МИКРОСХЕМЫ ИНТЕГРАЛЬНЫЕ
1967ВН028, 1967ВН044

Руководство по программированию
ТСКЯ.431281.002РП

© АО «ПКК Миландр» *Версия 2.4.0 от 31.07.2020*

Содержание

1 Введение 7
2 Архитектура процессора 13
2.1 Вычислительные модули 14
2.1.2 Арифметико-логическое устройство (ALU) 15
2.1.3 Коммуникационно-логическое устройство (CLU) 15
2.1.4 Умножитель с накоплением (умножитель) 16
2.1.5 Побитное сдвиговое устройство (сдвиговое устройство) 16
2.1.6 Целочисленное арифметико-логическое устройство (IALU) 16
2.1.7 Устройство управления потоком команд 17
2.2 Внутренняя память и внутренние шины 20
2.2.1 Внутренние шины 21
2.2.2 Внутренние пересылки 21
2.2.3 Обращение к данным 22
2.2.4 Квадрообращения 22
2.2.5 Начальная загрузка 22
2.3 Масштабируемость и работа в составе многопроцессорной системы 22
2.3.1 Эмуляция и поддержка тестирования 22
2.4 Синтаксис и структура командной строки 23
2.4.1 Соглашения о натации команд 24
2.4.2 Поддержка безусловного исполнения 25
2.4.3 Поддержка условного исполнения 25
2.4.4 Правила запараллеливания команд 25
2.4.5 Общие ограничения 33
2.5 Регистры вычислительных блоков 39
2.5.1 Регистры регистровых файлов 42
2.5.2 Числовые форматы 48
3 Арифметико-логическое устройство (ALU) 56
3.1 Операции ALU 58
3.1.1 Операции с константами для команд вычислительных модулей 59
3.1.2 Опции команд ALU 59
3.1.3 Статус выполнения ALU 63
3.1.4 ALU-предикаты 65
3.2 Статические флаги ALU 65
3.3 Примеры использования ALU 66
3.3.1 Пример параллельного сложения однобайтных данных 66
3.3.2 Пример продольного сложения однобайтных данных 67
3.3.3 Пример использования регистра побочных результатов (PR) 67
3.4 Сводка команд ALU 68
4 Коммуникационно-логическое устройство (CLU) 74
4.1 Операции CLU 76
4.1.1 Функция TMAX 77
4.1.2 Решётчатые функции 77
4.1.3 Функции сжатия частотной полосы сигнала 85
4.1.4 Функция вычисления кросс-корреляций 89
4.1.5 Функция сложить/сравнить/выбрать 96
4.1.6 Опции команд CLU 98
4.1.7 Статус выполнения CLU 99
4.2 Примеры использования CLU 99
4.3 Сводка команд CLU 101
5 Умножитель 102
5.1 Операции умножителя 104
5.1.1 Опции команд умножителя 108

5.1.2 Статус выполнения умножителя 114
5.1.3 Предикаты умножителя 116
5.1.4 Статические флаги умножителя 116
5.2 Примеры использования умножителя 117
5.3 Сводка команд умножителя 118
6 Сдвиговое устройство 121
6.1 Операции сдвигового устройства 122
6.1.2 Примеры использования сдвигового устройства 132
6.1.3 Сводка команд сдвигового устройства 132
7 Целочисленное АЛУ (IALU) 135
7.1 Операции IALU 138
7.1.1 Целочисленные (арифметические и логические) операции IALU 138
7.1.2 Статус выполнения IALU 140
7.1.3 IALU-предикаты 142
7.2 Сводка команда IALU 161
8 Устройство управления 164
8.1 Операции устройства управления 168
8.1.1 Отработка условных команд 172
8.1.2 Отработка переходов 175
8.1.3 Отработка циклов 177
8.1.4 Отработка прерываний 178
8.2 Операции конвейера команд 183
8.2.1 Буфер выравнивания команд (IAB) 187
8.2.2 Буфер целевых адресов перехода (BTB) 189
8.2.3 Воздействие условных переходов на конвейер 196
8.2.4 Роль зависимостей и ресурсов в работе конвейера 204
8.2.5 Воздействие прерываний на конвейер 213
8.2.6 Воздействие исключений на конвейер 220
8.2.7 Использование регистра CJMP в командах перехода 221
8.3 Примеры программирования устройства управления 225
8.4 Сводка команд устройства управления 227
9 Память и шины 230
9.1 Физическая структура блока памяти 232
9.1.1 Обращение к данным по чтению 234
9.1.2 Обращение к данным по записи 235
9.1.3 Арбитраж шин при обращении к памяти 235
9.2 Конвейер памяти 237
9.2.1 Обзор возможных потерь времени при обращении к памяти 240
9.2.2 Структура конвейера при доступе к памяти со стороны SOC-шины 242
9.3 Работа с памятью. Рекомендации по программированию 243
10 Набор команд 244
10.1 Команды ALU 244
10.1.1 Сложение/вычитание 244
10.1.2 Сложение/вычитание с переносом/займом 246
10.1.3 Сложение/вычитание с делением пополам 247
10.1.4 Абсолютное значение/ абсолютное значение суммы или разности 247
10.1.5 Дополнение 249
10.1.6 Максимум/минимум 249
10.1.7 Максимум/минимум по Витерби 251
10.1.8 Инкремент/декремент 253
10.1.9 Сравнение 254
10.1.10 Отсечение по границе 254
10.1.11 Продольная сумма 255

