1). После сдвига избыточного знакового бита результат умножителя будет представлен в формате 1.31, и может быть округлён до формата 1.15. Окончательный формат результата показан на рис. 2-4.

В режиме умножения целых чисел сдвиг влево не производится. Например, если операнды представлены в формате 16.0, 32-разрядный результат умножителя будет представлен в формате 32.0. Сдвиг влево производить не нужно, так как он может исказить результат. Формат результата для данного режима показан на рис. 2-5.

При обновлении значения аккумулятора результатом умножителя или при передаче результата в регистр регистрового файла генерируется информация о состоянии. Дополнительную информацию см. в разделе “Обзор команд умножителя”.

Типы данных устройства сдвига

Многие операции устройства сдвига в явном виде требуют использования знаковых (в дополнительном коде) или беззнаковых величин – в операциях логического сдвига предполагается использование беззнаковых чисел или двоичных строк, в операциях арифметического сдвига предполагается использование чисел в дополнительном коде.

Логика определения порядка работает с числами в дополнительном коде, а также поддерживает числа в формате с блочной плавающей точкой, который также основан на представлении дробных чисел в дополнительном коде.

На основании результатов устройства сдвига генерируется информация о состоянии. Дополнительную информацию об использовании состояния устройства сдвига см. в разделе “Обзор команд устройства сдвига”.

Обзор арифметических форматов

В таблицах 2-3, 2-4, 2-5 и 2-6 приведены некоторые характеристики арифметических операций.

Таблица 2-3. Форматы данных АЛУ при выполнении арифметических операций

Операция	Формат операндов	Формат результатов
Сложение	Знаковые или беззнаковые	Интерпретируется флагами
Вычитание	Знаковые или беззнаковые	Интерпретируется флагами
Логические операции	Двоичные строки	Такой же, как и формат входных операндов
Деление	Явно заданные знаковые или беззнаковые	Такой же, как и формат входных операндов

Таблица 2-4. Форматы данных умножителя при выполнении арифметических операций в режиме умножения дробных чисел

Операция	Формат операндов	Формат результатов
Умножение	Явно заданные знаковые или беззнаковые в формате 1.15	2.30 преобразуемый сдвигом в 1.31
Умножение/сложение	Явно заданные знаковые или беззнаковые в формате 1.15	2.30 преобразуемый сдвигом в 1.31
Умножение/вычитание	Явно заданные знаковые или беззнаковые в формате 1.15	2.30 преобразуемый сдвигом в 1.31

Таблица 2-5. Форматы данных умножителя при выполнении арифметических операций в режиме умножения целых чисел

Операция	Формат операндов	Формат результатов
Умножение	Явно заданные знаковые или беззнаковые в формате 16.0	32.0 без сдвига
Умножение/сложение	Явно заданные знаковые или беззнаковые в формате 16.0	32.0 без сдвига
Умножение/вычитание	Явно заданные знаковые или беззнаковые в формате 16.0	32.0 без сдвига

Таблица 2-6. Форматы данных устройства сдвига при выполнении арифметических операций

Операция	Формат операндов	Формат результатов
Логический сдвиг	Беззнаковые двоичные строки	Такой же, как и формат входных операндов
Арифметический сдвиг	Знаковые	Такой же, как и формат входных операндов
Определение порядка	Знаковые	Такой же, как и формат входных операндов

Использование целочисленного и дробного форматов умножителя

При организации в процессоре функций умножения-накопления возможны два режима – выполнение арифметических операций с дробными числами в формате (1.15) и арифметических операций с целыми числами в формате (16.0).

В режиме умножения дробных чисел перед сложением с содержимым аккумулятора `A0` или `A1` производится преобразование формата – 32-разрядное произведение дополняется знаковыми разрядами и сдвигается на один разряд влево. Например, 31-ый бит произведения выравнивается с 32-ым битом регистра `A0` (бит 0 регистра `A0.X`), бит 0-го произведения выравнивается с битом 1 регистра `A0` (бит 1 регистра `A0.W`). В младший бит записывается 0. Формат результата умножителя в режиме умножения дробных чисел показан на рис. 2-4.

В режиме умножения целых чисел сдвиг 32-разрядного результата перед сложением с `A0` или `A1` не производится. На рис. 2-5 показан формат результата умножителя в режиме умножения целых чисел.

При работе в обоих режимах результат умножителя подаётся на 40-разрядное устройство сложения/вычитания, в котором он складывается с текущим содержимым регистра `A0` или `A1` или вычитается из него, формируя окончательный 40-разрядный результат.

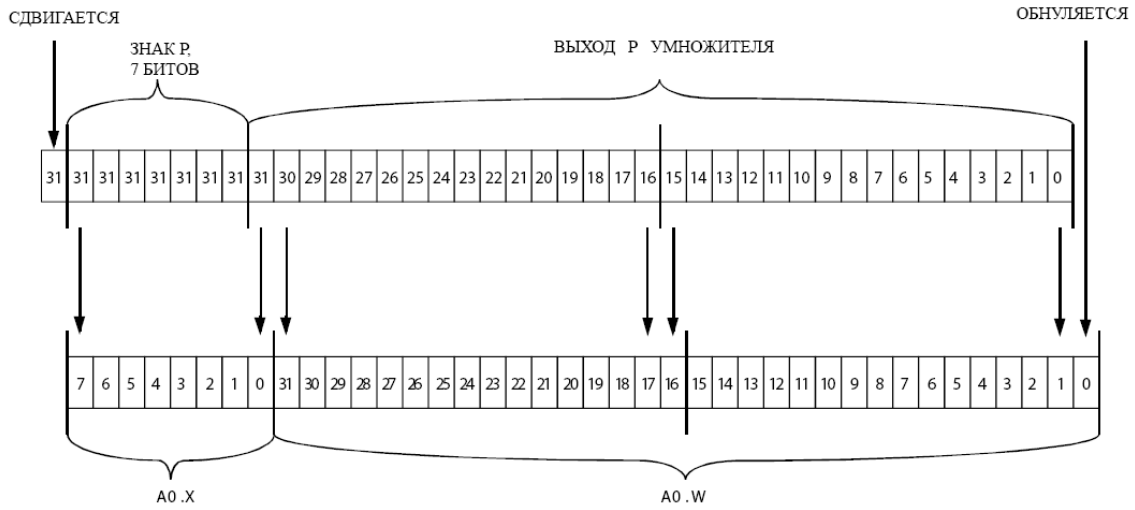

Рис. 2-4. Формат результата умножителя в режиме умножения дробных чисел

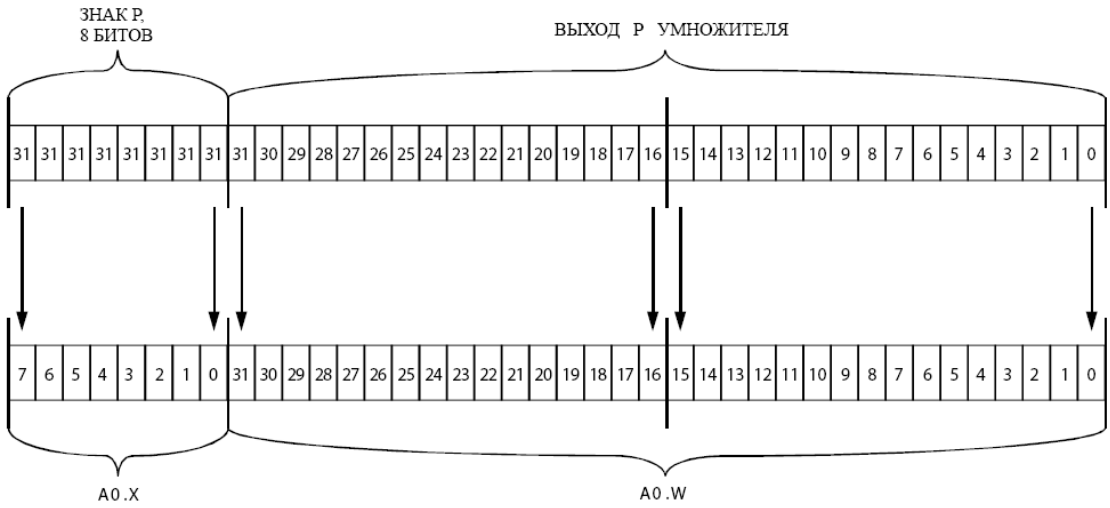

Рис. 2-5. Формат результата умножителя в режиме умножения целых чисел

Округление результатов умножителя

При выполнении многих операций умножителя процессор поддерживает округление результата (опция RND). Округление является средством понижения точности представления путём отбрасывания диапазона младших разрядов в записи числа и возможной модификации оставшейся части с целью более точного представления исходного значения. Например, исходное число имеет точность представления N бит, а новое число будет иметь точность представления только M бит (N>M). В процессе округления из записи числа удаляются N-M бит.

Тип округления при использовании опции `RND` определяется битом `RND_MOD` в регистре `ASTAT`. Может выполняться смещённое или несмещённое округление. Для использования *несмещённого* округления необходимо записать в бит `RND_MOD` значение 0. Для использования *смещённого* округления необходимо записать в бит `RND_MOD` значение 1.

В большинстве алгоритмов предпочтительнее использовать несмещённое округление.

Несмещённое округление

Метод сходящегося округления возвращает число, ближайшее к исходному. В случаях, когда исходное число лежит точно на границе между двумя числами, этот метод возвращает ближайшее чётное число, то есть число, содержащее 0 в младшем разряде. Например, при округлении 3-разрядного дробного числа 0.25 в дополнительном коде (0.01 в двоичной записи) до ближайшего 2-разрядного дробного числа в дополнительном коде результат составит 0.0, так как данное число является чётным при выборе из значений 0.5 и 0.0. Этот метод называется *несмещённым* округлением, так как округление и в большую, и в меньшую сторону производится на основании соседних значений.

При организации несмещённого округления используется способность АЛУ округлять 40-разрядный результат на границе между 15-ым и 16-ым битами. Необходимость округления может быть указана в теле кода команды. Когда выбрана опция округления, регистр результата содержит округлённый 16-разрядный результат; значение аккумулятора никогда не округляется.

Аккумулятор использует схему несмещённого округления. Стандартный метод смещённого округления заключается в прибавлении 1 к 15-му разряду в схеме суммирования. При этом возникает результирующее положительное смещение, так как значение, соответствующее середине диапазона (`A0.L/A1.L` = 0x8000), всегда округляется в большую сторону.