10.1.12 Подсчёт единиц....256
10.1.13 Загрузка/пересылка регистра PR (побочного результата) 257
10.1.14 Инкремент указателя однобитного FIFO 257
10.1.15 Абсолютное значение с побочным накоплением....258
10.1.16 Продольная сумма с побочным накоплением....259
10.1.17 Сложение/вычитание (двойная операция)....260
10.1.18 Передать 260
10.1.19 Логическое И/И-НЕ/ИЛИ/ИСКЛ-ИЛИ/НЕ 261
10.1.20 Расширение 261
10.1.21 Сжатие....264
10.1.22 Слияние....266
10.1.23 Переставить (однобайтные слова) 266
10.1.24 Переставить (короткие слова)....267
10.1.25 Сдвиг вправо на заданное число бит с упаковкой в меньший формат представления 268
10.1.26 Знаковое расширение с вычитанием....269
10.1.27 Отсечение по границе до положительных значений 270
10.1.28 Сложение/вычитание с делением пополам с округлением к бесконечности....270
10.1.29 Поэлементное векторное сравнение....271
10.2 Команды ALU плавающей арифметики....272
10.2.1 Сложение/Вычитание (плавающая арифметика)....272
10.2.2 Сложение/вычитание с делением пополам (плавающая арифметика)....273
10.2.3 Максимум/Минимум (плавающая арифметика) 274
10.2.4 Абсолютное значение суммы или разности (плавающая арифметика)....275
10.2.5 Смена знака (плавающая арифметика)....276
10.2.6 Сравнение (плавающая арифметика) 276
10.2.7 Преобразование из формата с плавающей точкой к формату с фиксированной точкой....277
10.2.8 Преобразование из формата с фиксированной точкой к формату с плавающей точкой 278
10.2.9 Преобразование из формата с плавающей точкой одинарной точности к формату с плавающей точкой повышенной точности (64 или 40 бит) 279
10.2.10 Преобразование из формата с плавающей точкой повышенной точности к формату с плавающей точкой одинарной точности 280
10.2.11 Отсечение по границе (плавающая арифметика)....280
10.2.12 Копирование знака (плавающая арифметика)....281
10.2.13 Масштабирование (плавающая арифметика)....282
10.2.14 Передать (плавающая арифметика)....283
10.2.15 Обратное приближение (плавающая арифметика) 283
10.2.16 Обратное приближение к квадратному корню (плавающая арифметика)....284
10.2.17 Мантисса (плавающая арифметика)....285
10.2.18 Логарифм (плавающая арифметика)....286
10.2.19 Сложение/Вычитание (двойная операция, плавающая арифметика)....287
10.2.20 Векторные команды ALU плавающей арифметики....287
10.3 Команды CLU....301
10.3.1 Решётчатый максимум (CLU)....301
10.3.2 Максимум (CLU)....302

10.3.3 Пересылка регистров результата (TR) и истории (TRH) решётчатых операций, а также коммуникационного управляющего регистра (CMCTL) 302
10.3.4 Сжатие частотной полосы сигнала с пересылкой регистра решётчатых операций (двойная операция, CLU) 303
10.3.5 Вычисление кросс-корреляций с пересылкой регистра решётчатых операций (двойная операция, CLU) 304
10.3.6 Сложить/сравнить/выбрать (CLU) 305
10.4 Команды умножителя 306
10.4.1 Умножение (обычных слов) 306
10.4.2 Умножение с накоплением (обычных слов) 308
10.4.3 Умножение с накоплением и пересылкой регистра MR (двойная операция, обычные слова) 310
10.4.4 Умножение (квадро, коротких слов) 314
10.4.5 Умножение с накоплением (квадро, коротких слов) 316
10.4.6 Умножение с накоплением и пересылкой регистра MR (двойная операция, квадро, короткие слова) 319
10.4.7 Комплексное умножение с накоплением (коротких слов) 322
10.4.8 Комплексное умножение с накоплением и пересылкой регистра MR (двойная операция, короткие слова) 324
10.4.9 Умножение (плавающая арифметика, обычных/расширенных слов) 328
10.4.10 Загрузка/Пересылка регистра результата умножения (MR) 329
10.4.11 Извлечение слов из регистра результата умножителя (MR) 333
10.4.12 Сжатие регистра результата умножителя (MR) 336
10.4.13 Векторные команды умножения для чисел плавающей запятой одинарной точности. 338
10.5 Команды сдвигового устройства 341
10.5.1 Арифметический/Логический сдвиг 341
10.5.2 Циклический сдвиг 343
10.5.3 Извлечение поля 344
10.5.4 Размещение поля 345
10.5.5 Поле/Битовая маска 346
10.5.6 Получить биты 347
10.5.7 Вставить биты 348
10.5.8 Проверить бит 349
10.5.9 Сбросить/Установить/Переключить бит 350
10.5.10 Выдать число ведущих нулей/единиц 351
10.5.11 Вычислить порядок 351
10.5.12 Блочная плавающая арифметика 352
10.5.13 Загрузка/Пересылка регистра временного однобитного FIFO переполнения (BFOTMP) 352
10.5.14 Загрузка/Пересылка статусных регистров вычислительных блоков (X/YSTAT) 353
10.5.15 Загрузка регистров CLU результата (TR) и истории (TRH) решётчатых операций, а также коммуникационного управляющего регистра (CMCTL) 354
10.6 Команды (целочисленные) IALU 355
10.6.1 Сложение/Вычитание (целочисленное) 356
10.6.2 Сложение/Вычитание с переносом/займом (целочисленное) 357
10.6.3 Сложение/Вычитание с делением пополам (целочисленное) 357
10.6.4 Сравнение (целочисленное) 358
10.6.5 Максимум/Минимум (целочисленный) 359
10.6.6 Абсолютное значение (целочисленное) 360
10.6.7 Логическое И/И-НЕ/ИЛИ/ИСКЛ-ИЛИ/НЕ (целочисленное) 360

10.6.8 Арифметический сдвиг/Логический сдвиг (целочисленный) 361
10.6.9 Циклический сдвиг влево/ Циклический сдвиг вправо (целочисленный).....362
10.6.10 Дополнительные арифметические команды IALU 362
10.7 Команды устройства управления (УУ).....376
10.7.1 Переход/Вызов.....377
10.7.2 Вычисляемый переход/вызов 378
10.7.3 Возврат (из прерывания).....378
10.7.4 Понизить статус (прерывания до подпрограммы).....379
10.7.5 Если - то (условное выполнение).....380
10.7.6 Если - иначе (условное ветвление и выполнение) 380
10.7.7 Загрузить значение предиката во флаг статических предикатов .381
10.7.8 Бездействие 381
10.7.9 Включить/Выключить обработку прерываний 381
10.7.10 Включить/Выключить BTB 382
10.7.11 Сделать записи в BTB недействительными.....382
10.7.12 Ловушка.....382
10.7.13 Ловушка эмулятора 383
10.7.14 Нет операции 383
Приложение А Краткий справочник 384
A.1 Краткий справочник по ALU 384
A.2 Краткий справочник по CLU 387
A.3 Краткий справочник по умножителю.....387
A.4 Краткий справочник по сдвиговому устройству 389
A.5 Краткий справочник по IALU 389
A.6 Краткий справочник по устройству управления.....391
Приложение Б Определения регистров и битов.....393
Приложение В Декодирование команд.....417
B.1 Структура команды.....417
B.2 Формат команды вычислительного блока.....418
B.3 Команды ALU 419
B.4 Команды CLU.....425
B.5 Команды умножителя.....426
B.6 Команды сдвигового устройства.....429
B.7 Команды IALU 433
B.8 Команды устройства управления 443
B.9 Коды предикатов 446
Приложение Г Глоссарий.....449
Лист регистрации изменений 455

1 Введение

Настоящий документ содержит сведения, необходимые для программирования систем на базе микросхем 1967ВН028, 1967ВН044 (далее процессор), представляющих собой высокопроизводительный 32-разрядный процессор.