Аккумулятор устраняет смещение принудительной записью нуля в 16-ый бит результата при обнаружении точки середины диапазона. Эта операция приводит к округлению нечётных значений `A0.L/A1.L` к старшему, а чётных – к младшему значению, что в свою очередь, с учётом предположения о равномерном распределении величин, приводит к нулевому смещению для больших длин выборок.

В последующих примерах символ “*x*” используется для обозначения произвольного набора битов (не все нули). На рис. 2-6 приведён пример типичной операции округления `A0`; данный пример также распространяется на `A1`.

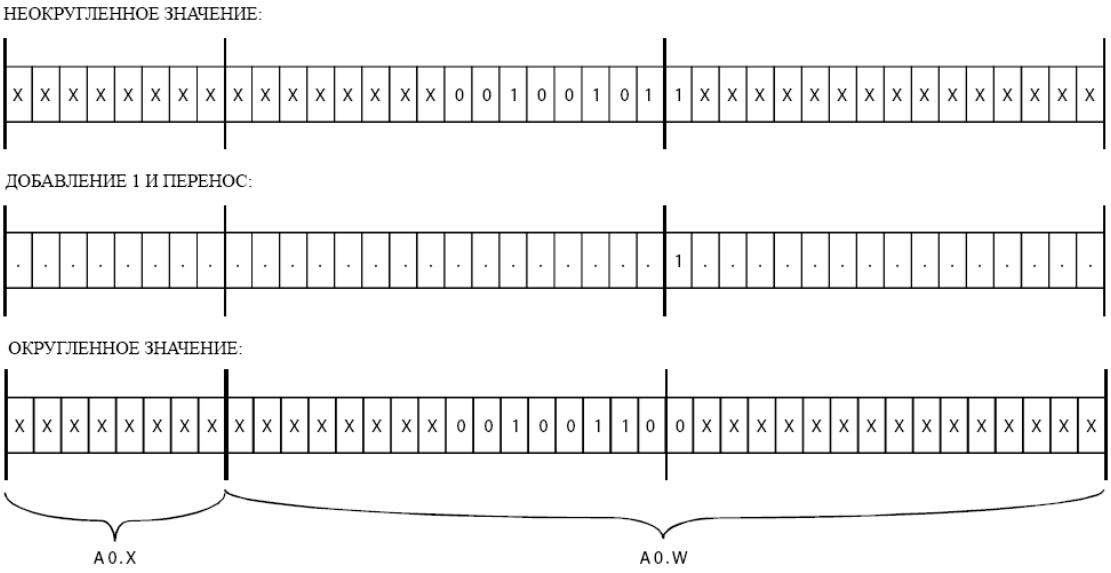

Рис. 2-6. Типичный пример несмещённого округления в умножителе

Компенсация, вносимая для устранения результирующего смещения, становится очевидной, когда все младшие 15 битов содержат нули, а 15-ый бит – единицу (точка середины диапазона), как показано на рис. 2-7.

На рис. 2-7 16-ый бит регистра `A0` принудительно обнуляется. Этот алгоритм используется в каждой операции округления, но становится очевиден, только когда содержимое младших 16 разрядов соответствует случаю, показанному в следующем примере.

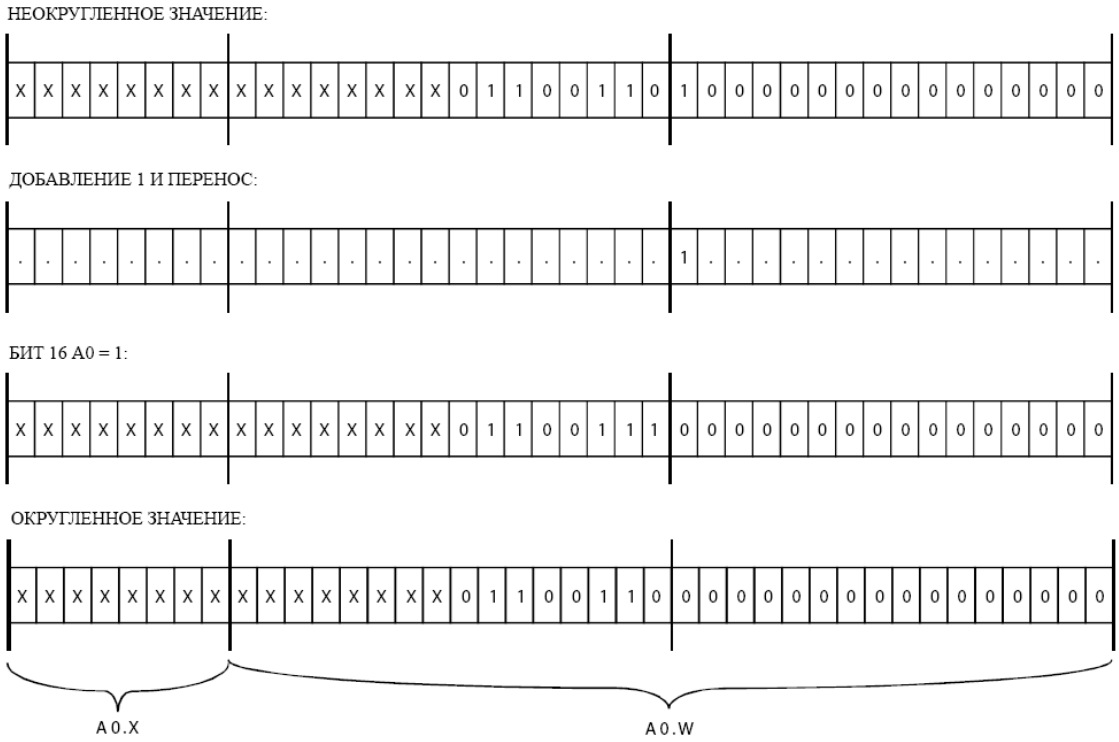

Рис. 2-7. Устранение результирующего смещения при несмещённом округлении в умножителе

Смещённое округление

Метод *округления к ближнему* также возвращает ближайшее к округляемому числу значение. Однако, в данном методе число, находящееся точно на границе между двумя числами, всегда округляется к большему. Например, при округлении 3-разрядного дробного числа 0.25 в дополнительном коде (0.01 в двоичной записи) до ближайшего 2-разрядного дробного числа в дополнительном коде, этот метод возвращает 0.5 (0.1 в двоичной записи). Исходное дробное число находится точно на границе между 0.5 и 0.0 (0.0 в двоичной записи), таким образом, этот метод производит округление к большему. Этот метод называется *смещённым округлением*, так как он всегда округляет числа в большую сторону.

Использование смещённого округления разрешается битом `RND_MOD` регистра `ASTAT`. При сброшенном бите `RND_MOD` опция `RND` команды умножителя использует нормальную операцию несмещённого округления, описанную в разделе “Несмещённое округление”.

Когда бит `RND_MOD` установлен (=1), процессор использует смещённое округление вместо несмещённого. В режиме смещённого округления, все значения, у которых `A0.l/A1.L` равно 0x8000, округляются в большую сторону, вместо округления в большую сторону только для нечётных значений. Пример смещённого округления приведён в таблице 2-7.

Таблица 2-7. Смещённое округление в умножителе

A0/A1 до выполнения округления	Результат при смещённом округлении	Результат при несмещённом округлении
0x00 0000 8000	0x00 0001 8000	0x00 0000 0000
0x00 0001 8000	0x00 0002 0000	0x00 0002 0000
0x00 00008001	0x00 0001 0001	0x00 0001 0001
0x00 0001 8001	0x00 0002 0001	0x00 0002 0001
0x00 0000 7FFF	0x00 0000 FFFF	0x00 0000 FFFF
0x00 0001 7FFF	0x00 0001 FFFF	0x00 0001 FFFF

Смещённое округление влияет на результат только в случае, когда регистр `A0.L/A1.L` содержит 0x8000; во всех остальных случаях операция округления работает обычным образом. Этот режим позволяет более эффективно реализовывать бит-ориентированные алгоритмы, реализующие смещённое округление (например, процедура сжатия речи в стандарте GSM).

Усечение

Другим общепринятым способом снижения количества значащих разрядов в записи числа является простое маскирование младших N-M разрядов. Этот процесс называется *усечением* и приводит к относительно большому смещению. Усечение используется в командах, не поддерживающих округление. Бит `RND_MOD` в регистре `ASTAT` не влияет на операцию усечения.

Специальные команды округления

АЛУ обеспечивает возможность округления арифметических результатов прямо в регистре данных с помощью операций смещённого и несмещённого округления, описанных ранее. Оно также обеспечивает возможность округления по различным границам разрядов. При использовании опций `RND12`, `RND` и `RND20` выполняется смещённое округление по границе 12-го, 16-го и 20-го разрядов, соответственно, независимо от состояния бита `RND_MOD` в регистре `ASTAT` с последующим помещением результата в полуслово.

Например:

```
R3.L = R4 (RND);
```

выполняется смещённое округление по границе 16-го разряда с помещением результата в полуслово.

```
R3.L = R4 + R5 (RND12);
```

выполняется сложение двух 32-разрядных чисел и смещённое округление по границе 12-го разряда с помещением результата в полуслово.

```
R3.L = R4 + R5 (RND 20);
```

выполняется сложение двух 32-разрядных чисел и смещённое округление по границе 20-го разряда с помещением результата в полуслово.

Использование информации арифметического состояния

Умножитель, АЛУ и устройство сдвига обновляют флаг переполнения и другие флаги в регистре арифметического состояния (ASTAT) процессора. Флаги состояния, получаемые по результатам вычислительных операций, могут использоваться для проверки условий в процессе выполнения программы. Для этого следует применять условные команды проверки флага CC регистра ASTAT после выполнения команды вычислительного устройства. Этот метод позволяет осуществлять контроль результатов выполнения команд. Регистр ASTAT представляет собой 32-разрядный регистр, некоторые биты в котором зарезервированы. Для совместимости с последующими разработками, при записи в этот регистр в позиции зарезервированных битов необходимо записывать значения, полученные при чтении.

Регистр арифметического состояния (ASTAT)

На рис. 2-8 показан регистр ASTAT. Процессор обновляет биты состояния регистра ASTAT, отображая состояние последней операции АЛУ, умножителя или устройства сдвига.

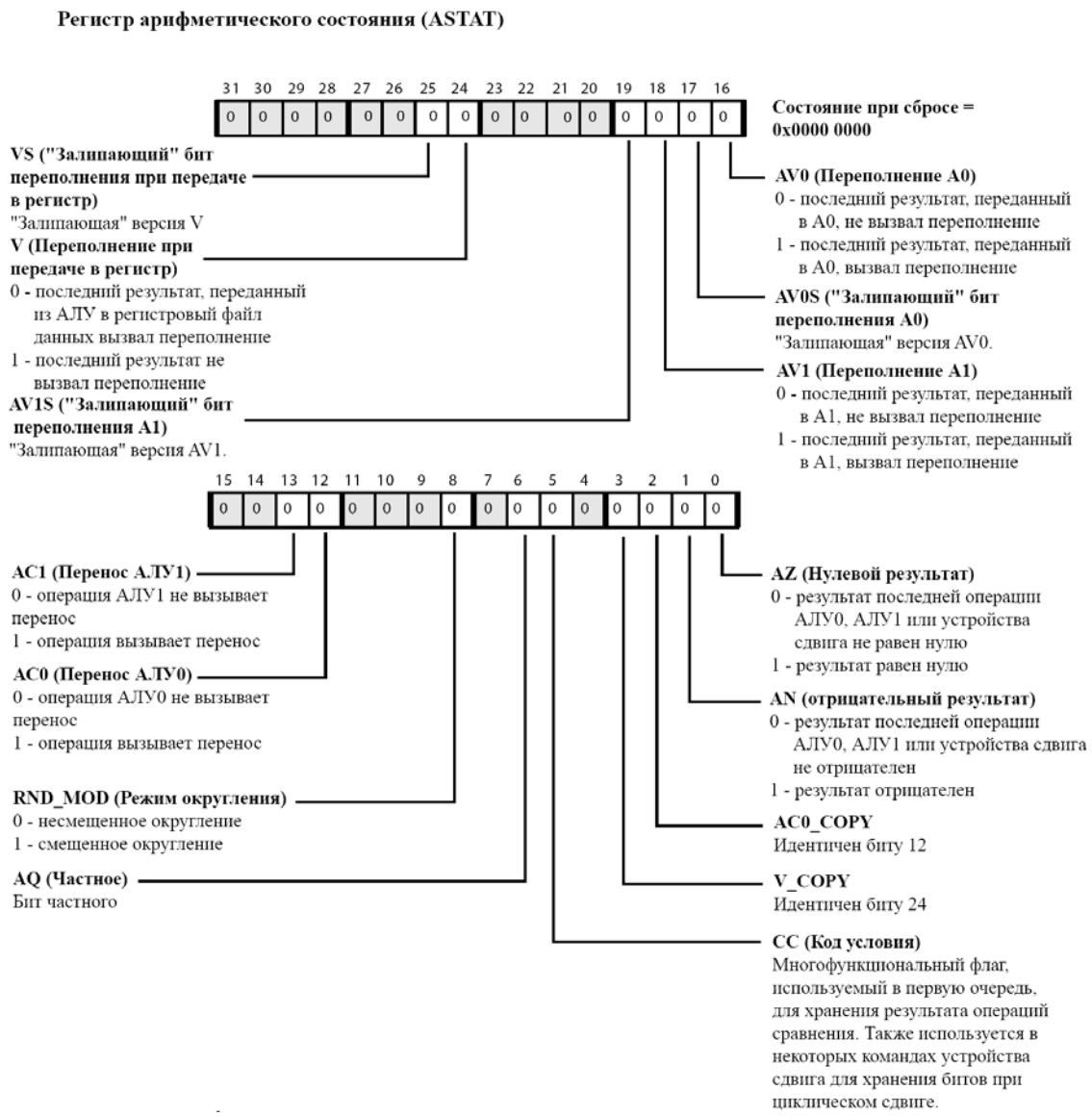

Рис. 2-8. Регистр арифметического состояния

Арифметико-логическое Устройство (АЛУ)

Два АЛУ выполняют арифметические и логические операции над данными с фиксированной точкой. Команды АЛУ с фиксированной точкой оперируют над 16-разрядными, 32-разрядными и 40-разрядными операндами с фиксированной точкой и выдают 16-разрядные, 32-разрядные и 40-разрядные результаты с фиксированной точкой. Команды АЛУ:

- сложение и вычитание регистров с фиксированной точкой;
- сложение и вычитание явно заданных значений;
- накопление и вычитание результатов умножителя;
- логические операции: AND, OR, NOT, XOR, побитовое XOR, взятие отрицательного значения;
- функции: ABS, MAX, MIN, округление, примитивы деления.

Операции АЛУ

Первичные операции АЛУ выполняются в АЛУ0, параллельные операции выполняются с использованием АЛУ1, которое выполняет поднабор операций АЛУ0.

В таблице 2-8 описаны возможные комбинации входных и выходных данных каждого АЛУ.

Таблица 2-8. Входные и выходные данные каждого АЛУ.

Вход	Выход
Два или четыре 16-разрядных операнда	Один или два 16-разрядных результата
Два 32-разрядных операнда	Один 32-разрядный результат
32-разрядный результат умножителя	Комбинация 32-разрядного результата умножителя и 40-разрядного результата аккумулятора

Объединяя вычисления в обоих АЛУ, при помощи одной команды можно получить четыре 16-разрядных результата, два 32-разрядных результата или два 40-разрядных результата.

Простые 16-разрядные операции

В простых 16-разрядных операциях в качестве входа АЛУ могут использоваться любые две 16-разрядные половины регистра. При сложении, вычитании или логической операции формируется 16-разрядный результат, помещаемый в произвольную половину регистра. Для таких операций используется АЛУ0, так как оно является первичным ресурсом для выполнения операций АЛУ.

Например:

```
R3.H = R1.H + R2.L (NS);
```

выполняется сложение 16-разрядного содержимого `R1.H` (старшая половина `R1`) с содержимым `R2.L` (младшая половина `R2`), результат помещается в `R3.H` (старшая половина `R3`), насыщение не производится.

Двойные 16-разрядные операции АЛУ

В двойных 16-разрядных операциях в качестве входов АЛУ могут использоваться любые 32-разрядные регистры, представляющие в данном случае пару 16-разрядных операндов. При сложении, вычитании или выполнении логической операции формируется два 16-разрядных результата, помещаемые в произвольный 32-разрядный регистр. Для таких операций используется АЛУ0, так как оно является первичным ресурсом для выполнения операций АЛУ.

Например:

```
R3 = R1 +|- R2 (S);
```

выполняется сложение 16-разрядного содержимого `R2.H` (старшая половина `R2`) с содержимым `R1.H` (старшая половина `R1`), результат помещается в `R3.H` (старшая половина `R3`), используется насыщение.

Также при выполнении этой команды производится вычитание 16-разрядного содержимого `R2.L` (младшая половина `R2`) из содержимого `R1.L` (младшая половина `R1`), результат помещается в `R3.L` (младшая половина `R3`), используется насыщение (см. рис. 2-10).

Счетверённые 16-разрядные операции

В счетверённых 16 разрядных операциях в качестве входов АЛУ0 и АЛУ1 могут использоваться любые два 32-разрядных регистра, представляющие пару 16-разрядных операндов. Некоторые операции сложения или вычитания позволяют получить четыре 16-разрядных результата, помещаемые в два произвольных 32-разрядных регистра. В этих операциях задействованы оба АЛУ. Так как Регистровый Файл Данных соединён с арифметическими устройствами только двумя 32-разрядными шинами данных, на вход АЛУ1 поступают те же две пары 16-разрядных входных значений, что и на вход АЛУ0. Конструкция команды при этом идентична конструкции команд при выполнении двойных 16-разрядных операций; входные операнды должны быть одинаковы для обоих АЛУ.

Например:

```
R3 = R0 +|+ R1, R2 = R0 -|- R1 (S);
```

Выполняются следующие четыре операции:

- Сложение 16-разрядного содержимого `R1.H` (старшая половина `R1`) с 16-разрядным содержимым `R0.H` (старшая половина `R0`) и помещение результата в `R3.H` с насыщением.
- Сложение `R1.L` с `R0.L` и помещение результата в `R3.L` с насыщением.
- Вычитание 16-разрядного содержимого `R1.H` (старшая половина `R1`) из 16-разрядного содержимого `R0.H` (старшая половина `R0`) и помещение результата в `R2.H` с насыщением.
- Вычитание `R1.L` из `R0.L` и помещение результата в `R2.L` с насыщением.

Эта операция эквивалентна следующим четырём командам:

```
R3.H = R0.H + R1.H (S);
R3.L = R0.L + R1.L (S);
R2.H = R0.H - R1.H (S);
R2.L = R0.L - R1.L (S);
```

Простые 32-разрядные операции

В простых 32-разрядных операциях в качестве входов АЛУ могут использоваться любые два 32-разрядных регистра, представляющие 32-разрядные операнды. При выполнении сложения, вычитания или логических операций выдаётся 32-разрядный результат, помещаемый в произвольный 32-разрядный регистр. Для таких операций используется АЛУ0, так как оно является первичным ресурсом для выполнения операций АЛУ.

Помимо возможности извлечения 32-разрядных входных операндов из регистрового файла данных, они также могут извлекаться из регистрового файла указателей, состоящего из регистров `P[5:0]`, `SP`, `FP`, и помещаться в него.

Смешанное использование регистров указателей и регистров данных в командах невозможно.

Например:

```
R3 = R1 + R2 (NS);
```
выполняется сложение 32-разрядного содержимого `R2` с 32-разрядным содержимым `R1`, результат помещается в `R3`, насыщение не производится.

```
R3 = R1 + R2 (S);
```
выполняется сложение 32-разрядного содержимого `R2` с 32-разрядным содержимым `R1`, результат помещается в `R3`, производится насыщение.

Двойные 32-разрядные операции

В двойных 32-разрядных операциях в качестве входов АЛУ0 и АЛУ1 могут использоваться любые два 32-разрядных регистра, представляющие пару 32-разрядных операндов,. При сложении или вычитании формируется два 32-разрядных результата, помещаемые в два 32-разрядных регистра. В этих операциях задействованы оба АЛУ. Так как Регистровый Файл Данных соединён с арифметическими устройствами только двумя 32-разрядными шинами данных, на вход АЛУ1 поступают те же два 32-разрядных регистра, что и на вход АЛУ0.

Например:

```
R3 = R1 + R2, R4 = R1 – R2 (NS);
```

выполняется сложение 32-разрядного содержимого `R2` с 32-разрядным содержимым регистра `R1`, результат помещается в `R3`, насыщение не производится.

Также по этой команде выполняется вычитание 32-разрядного содержимого `R2` из `R1`, результат помещается в `R4`, насыщение не производится.

Существует специальный вариант этой команды, в котором в качестве входных операндов используются 40-разрядные результаты АЛУ, а результатом является сумма и разность регистров A0 и A1.