Процессор совмещает работу нескольких вычислительных устройств, как с фиксированной, так и с плавающей точкой, допуская при этом использование слов самой различной длины. Процессор допускает построение на своей основе масштабируемых вычислительных систем. Процессор разработан в соответствии с идеологией т.н. "системы-на-кристалле" ("system-on-chip" – SOC) и имеет в своем составе 24 Мбит внутрикристальной оперативной статической памяти (SRAM), встроенные периферийные устройства ввода/вывода, интерфейс хост-процессора, контроллеры DMA (Direct Memory Access – прямого доступа к памяти), порты LVDS-линков (т.е. линков, использующих Low Voltage Differential Signal – низковольтный дифференциальный сигнал), а также доступ к общей шине для непосредственного взаимодействия нескольких процессоров.

Процессор является функциональным аналогом процессора ADSP-TS201 TigerSHARC. Обеспечивается совместимость на уровне двоичных кодов с некоторым возможным различием в быстродействии. Процессор состоит их двух архитектурных частей: ядра процессора (Рисунок 1), где исполняются команды, и периферийных устройств (Рисунок 2), где хранятся данные и, которые осуществляют операции обмена с внешними устройствами. В процессоре можно выделить следующие элементы:

- пару вычислительных модулей: X и Y, каждый из которых содержит умножитель, ALU, CLU, сдвиговое устройство и регистровый файл объемом в 32 слова;
- пару блоков целочисленных ALU: J и K, каждый из которых содержит 32-битное целочисленное ALU, а также регистровый файл объемом в 32 слова;
- устройство управления (Sequencer), управляющее ходом исполнения программы, содержащее буфер выравнивания команд (instruction alignment buffer - IAB) и буфер целевых адресов перехода (branch target buffer - BTB);
- три 128-битные шины, обеспечивающие возможность высокоскоростного обмена между внутренней памятью и другими компонентами ядра процессора (вычислительными блоками, блоками целочисленных ALU, устройством управления и SOC-интерфейсом);
- 128-битную шину, обеспечивающую возможность высокоскоростного обмена между внутренней памятью и периферийными устройствами внешнего ввода/вывода (DMA, внешним портом и портами линков);
- SOC-интерфейс, обеспечивающий связь между внутренними шинами ядра и шиной периферийных устройств;
- интерфейс внешнего порта, включая хост-интерфейс, контроллер SDRAM, конвейерный интерфейс со статической организацией конвейера, четыре канала DMA, четыре порта LVDS-линков (с двумя каналами DMA каждый) и поддержку многопроцессорной работы;
- 24 Мбит внутренней памяти, организованной как шесть блоков по 4 Мбит, каждый из которых содержит 128K 32-битных слов.
- средства поддержки отладки;
- тестовый порт JTAG.