Например:

```
R3 = A0 + A1, R4 = A0 - A1 (S);
```

В регистры результата передаётся два 32-разрядных значения суммы и разности регистров АЛУ, выполняется насыщение.

Обзор команд АЛУ

В таблице 2-9 перечислены команды АЛУ. Дополнительную информацию о синтаксисе языка ассемблера и влиянии команд АЛУ на флаги состояния см. в *Справочном руководстве по набору команд процессора Blackfin ADSP-BF53x*.

В таблице 2-9 используются следующие обозначения:

- Dreg – любой регистр регистрового файла данных.
- Preg – любой регистр указателя, регистр FP или SP.
- Dreg_lo_hi – любая 16-разрядная половина регистра любого регистра регистрового файла данных.
- Dreg_lo – младшие 16 разрядов любого регистра регистрового файла данных.
- Imm7 – явно заданное знаковое 7-разрядное число.
- An – любой регистр результата АЛУ A0 или A1.
- DIVS – примитив знака при делении.
- DIVQ – примитив частного при делении.
- MAX – максимальное (наиболее положительное) из значений сравниваемых регистров.
- MIN – минимальное из значений сравниваемых регистров.
- ABS – абсолютное значение верхней и нижней половин 32-разрядного регистра.
- RND – округление полуслова.
- RND12 – насыщение результата сложения или вычитания и округление полученного значения по границе 12-го бита.
- RND20 – насыщение результата сложения или вычитания и округление полученного значения по границе 20-го бита.
- SIGNBITS – значение, на единицу меньшее количества знаковых разрядов в числе.
- EXPADJ – меньшее из значения, на единицу меньшего количества знаковых разрядов в числе, и значения порога.
- “*” – флаг может устанавливаться или сбрасываться в зависимости от результата выполнения команды.
- “**”– флаг сбрасывается.
- “–“ – команда не влияет на флаги.
- “*d*” – AQ содержит результат операции “исключающее ИЛИ” старшего бита делимого и старшего бита делителя.

Таблица 2-9. Обзор команд АЛУ

Команда	Флаги статуса регистра ASTAT						
	AZ	AN	AC0 AC0_COPY AC1	AV0 AV0S	AV1 AV1S	V V_COPY VS	AQ
Preg = Preg + Preg ;	–	–	–	–	–	–	–
Preg += Preg ;	–	–	–	–	–	–	–
Preg -= Preg ;	–	–	–	–	–	–	–
Dreg = Dreg + Dreg ;	*	*	*	–	–	*	–
Dreg = Dreg – Dreg (S) ;	*	*	*	–	–	*	–
Dreg = Dreg + Dreg, Dreg = Dreg – Dreg ;	*	*	*	–	–	*	–
Dreg_lo_hi = Dreg_lo_hi + Dreg_lo_hi ;	*	*	*	–	–	*	–
Dreg_lo_hi = Dreg_lo_hi – Dreg_lo_hi (S) ;	*	*	*	–	–	*	–
Dreg = Dreg +|+ Dreg ;	*	*	*	–	–	*	–
Dreg = Dreg +|- Dreg ;	*	*	*	–	–	*	–
Dreg = Dreg -|+ Dreg ;	*	*	*	–	–	*	–
Dreg = Dreg -|- Dreg ;	*	*	*	–	–	*	–
Dreg = Dreg +|+ Dreg, Dreg = Dreg -|- Dreg ;	*	*	–	–	–	*	–
Dreg = Dreg +|- Dreg, Dreg = Dreg -|+ Dreg ;	*	*	–	–	–	*	–
Dreg = An + An, Dreg = An – An ;	*	*	*	–	–	*	–
Dreg += imm7 ;	*	*	*	–	–	*	–
Preg += imm7 ;	–	–	–	–	–	–	–
Dreg = (A0 += A1) ;	*	*	*	*	–	*	–
Dreg_lo_hi = (A0 += A1) ;	*	*	*	*	–	*	–
A0 += A1 ;	*	*	*	*	–	–	–
A0 -= A1;	*	*	*	*	–	–	–
DIVS (Dreg, Dreg) ;	*	*	*	*	–	–	d
DIVQ (Dreg, Dreg);	*	*	*	*	–	–	d
Dreg = MAX (Dreg , Dreg) (V);	*	*	–	–	–	**/–	–
Dreg = MIN (Dreg, Dreg) (V) ;	*	*	–	–	–	**/–	–
Dreg = ABS Dreg (V) ;	*	**	–	–	–	*	–
An = ABS An ;	*	**	–	*	*	*	–
An = ABS An, An = ABS An ;	*	**	–	*	*	*	–
An = -An ;	*	*	*	*	*	*	–

Таблица 2-9. Обзор команд АЛУ (продолжение)

Команда	Флаги состояния регистра ASTAT						
	AZ	AN	AC0 AC0_COPY AC1	AV0 AV0S	AV1 AV1S	V V_COPY VS	AQ
An = -An, An = -An ;	*	*	*	*	*	*	–
An = An (S) ;	*	*	–	*	*	–	–
An = An (S), An = An (S) ;	*	*	–	*	*	–	–
Dreg_lo_hi = Dreg (RND);	*	*	–	–	–	*	–
Dreg_lo_hi = Dreg + Dreg (RND12) ;	*	*	–	–	–	*	–
Dreg_lo_hi = Dreg – Dreg (RND12) ;	*	*	–	–	–	*	–
Dreg_lo_hi = Dreg + Dreg (RND20) ;	*	*	–	–	–	*	–
Dreg_lo_hi = Dreg – Dreg (RND20) ;	*	*	–	–	–	*	–
Dreg_lo = SIGNBITS Dreg;	–	–	–	–	–	–	–
Dreg_lo = SIGNBITS Dreg_lo_hi;	–	–	–	–	–	–	–
Dreg_lo = SIGNBITS An ;	–	–	–	–	–	–	–
Dreg_lo = EXPADJ (Dreg, Dreg_lo) (V) ;	–	–	–	–	–	–	–
Dreg_lo = EXPADJ (Dreg_lo_hi, Dreg_lo) ;	–	–	–	–	–	–	–
Dreg = Dreg & Dreg ;	*	*	**	–	–	**/–	–
Dreg = ~ Dreg ;	*	*	**	–	–	**/–	–
Dreg = Dreg | Dreg ;	*	*	**	–	–	**/–	–
Dreg = Dreg ^ Dreg ;	*	*	**	–	–	**/–	–
Dreg =– Dreg ;	*	*	*	–	–	*	–

Поток данных в АЛУ

На рис. 2-9 представлена более подробная, по сравнению с рис. 2-1, схема арифметических устройств и регистрового файла данных.

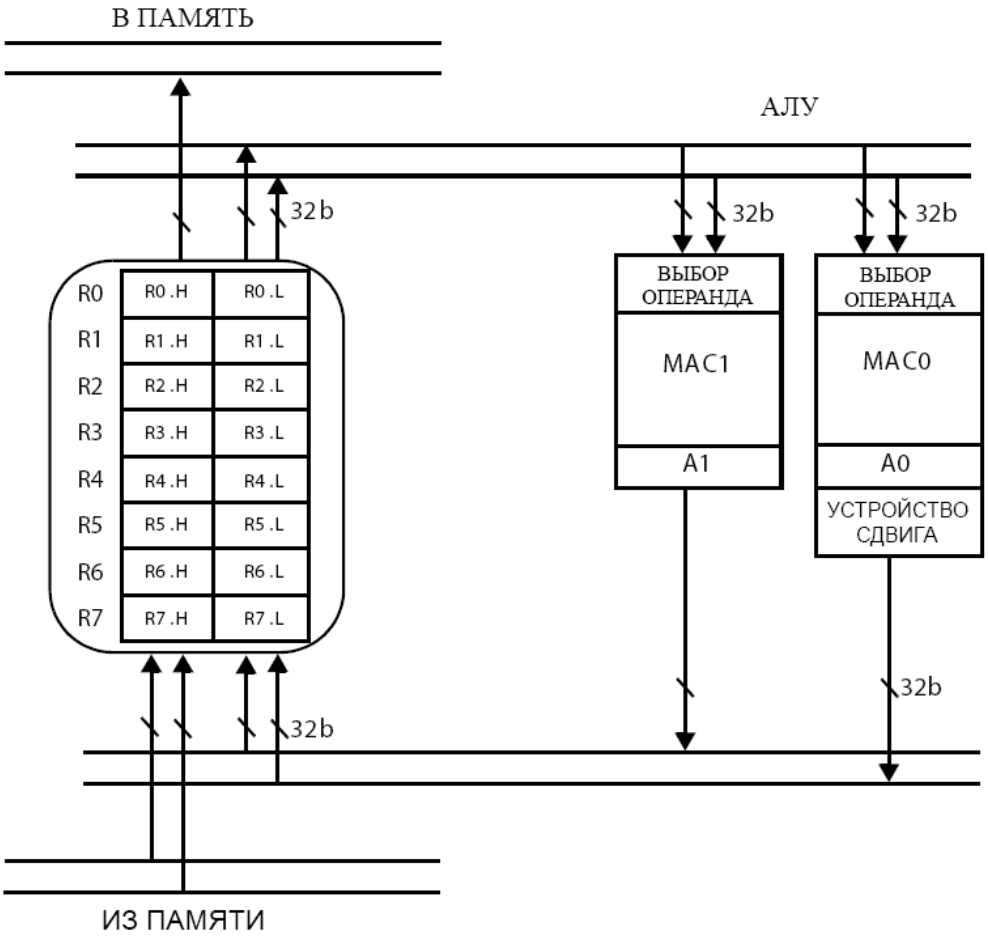

Рис. 2-9. Регистровые файлы и АЛУ

Для удобства далее описывается АЛУ0. Устройство АЛУ1 очень похоже – оно является поднабором АЛУ0.

Каждое АЛУ выполняет 40-разрядное сложение при накоплении результатов умножителя, а также 32-разрядные и двойные 16-разрядные операции. Каждое АЛУ содержит два 32-разрядных входных порта, которые могут рассматриваться как пара 16-разрядных операндов или один 32-разрядный операнд. В простых 16-разрядных операциях любой из четырёх возможных входных 16-разрядных операндов может использоваться совместно с любым другим 16-разрядным операндом, присутствующим на входе АЛУ.