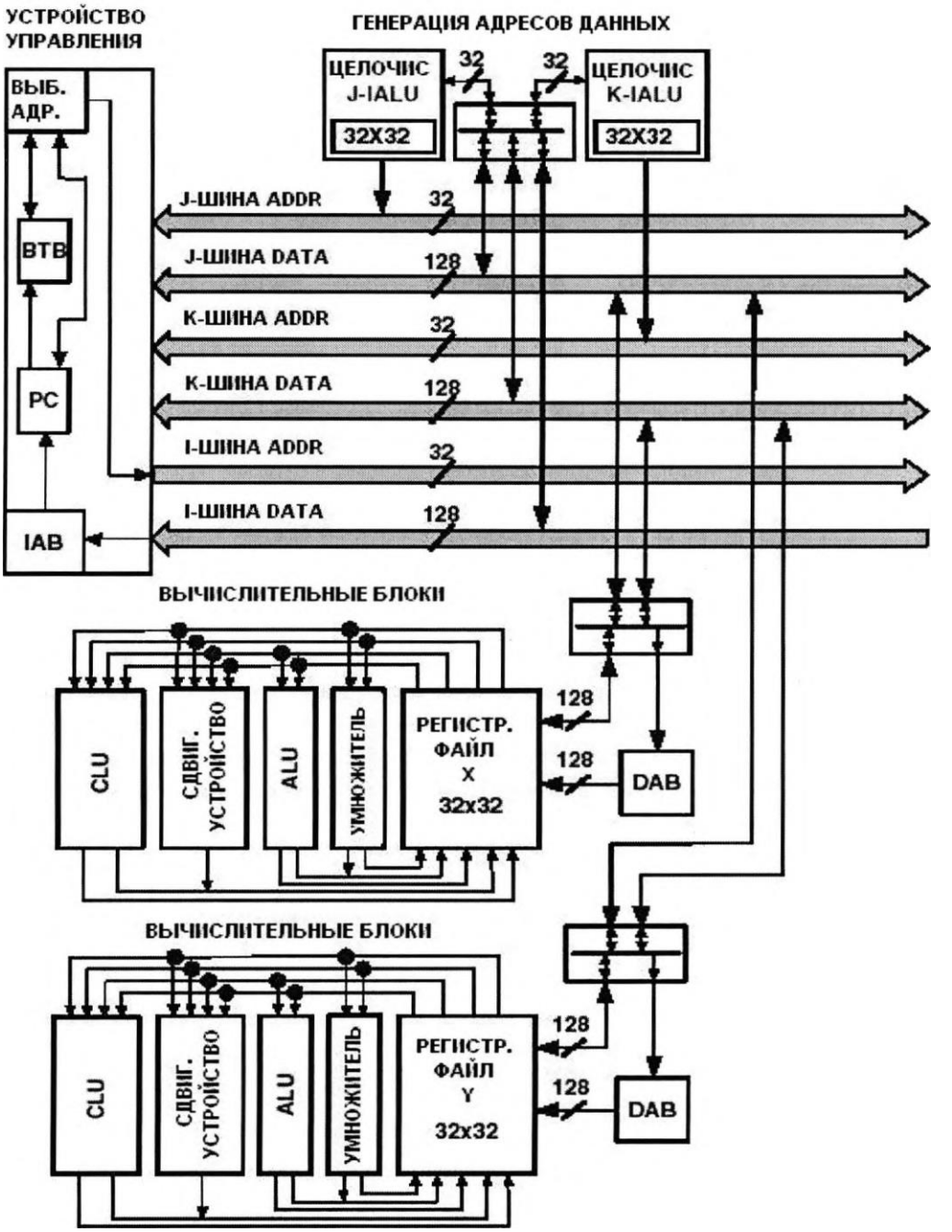


Рисунок 1 – Схема ядра процессора

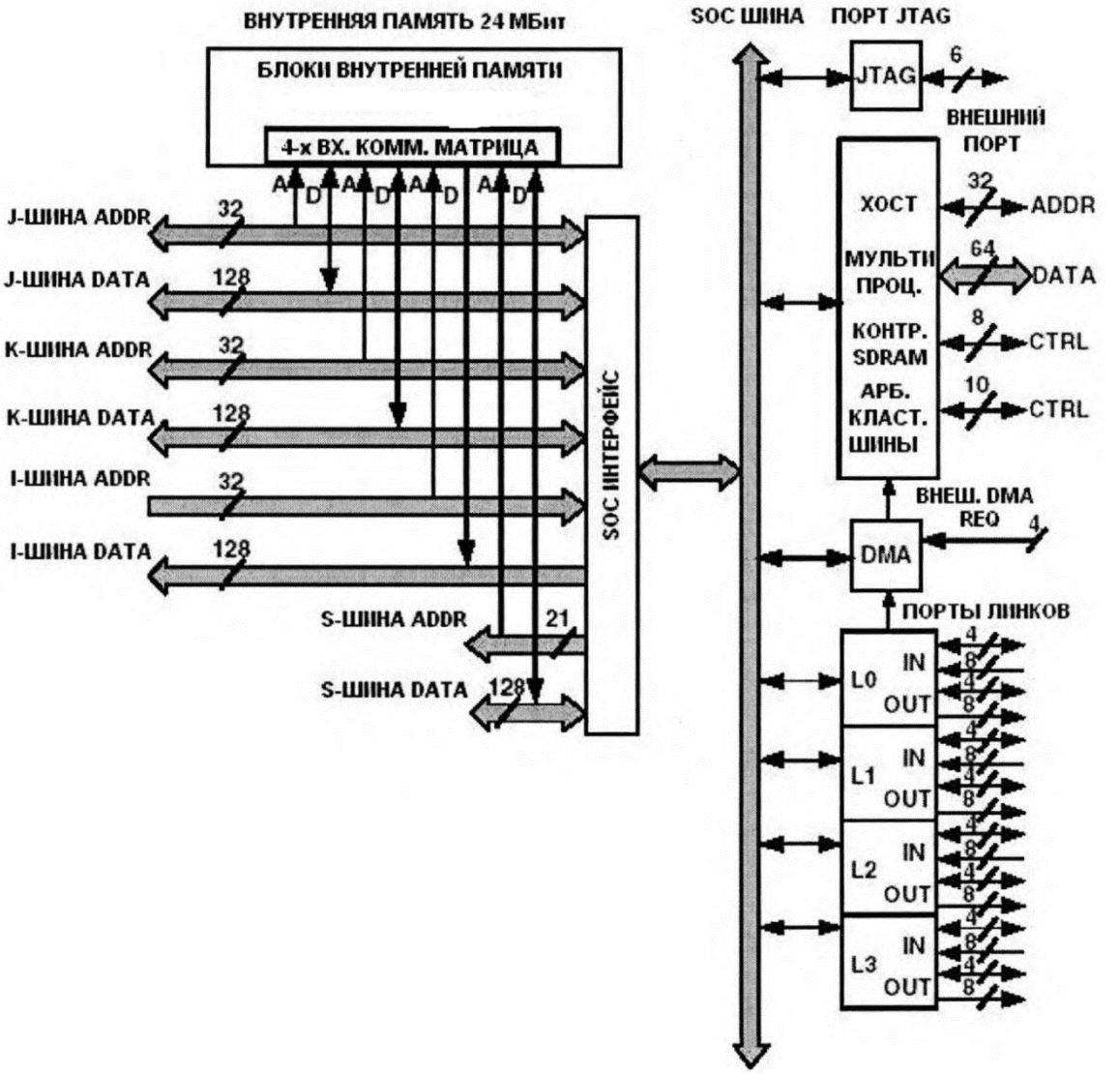


Рисунок 2 – Схема подключения периферийных устройств к ядру процессора

Внешний порт процессора обеспечивает интерфейс с внешней памятью, с устройствами ввода/вывода, отображенными в память, с хост-процессором и с дополнительными процессорами. Внешний порт выполняет арбитраж внешней шины и подает управляющие сигналы на общую глобальную память, SDRAM и устройства ввода/вывода.

Рисунок 2 показывает типовую однопроцессорную систему. Мультипроцессорная система показана на рисунке 4.

В процессоре имеется ряд особенностей, которые упрощают создание систем на его базе. Эти особенности заключаются в трех ключевых моментах:

- поддержка форматов с плавающей точкой по стандарту IEEE;
- средства последовательного сканирования и внутрикристальной эмуляции в соответствии со стандартом IEEE 1149.1 JTAG;
- архитектурные особенности, обеспечивающие поддержку языков высокого уровня и операционных систем.

Особенности процессора, которые непосредственно обеспечивают поддержку компиляторов языков высокого уровня и операционных систем, включают в себя:

- простые ортогональные команды, позволяющие компилятору эффективно использовать поля многокомандных строк;

- файлы регистров данных общего назначения и регистров целочисленных ALU;
- аппаратно поддерживаемые 32-, 64-битные (в соответствии со стандартом IEEE 754/854) и 40-битные типы данных с плавающей точкой, а также аппаратно поддерживаемые 8-, 16-, 32- и 64-битные типы данных с фиксированной точкой;
- большое адресное пространство;