Как показано на рис. 2-10, при двойных 16-разрядных операциях, старшие и младшие половины группируются, обеспечивая четыре возможные комбинации операций сложения и вычитания:

```
(A) H+H, L+L    (B) H+H, L-L
(C) H-H, L+L    (D) H-H, L-L
```

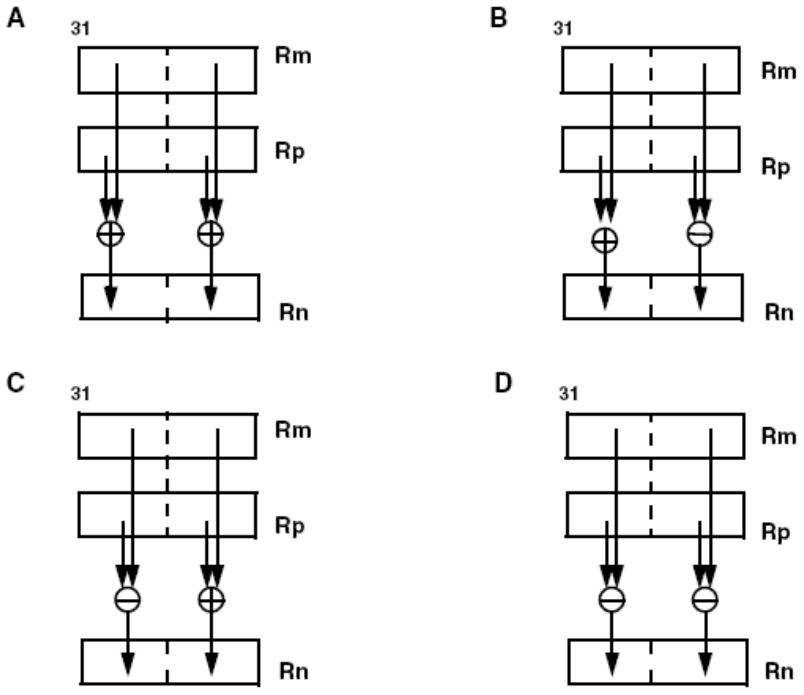

Рис. 2-10. Двойные 16-разрядные операции АЛУ

Опция перекрещивания в двойных 16-разрядных операциях

При выполнении двойных 16-разрядных операций результаты могут “перекрещиваться”. “Перекрещивание результатов” изменяет порядок размещения результата вычислений в регистре результата. Обычно, результат старшей части вычислений помещается в старшую половину регистра результата, результат младшей части вычислений помещается в младшую половину регистра результата. При использовании опции перекрещивания старший результат помещается в младшую половину регистра, а младший результат помещается в старшую половину регистра (см. рис. 2-11). Это в частности полезно при операциях с комплексными числами и при вычислении быстрого преобразования Фурье (БПФ). Опция перекрещивания доступна только при работе с АЛУ0.

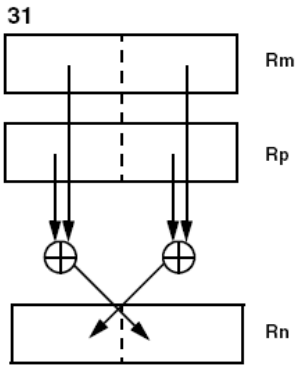

Рис. 2-11. Опция перекрещивания в двойных 16-разрядных операциях АЛУ

Сигналы состояния АЛУ

Каждое АЛУ генерирует шесть сигналов состояния: нулевого результата (AZ), отрицательного результата (AN), переноса (ACn), “залипшего” переполнения (AVnS), мгновенного переполнения (AVn) и частного (AQ). Все сигналы арифметического состояния фиксируются в регистре арифметического состояния (ASTAT) в конце такта. Влияние команд АЛУ на флаги состояния см. в таблице 2-9.

В зависимости от команды входные данные могут поступать из регистрового файла данных, регистрового файла указателей и регистров арифметического результата. Операции АЛУ с повышенной точностью поддерживаются 32-разрядной арифметикой.

Поддержка деления в АЛУ

АЛУ поддерживает деление с использованием двух специальных примитивов деления. При помощи этих команд (DIS, DIVQ) программы могут реализовывать условный (с проверкой ошибок) алгоритм деления без восстановления, основанный на сложении вычитании.

Деление может быть либо знаковым, либо беззнаковым, при этом делимое и делитель должны быть одного типа. Подробности использования операции деления и примеры программ приведены в *Справочном руководстве пользователя по набору команд процессора Blackfin ADSP-BF53x*.

Специальные SIMD операции видео АЛУ

Четыре 8-разрядных видео АЛУ позволяют процессору обрабатывать видеоинформацию с высокой эффективностью. Каждая команда видео АЛУ может принимать от одной до четырёх пар 8-разрядных входных данных и возвращать от одного до четырёх 8-разрядных результатов. Данные на вход видео АЛУ поступают в виде двух 32-разрядных слов из Регистрового файла данных. Возможные операции видео АЛУ:

- счетверённое 8-разрядное сложение или вычитание;
- счетверённое 8-разрядное усреднение;
- счетверённое 8-разрядное упаковывание и распаковывание;
- счетверённое 8-разрядное вычитание - взятие абсолютного значения - накопление;
- выравнивание байтов.

Дополнительную информацию о работе этих команд см. в *Справочном руководстве пользователя по набору команд процессора Blackfin ADSP-BF53x*.

Умножители-накопители (Умножители)

Два умножителя (MAC0 и MAC1) выполняют операции умножения и умножения-накопления с фиксированной точкой. Операция умножения-накопления может выполняться с использованием кумулятивного суммирования или кумулятивного вычитания.

Команды умножителя оперируют над 16-разрядными данными с фиксированной точкой и формируют 32-разрядные результаты, которые могут прибавляться к содержимому 40-разрядного аккумулятора или вычитаться из него.

Входные данные обрабатываются как дробные или целые, беззнаковые или в дополнительном коде. Команды умножителя:
- умножение;
- умножение-накопление со сложением, необязательное округление;
- умножение-накопление с вычитанием, необязательное округление;
- двойные версии перечисленных выше команд.

Работа умножителя

Каждый умножитель имеет два 32-разрядных входа, по которым он получает два 16-разрядных операнда. В простых командах умножения-накопления этими операндами могут являться любые регистры регистрового файла данных. Каждый умножитель может выполнять накопление результатов в соответствующих регистрах аккумулятора, `A1` или `A0`. Результаты аккумулятора могут насыщаться до 32 или 40 разрядов. Результат умножителя также может быть напрямую записан в 16- или 32-разрядный регистр с необязательным выполнением округления.

В каждой команде умножителя указывается, являются ли оба входных числа целыми или дробными. Формат результата соответствует формату входных данных. В MAC0 оба входа обрабатываются как знаковые или беззнаковые. В MAC1 существует опция смешанного режима.

Если оба входных числа дробные и знаковые, умножитель автоматически сдвигает результат на один разряд влево для устранения избыточного знакового разряда. В режимах работы с беззнаковыми дробными числами, целыми числами, и в смешанном режиме сдвиг с целью коррекции знакового бита не выполняется. Формат входных данных определяется опциями команд умножителя. Дополнительную информацию см. в разделе “Опции команд умножителя”.

Размещение результатов умножителя в регистрах аккумуляторов

Как показано на рис. 2-9, каждому умножителю соответствует аккумулятор, A0 или A1. Регистр каждого аккумулятора разделён на три части – `A0.L/A1.L` (биты 15:0), `A0.H/A1.H` (биты 31:16) и `A0.X/A1.X` (биты 39:32).

Когда умножитель выполняет запись в регистры результата аккумулятора, 32-разрядный результат размещается в младших разрядах объединённого регистра аккумулятора, старшие восемь битов регистра (A0.X/A1.X) заполняются знаковыми разрядами в соответствии со значением старшего разряда результата.

Выходное значение умножителя может помещаться не только в регистры A0 или A1, а также в различные 16- или 32-разрядные регистры регистрового файла данных.

Округление и насыщение результатов умножителя

В операции умножения-накопления данные аккумулятора могут подвергаться насыщению и необязательному округлению при помещении в регистр или половину регистра. Когда при умножении в регистр или половину регистра помещается только результат умножения, операции насыщения и округления эквивалентны. Операции округления и насыщения выполняются следующим образом:

- Округление применяется только к дробным результатам, за исключением случая использования опции IH, когда производится округление и извлечение старшей половины целочисленного результата.
  При использовании опции IH округлённый результат получается путём прибавления 0x8000 к содержимому аккумулятора (в операции умножения-накопления) или результату умножения (в операции умножения) и последующего насыщения до 32-ух разрядов. Дополнительную информацию см. в разделе “Округление результатов умножителя”.
- Если произошло переполнение или потеря значащих разрядов, при выполнении операция насыщения в определённый регистр результата записывается максимальное положительное или отрицательное значение. Дополнительную информацию см. в следующем разделе.

Насыщение результатов умножителя при переполнении

Три бита в регистре ASTAT отображают состояние переполнения умножителя:

- В бит 16 (AV0) и бит 18 (AV1) записывается условие переполнения (превысил ли результат границу 32-разрядного значения) для аккумуляторов A0 и A1, соответственно.
  Когда бит сброшен (=0), то переполнения или потери значащих разрядов не произошло. Когда бит установлен (=1), произошло переполнение или потеря значащих разрядов. Биты AV0S и AV1S – “залипающие” версии битов AV0 и AV1.
- Биты 24 (V) и 25 (VS) устанавливаются, когда переполнение происходит при передаче результата аккумулятора в регистр.

Обзор команд умножителя

В таблице 2-10 перечислены команды умножителя. Дополнительную информацию о синтаксисе языка ассемблера и влиянии команд умножителя на флаги состояния см. в *Справочном руководстве по набору команд процессора Blackfin ADSP-BF53x*.

В таблице 2-10 используются следующие символы:

- Dreg – любой регистр Регистрового Файла Данных.
- Dreg_lo_hi – любая 16-разрядная половина любого регистра Регистрового Файла Данных.
- Dreg_lo – младшие 16 разрядов любого регистра Регистрового Файла Данных.
- Dreg_hi – старшие 16 разрядов любого регистра Регистрового Файла Данных.
- An – любой регистр аккумулятора MAC, `A0` или `A1`.
- "*" – флаг может устанавливаться или сбрасываться, в зависимости от команды.
- "–" – команда не влияет на флаги.

Опции команд умножителя описаны в следующем разделе.

Таблица 2-10. Обзор команд умножителя

Команда	Флаги состояния регистра ASTAT		
	AV0 AV0S	AV1 AV1S	V V_COPY VS
Dreg_lo = Dreg_lo_hi * Dreg_lo_hi ;	–	–	*
Dreg_hi = Dreg_lo_hi * Dreg_lo_hi ;	–	–	*
Dreg = Dreg_lo_hi * Dreg_lo_hi ;	–	–	*
A*n* = Dreg_lo_hi * Dreg_lo_hi ;	*	*	–
A*n* += Dreg_lo_hi * Dreg_lo_hi ;	*	*	–
A*n* -= Dreg_lo_hi * Dreg_lo_hi ;	*	*	–
Dreg_lo = (A0 = Dreg_lo_hi * Dreg_lo_hi) ;	*	*	*
Dreg_lo = (A0 += Dreg_lo_hi * Dreg_lo_hi) ;	*	*	*
Dreg_lo = (A0 -= Dreg_lo_hi * Dreg_lo_hi) ;	*	*	*
Dreg_hi = (A1 = Dreg_lo_hi * Dreg_lo_hi) ;	*	*	*
Dreg_hi = (A1 += Dreg_lo_hi * Dreg_lo_hi) ;	*	*	*
Dreg_hi = (A1 -= Dreg_lo_hi * Dreg_lo_hi) ;	*	*	*
Dreg = (A*n* = Dreg_lo_hi * Dreg_lo_hi) ;	*	*	*
Dreg = (A*n* += Dreg_lo_hi * Dreg_lo_hi) ;	*	*	*
Dreg = (A*n* -= Dreg_lo_hi * Dreg_lo_hi) ;	*	*	*
Dreg *= Dreg ;	–	–	–

Опции команд умножителя

Приведённое ниже описание опций команд умножителя носит обзорный характер. Не все опции могут использоваться в каждой команде. Информацию об использовании этих опций и соответствующих команд см. в *Справочном руководстве по набору команд процессора Blackfin ADSP-BF53x*.

Режим работы по умолчанию	Не является опцией; входные данные представлены дробными знаковыми числами
(IS)	Входные операнды – знаковые целые числа. Коррекция сдвигом не производится.
(FU)	Входные операнды – беззнаковые дробные числа. Коррекция сдвигом не производится.
(IU)	Входные операнды – беззнаковые целые числа. Коррекция сдвигом не производится.
(T)	Входные операнды – знаковые дробные числа. При копировании в половину регистра содержимое аккумулятора усекается отбрасыванием младших 16 разрядов.
(TFU)	Входные операнды – беззнаковые дробные числа. При копировании в половину регистра содержимое накопителя усекается отбрасыванием младших 16 разрядов.
(S2RND)	При умножении и накоплении с передачей в регистр: Входные операнды – знаковые дробные числа. При копировании в регистр содержимое аккумулятора масштабируется (умножается на 2 путём сдвига влево на одну позицию) и округляется. Если при масштабировании и округлении получается знаковое число, для представления которого требуется более 32-ух разрядов, оно насыщается. При умножении и накоплении с передачей в половину регистра: Входные операнды – знаковые дробные числа. При копировании в регистр содержимое аккумулятора масштабируется (умножается на 2 путём сдвига влево на одну позицию), 16 старших разрядов округляются перед усечением младших 16 разрядов аккумулятора. Если при масштабировании и округлении получается знаковое число, для представления которого требуется более 16 разрядов, оно насыщается.
(ISS2)	При умножении и накоплении с передачей в регистр: Входные операнды – знаковые целые числа. При копировании в регистр содержимое аккумулятора масштабируется (умножается

на 2 путём сдвига влево на одну позицию). Если при масштабировании и округлении получается знаковое число, для представления которого требуется более 32-ух разрядов, оно насыщается.

При умножении и накоплении с передачей в половину регистра: При копировании младших 16 разрядов в половину регистра содержимое аккумулятора масштабируется. Если при масштабировании получается знаковое число, для представления которого требуется более 16 разрядов, оно насыщается.

`(IH)` Эта опция указывает на операцию умножения целых чисел с извлечением старшего полуслова. Содержимое аккумулятора насыщается до 32 разрядов, биты [31:16] аккумулятора округляются, после чего передаются в половину регистра.

`(W32)` Входные операнды – знаковые дробные числа без битов расширения аккумуляторов (`A0.X/A1.X`). При необходимости выполняется коррекция произведения посредством сдвига влево. Эта опция применяется в алгоритмах речевых вокодеров предыдущего поколения системы GSM , написанных для использования с 32-разрядными аккумуляторами. Только в этой опции 0x8000 x 0x8000 = 0x7FFF.

`(M)` Режим смешанного умножения. Применяется только в версиях команд для MAC1. Выполняется умножение знакового дробного на беззнаковый дробный операнд без коррекции сдвигом влево. Первый операнд – знаковый; второй операнд – беззнаковый. По умолчанию MAC0 выполняет несмешанное умножение знаковых дробных чисел или чисел в других форматах соответствии с определёнными опциями. Опция (`M`) может использоваться отдельно или совместно с одной из опций указывающих формат.

Поток данных в умножителе

На рис. 2-12 показаны регистровые файлы, АЛУ и умножители-накопители.

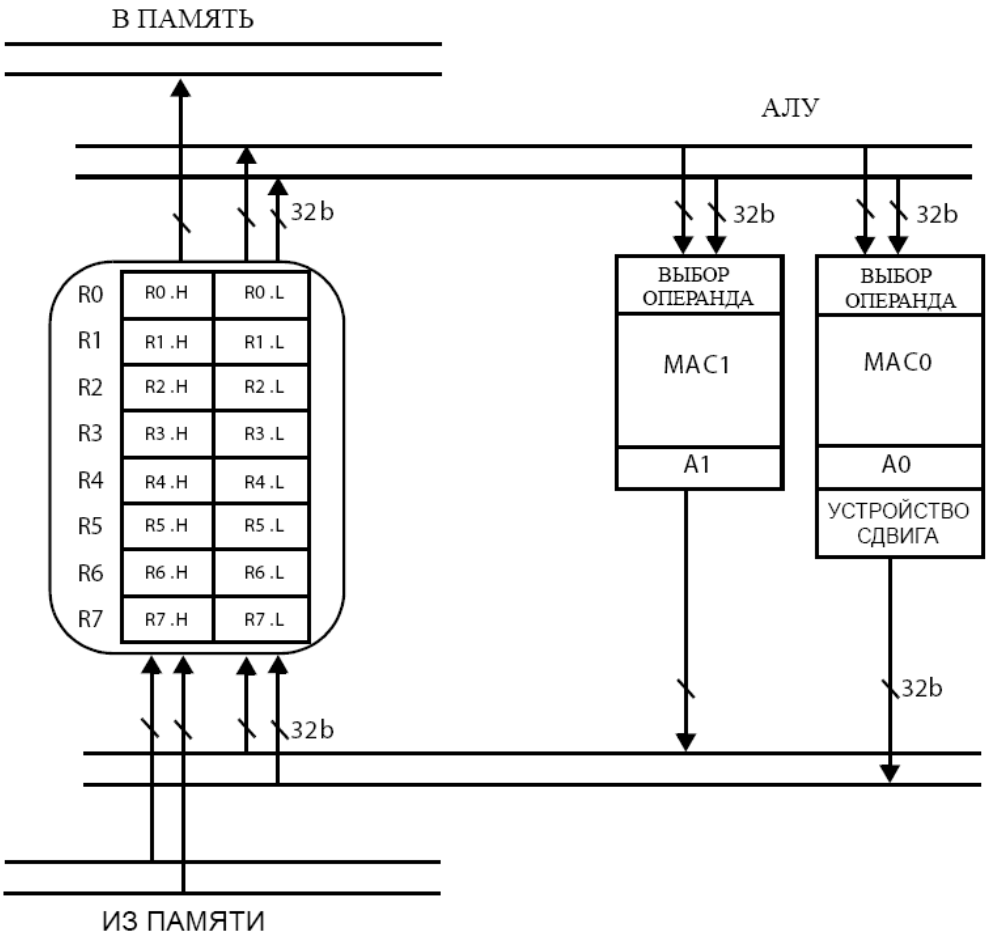

Рис. 2-12. Регистровые файлы и АЛУ

Каждый умножитель имеет два 16-разрядных входа. Он выполняет 16-разрядное умножение и сохраняет результат в 40-разрядном накопителе или извлекает часть результата и помещает её в 16- или 32-разрядный регистр. На входы MAC поступают два 32-разрядных слова, что обеспечивает выбор из четырёх 16-разрядных операндов.

Один из операндов выбирается из старшей или младшей половины одного 32-разрядного слова. Другой операнд должен выбираться из старшей или младшей половины другого 32-разрядного слова. Таким образом, каждый MAC имеет четыре возможные комбинации входных операндов. Два 32-разрядных слова могут содержать информацию из одного регистра, что позволяет организовать возведение в квадрат и умножение старшей половины и младшей половины одного и того же регистра. На рис. 2-13 показаны возможные комбинации входных данных MAC.

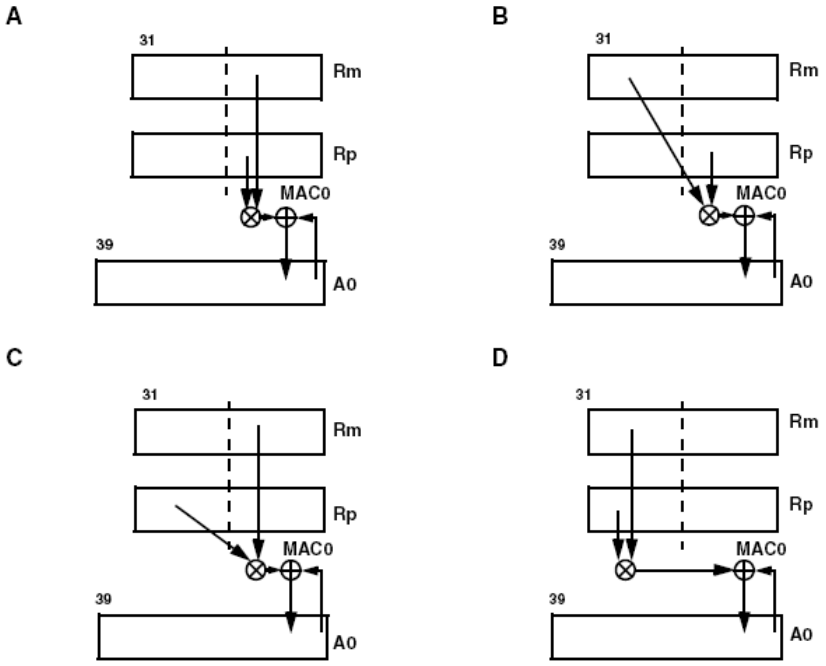

Рис. 2-13. Четыре возможные комбинации операций MAC

32-разрядное произведение передаётся в 40-разрядное устройство сложения/вычитания, которое может выполнять операцию вычитания нового произведения из содержимого регистра результата аккумулятора, складываться с ним или передавать произведение в регистры результата регистрового файла данных. Результаты, содержащиеся в регистрах `A0` и `A1`, имеют разрядность 40 битов. Каждый из этих регистров состоит из 32- и 8-разрядного регистров: `A0.W`, `A1.W`, `A0.X` и `A1.X`.