- возможность непосредственной модификации адреса;
- легко поддерживаемая перемещаемость кода и данных;
- быстрое сохранение и восстановление регистров процессора в стеках во внутренней памяти.

Архитектура данного процессора ближе к архитектуре VLIW (Very Large Instruction Word), где исполняющее устройство (одно из нескольких) для команды назначается заранее (статически) – на этапе компиляции или на этапе написания программы (если программа на ассемблере). Высокая производительность процессора достигается отчасти за счет возможности исполнения до четырех 32-битных команд в одном такте. Команды не обязаны быть выровнены на какие- то границы в памяти, что позволяет не расходовать программную память впустую.

Внутренняя память объемом 24 Мбит разбита на шесть блоков памяти по 128К слов. Каждая из четырех пар внутренних шин «адрес/данные» подсоединена ко всем шести блокам памяти. Шесть блоков памяти поддерживают до четырех обращений в каждом такте, причем каждый блок памяти может выполнить 128-битное обращение за такт.

Кластерная шина, на которую выходит внешний порт, имеет ширину в 64 бита. Высокая пропускная способность ввода/вывода сочетается с высокой скоростью работы ядра. Для достижения высокой тактовой частоты процессор использует конвейерную внешнюю шину с программируемой глубиной конвейера для межпроцессорных обменов, а также для синхронной статической памяти (SSRAM) и для синхронной динамической памяти (SDRAM).

Высокую пропускную способность передачи данных из точки в точку поддерживают четыре порта LVDS-линков. Каждый линк-порт обеспечивает полнодуплексную связь.