Примеры некоторых команд:

```
A0 = R3.L * R4.H;
```

По данной команде умножитель/накопитель MAC0 выполняет умножение и помещает результат в регистр аккумулятора.

```
A1 += R3.H * R4.H ;
```

По данной команде умножитель/накопитель MAC1 выполняет умножение и прибавляет результат к предыдущему результату, содержащемуся в аккумуляторе A1.

Умножение без накопления

Умножитель может не выполнять функцию накопления. Если накопление не используется, результат может напрямую помещаться в регистр регистрового файла данных или в регистр аккумулятора. Регистр, в который помещается результат, должен быть 16- или 32-разрядным. Если 16-разрядный регистр, в который помещается результат, является младшей половиной 32-разрядного

регистра, используется MAC0; если он является старшей половиной – используется MAC1. Если результат помещается в 32-разрядный регистр, может использоваться любой умножитель-накопитель.

Если результат помещается в 16-разрядный регистр, то слово, извлекаемое из умножителя, зависит от типа входных данных:

- Если выполняется умножение дробных чисел или используется опция `IH`, извлекается и помещается в 16-разрядный регистр старшая половина результата (см. рис. 2-14).
- Если выполняется умножение целых чисел, извлекается и помещается в 16-разрядный регистр младшая половина результата. Извлекаемое таким способом 16-разрядное слово обеспечивает наибольшую информацию для выбранного типа данных (см рис. 2-15).

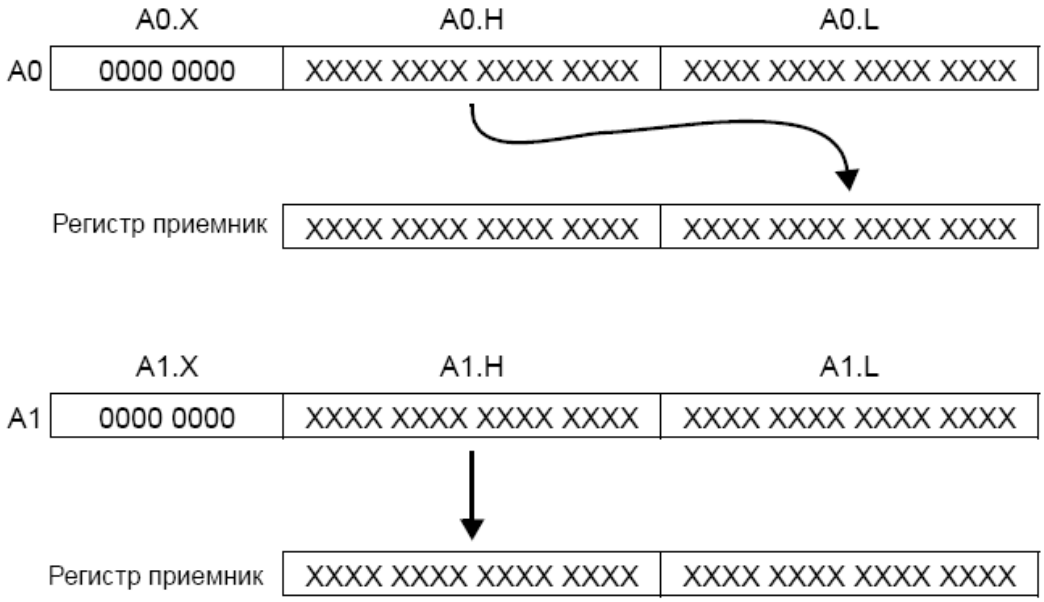

Рис. 2-14. Умножение дробных чисел.

Например, в данной команде используются дробные беззнаковые операнды:

```
R0.L = R1.L * R2.L (FU);
```

По этой команде старшие 16 разрядов произведения помещаются в младшую половину `R0`, выполняется насыщение и округление, используется MAC0.

В данной команде используются беззнаковые целочисленные операнды:

```
R0.H = R2.H * R3.H (IU);
```

По этой команде младшие16 разрядов произведения помещаются в старшую половину `R0`, выполняется любое необходимое насыщение, используется MAC1.

```
R0 = R1.L * R2.L;
```

Независимо от типов операндов 32 разрядов произведения помещаются в `R0`, выполняется насыщение, используется MAC0.

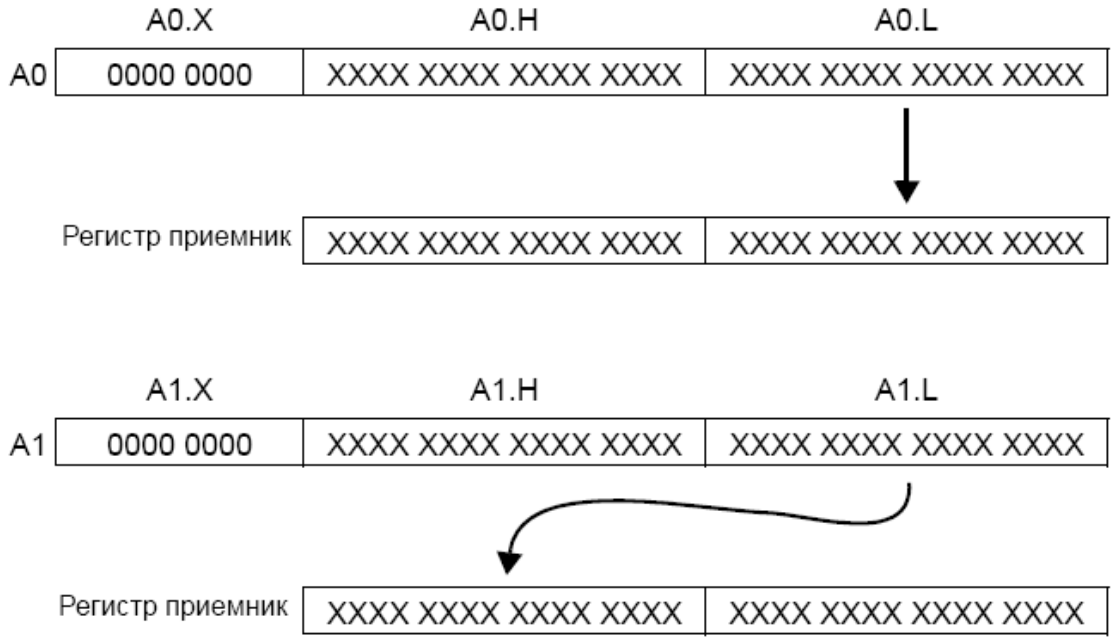

Рис. 2-15. Умножение целых чисел

Специальная команда умножения-накопления 32-разрядных целых чисел

Процессор поддерживает команду 32-разрядного умножения/накопления, выполняемую за несколько тактов:

```
Dreg *= Dreg
```

По этой команде выполняется умножение двух 32-разрядных целочисленных операндов и формируется 32-разрядный целочисленный результат, помещаемый в один из входных операндов.

Эта команда выполняется за несколько тактов. Дополнительную информацию о работе этой команды см. в спецификации продукта и *Справочном руководстве по набору команд процессора Blackfin ADSP-BF53x*. Выполнение этой макрофункции может прерываться; она не модифицирует данные регистров аккумуляторов.

Двойные операции умножения-накопления

Процессор имеет два 16-разрядных умножителя-накопителя. Для удвоения производительности в одной операции могут использоваться оба умножителя-накопителя. На вход каждого умножителя-накопителя поступают два одинаковых 32-разрядных регистра, что обеспечивает четыре возможных комбинации 16-разрядных входных операндов. Двойные операции умножителя-накопителя часто

называют векторными операциями, так как программа может размещать в четырёх входных операндах вектор отсчётов и выполнять векторные вычисления.

Пример команды двойного умножения-накопления:

```
A1 += R1.H * R2.L, A0 += R1.L * R2.H;
```

Эта команда представляет собой две операции умножения/накопления:

- в первой операции (выполняемой в MAC1) старшая половина `R1` умножается на младшую половину `R2` и складывается с содержимым аккумулятора `A1`;
- во второй операции (выполняемой в MAC0) младшая половина `R1` умножается на старшую половину `R2` и складывается с содержимым аккумулятора `A0`;

Результаты операций умножителя-накопителя могут записываться в регистры несколькими способами: как пара 16-разрядных половин регистров, как пара 32-разрядных регистров, как независимая 16-разрядная половина регистра или независимый 32-разрядный регистр.

Например:

```
R3.H = (A1 += R1.H * R2.L), R3.L = (A0 += R1.L * R2.L);
```

При выполнении этой команды содержимое 40-разрядного аккумулятора упаковывается в 16-разрядную половину регистра. Результат MAC1 должен быть передан в старшую половину регистра, результат MAC0 должен быть передан в младшую половину того же регистра.

Положение разрядов, извлекаемых из аккумулятора и помещаемых в 16-разрядный регистр, определяется типом операндов. См. раздел “Умножение без накопления”.

```
R3 = (A1 += R1.H * R2.L), R2 = (A0 += R1.L * R2.L);
```

При выполнении этой команды содержимое 40-разрядных аккумуляторов упаковывается в два 32-разрядных регистра. Регистры должны принадлежать одной паре (`R[1:0]`;`R[3:2]`;`R[5:4]`;`R[7:6]`).

```
R3.H = (A1 += R1.H * R2.L), A0 += R1.L * R2.L;
```

Эта команда является примером передачи в регистр содержимого только одного аккумулятора. Регистр, в который помещается результат, может быть 16- или 32-разрядным.

Устройство сдвига

Устройство сдвига выполняет функции поразрядного сдвига 16- или 32-разрядных входных данных, формируя 16-, 32- или 40-разрядные результаты. Эти функции включают арифметический сдвиг, логический сдвиг, циклический сдвиг,

различные функции проверки, установки, упаковывания и распаковывания битов, а также функцию определения порядка. Эти функции могут комбинироваться для реализации управления форматами чисел, включая представление чисел в формате с плавающей точкой.

Работа устройства сдвига

Команды устройства сдвига (>>>, >>, ASHIFT, LSHIFT, ROT) могут использоваться различными способами, в зависимости от требований используемой арифметики. Команды ASHIFT и >>> реализуют арифметический сдвиг. Команды LSHIFT, << и >> реализуют логический сдвиг.

Операции арифметического и логического сдвига можно разделить на подгруппы. В операциях с одним 16-разрядным числом или парой 16-разрядных чисел (применяемых во многих алгоритмах ЦОС), могут использоваться команды ASHIFT и LSHIFT. Эти команды обычно имеют три операнда.

В операциях с 32-разрядными значениями регистров, в которых используются два операнда, (например, команды, часто используемые компилятором) могут применяться команды >>> и >>.

Аргумент сдвига в командах арифметического, логического и циклического сдвига может поступать из регистра или указываться в виде непосредственного значения в теле команды. Подробную информацию о командах устройства сдвига см. в разделе “Обзор команд устройства сдвига”.

Операции сдвига с двумя операндами

Команды сдвига с двумя операндами выполняют сдвиг содержимого входного регистра и помещают результат в тот же регистр.

Непосредственный сдвиг

При операции непосредственного сдвига выполняется сдвиг входной битовой последовательности вправо (в сторону младших разрядов) или влево (в сторону старших разрядов) на заданное количество позиций. В команде непосредственного сдвига для управления величиной и направлением сдвига используется число, задаваемое в теле команды.

В следующем примере демонстрируется сдвиг входного значения в сторону младших разрядов.

```
R0 содержит 0000 B6A3;
R0 >>= 0x04;
```

Результат:

```
R0 содержит 0000 0B6A;
```

В следующем примере демонстрируется сдвиг входного значения в сторону старших разрядов.

```
R0 содержит 0000 B6A3;
R0 <<= 0x04;
```

Результат:

```
R0 содержит 000B 6A30;
```

Сдвиг, определяемый значением регистра

В этом типе операций для задания величины сдвига используется значение, содержащееся в регистре. Для получения величины сдвига используется целый 32-разрядный регистр; когда его абсолютное значение больше или равно 32, результат равен 0 или -1.

В следующем примере демонстрируется сдвиг входного значения в сторону старших разрядов.

```
R0 содержит 0000 B6A3;
R2 содержит 0000 0004;
R0 <<= R2;
```

Результат:

```
R0 содержит 000B 6A30;
```

Операции сдвига с тремя операндами

Команды сдвига с тремя операндами выполняют сдвиг входного регистра и помещение результата в регистр, отличный от входного.

Непосредственный сдвиг

В команде непосредственного сдвига для управления величиной и направлением сдвига используется число, задаваемое в теле команды.

В следующем примере демонстрируется сдвиг входного значения в сторону младших разрядов.

```
R0 содержит 0000 B6A3;
R1 = R0 >> 0x04;
```

Результат:

```
R1 содержит 0000 0B6A;
```

В следующем примере демонстрируется сдвиг входного значения в сторону старших разрядов.

```
R0.L содержит B6A3;
R1.H = R0.L << 0x04;
```

Результат:

```
R1.H содержит 6A30;
```

Сдвиг, определяемый значением регистра

В этом типе операций для задания величины сдвига используется значение, содержащееся в регистре. При использовании регистра для хранения величины сдвига (в командах `ASHIFT`, `LSHIFT` или `ROT`), она всегда содержится в младшей половине регистра (`Rn.L`). Младшие 6 разрядов `Rn.L` не маскируются и используются в качестве величины сдвига.

В следующем примере демонстрируется сдвиг входного значения в сторону старших разрядов.

```
R0 содержит 0000 B6A3;
R2.L содержит 0004;
R1 = R0 ASHIFT by R2.L;
```

Результат:

```
R1 содержит 000B 6A30;
```

В следующем примере демонстрируется циклический сдвиг входного значения. Предполагается, что бит кода условия (`CC`) установлен в значение 0. Дополнительную информацию о бите `CC` см. в разделе “Флаг кода условия” в главе 4.

```
R0 содержит ABCD EF12;
R2.L содержит 0004;
R1 = R0 ROT by R2.L;
```

Результат:

```
R1 содержит BCDE F125;
```

Заметьте, что бит `CC` включён в бит 3 результата.

Проверка, установка, сброс, инвертирование битов

Устройство сдвига обеспечивает метод проверки, установки, сброса и инвертирования определённых битов регистра данных. Все команды имеют два аргумента – регистр-источник и значение битового поля. Команда проверки не изменяет содержимое регистра. Результат команды проверки сохраняется в бите CC.

В следующих примерах демонстрируется применение описанных операций.

```
BITCLR (R0, 6);
BITSET (R2, 9);
BITTGL (R3, 2);
CC = BITTST (R3, 0);
```

Извлечение и внесение битового поля

При использовании устройства сдвига в любую область внутри 32-разрядного поля может вноситься битовое поле. Исходное поле может иметь длину от 1 до 16 битов. Кроме того, из любой части исходного 32-разрядного битового поля может извлекаться поле длиной от 1 до 16 битов.

В этих функциях в качестве аргументов используются два регистра. Один содержит 32-разрядный источник или приёмник. Второй содержит извлекаемое/вносимое поле, его длину и положение внутри исходного поля.

Обзор команд устройства сдвига

В таблице 2-11 перечислены команды устройства сдвига. Дополнительную информацию о синтаксисе языка ассемблер и влиянии команд устройства сдвига на флаги состояния см. в *Справочном руководстве по набору команд процессора Blackfin ADSP-BF53x*.

В таблице 2-11 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.
- “*” – флаг может быть установлен или сброшен в зависимости от результата выполнения команды.
- “*0” – версии команд, которые пересылают результат в аккумулятор A0, устанавливают или сбрасывают AV0.
- “*1” – версии команд, которые пересылают результат в аккумулятор A1, устанавливают или сбрасывают AV1.
- “**” – сброс флага.
- “***” – CC содержит значение последнего из задействованных в операции циклического сдвига битов.
- “–” – команда не влияет на флаги.

Таблица 2-11. Обзор команд устройства сдвига

Команда	Флаги статуса регистра ASTAT						
	AZ	AN	AC0 AC0_COPY AC1	AV0 AV0S	AV1 AV1S	CC	V V_COPY VS
BITCLR (Dreg, uimm5) ;	*	*	**	–	–	–	**/–
BITSET (Dreg, uimm5) ;	**	*	**	–	–	–	**/–
BITTGL (Dreg, uimm5) ;	*	*	**	–	–	–	**/–
CC = BITTST (Dreg, uimm5) ;	–	–	–	–	–	*	–
CC = !BITTST (Dreg, uimm5) ;	–	–	–	–	–	*	–
Dreg = DEPOSIT (Dreg, Dreg) ;	*	*	**	–	–	–	**/–
Dreg = EXTRACT (Dreg, Dreg) ;	*	*	**	–	–	–	**/–
BITMUX (Dreg, Dreg, A0) ;	–	–	–	–	–	–	–
Dreg_lo = ONES Dreg ;	–	–	–	–	–	–	–
Dreg = PACK (Dreg_lo_hi, Dreg_lo_hi) ;	–	–	–	–	–	–	–
Dreg >>>= uimm5 ;	*	*	–	–	–	–	**/–
Dreg >>= uimm5 ;	*	*	–	–	–	–	**/–
Dreg <<= uimm5 ;	*	*	–	–	–	–	**/–
Dreg = Dreg >>> uimm5 ;	*	*	–	–	–	–	**/–
Dreg = Dreg >> uimm5 ;	*	*	–	–	–	–	**/–
Dreg = Dreg << uimm5 ;	*	*	–	–	–	–	*
Dreg = Dreg >>> uimm4 (V) ;	*	*	–	–	–	–	**/–
Dreg = Dreg >> uimm4 (V) ;	*	*	–	–	–	–	**/–
Dreg = Dreg << uimm4 (V) ;	*	*	–	–	–	–	*
A*n* = A*n* >>> uimm5 ;	*	*	–	**0/ –	**1/ –	–	–
A*n* = A*n* >> uimm5 ;	*	*	–	**0/ –	**1/ –	–	–
A*n* = A*n* << uimm5 ;	*	*	–	*0	*1	–	–
Dreg_lo_hi = Dreg_lo_hi >>> uimm4 ;	*	*	–	–	–	–	**/–
Dreg_lo_hi = Dreg_lo_hi >> uimm4 ;	*	*	–	–	–	–	**/–
Dreg_lo_hi = Dreg_lo_hi << uimm4 ;	*	*	–	–	–	–	*
Dreg >>>= Dreg ;	*	*	–	–	–	–	**/–
Dreg >>= Dreg ;	*	*	–	–	–	–	**/–
Dreg <<= Dreg ;	*	*	–	–	–	–	**/–
Dreg = ASHIFT Dreg BY Dreg_lo ;	*	*	–	–	–	–	*

Таблица 2-11. Обзор команд устройства сдвига (продолжение)

Команда	Флаги статуса регистра ASTAT						
	AZ	AN	AC0 AC0_COPY AC1	AV0 AV0S	AV1 AV1S	CC	V V_COPY VS
Dreg = LSHIFT Dreg BY Dreg_lo ;	*	*	–	–	–	–	**/–
Dreg = ROT Dreg BY imm6 ;	–	–	–	–	–	***	–
Dreg = ASHIFT Dreg BY Dreg_lo (V) ;	*	*	–	–	–	–	*
Dreg = LSHIFT Dreg BY Dreg_lo (V) ;	*	*	–	–	–	–	**/–
Dreg_lo_hi = ASHIFT Dreg_lo_hi BY Dreg_lo ;	*	*	–	–	–	–	*
Dreg_lo_hi = LSHIFT Dreg_lo_hi BY Dreg_lo ;	*	*	–	–	–	–	**/–
A*n* = A*n* ASHIFT BY Dreg_lo ;	*	*	–	*0	*1	–	–
A*n* = A*n* ROT BY imm6 ;	–	–	–	–	–	***	–
Preg = Preg >> 1 ;	–	–	–	–	–	–	–
Preg = Preg >> 2 ;	–	–	–	–	–	–	–
Preg = Preg << 1 ;	–	–	–	–	–	–	–
Preg = Preg << 2 ;	–	–	–	–	–	–	–
Dreg = (Dreg + Dreg) << 1 ;	*	*	*	–	–	–	*
Dreg = (Dreg + Dreg) << 2 ;	*	*	*	–	–	–	*
Preg = (Preg + Preg) << 1;	–	–	–	–	–	–	–
Preg = (Preg + Preg) << 2;	–	–	–	–	–	–	–
Preg = Preg + (Preg << 1) ;	–	–	–	–	–	–	–
Preg = Preg + (Preg << 2) ;	–	–	–	–	–	–	–