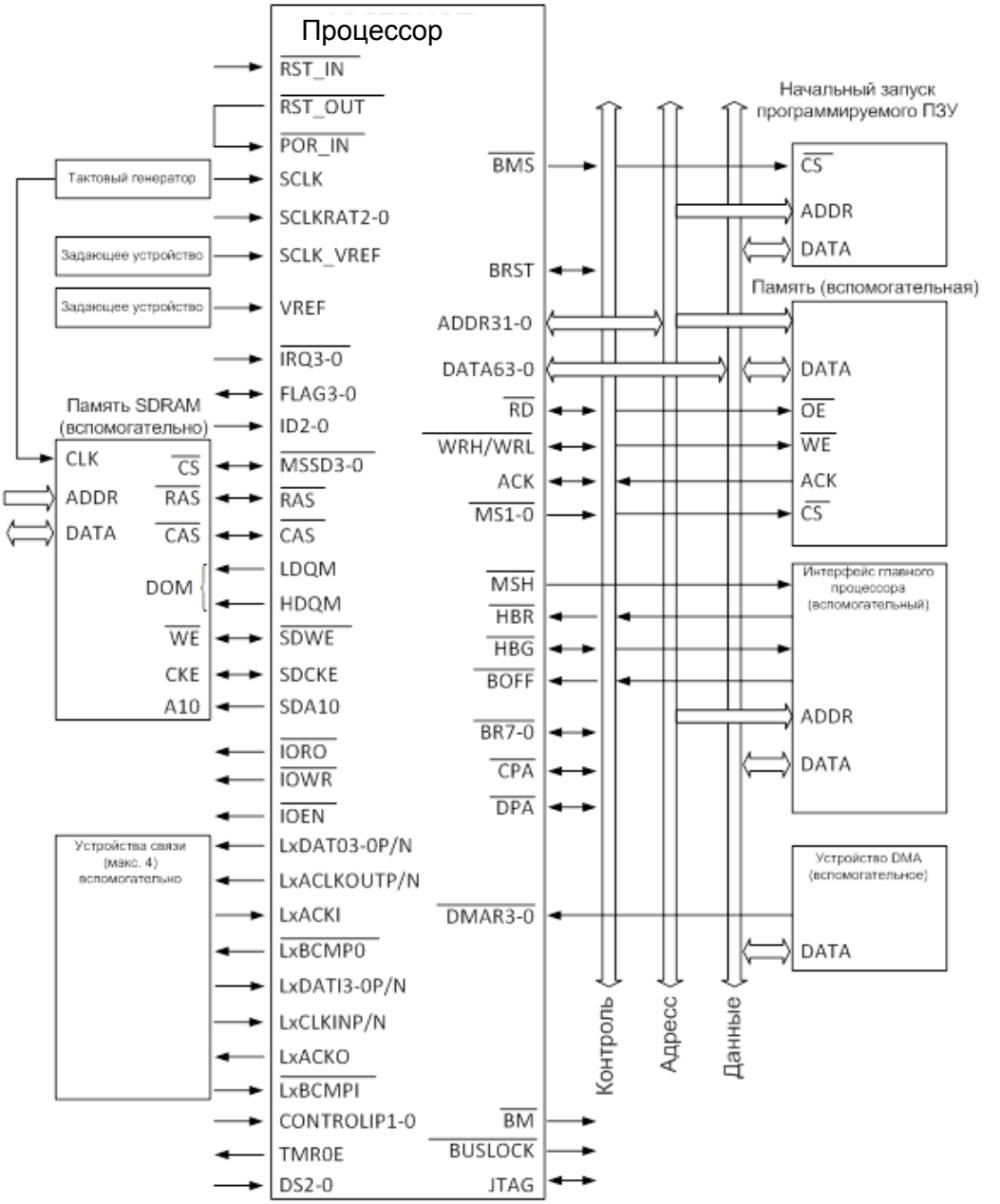


Рисунок 3 – Однопроцессорная конфигурация

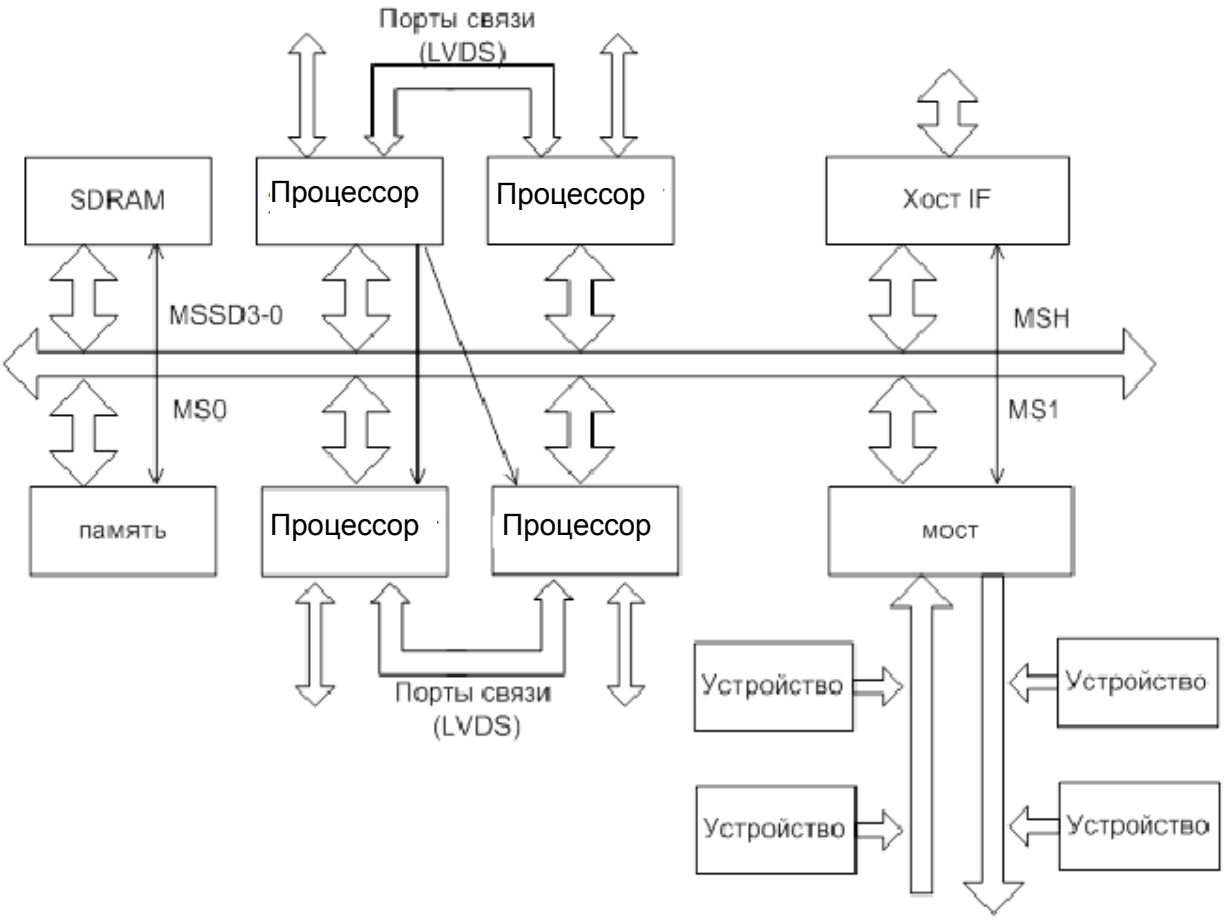


Рисунок 4 – Мультипроцессорная кластерная конфигурация

2 Архитектура процессора

Как видно из рисунка 1 в ядре процессора можно выделить пять самостоятельных функциональных модуля: X, Y, J, K, S. Модули X и У называют вычислительными модулями (устройствами), модули J и К называют целочисленными АЛУ, а модуль S есть устройство управления. Модули X и У получили свое название из-за того, что в них происходит основная обработка данных и они в свою очередь включают в себя различные устройства обработки данных. Модули J и К можно было бы назвать адресными генераторами (или адресными АЛУ), т.к. только они имеют возможность формирования адреса для доступа к памяти. Однако, кроме адресных функций эти модули способны выполнять и функции по обработке данных. При этом данные (как и адреса) могут иметь только тип Integer, что соответствует 32-разрядному целому числу. Отсюда и название – целочисленное АЛУ(IALU). Модуль S выполняет формирование адресов команд, управляет потоком команд, а также управляет работой всего конвейера ядра.

Модули X и У абсолютно симметричны, т.е. все что делает модуль X, может делать и модуль У. Каждый модуль обрабатывает свой поток команд. При этом имеется возможность рассматривать модули X и У как один объединенный модуль, который может управляться одной командой.

Модули J и К также симметричны, и каждый из них выполняет свой поток команд.

Устройство управления (модуль S) также имеет способность выполнять несколько потоков команд определенных типов. Среди них основными можно назвать команды переходов и вызовов.

Модульная архитектура процессора с независимыми потоками команд позволяет организовать высокопараллельные вычисления. Так в процессе интенсивных вычислений одно или оба целочисленных ALU вычисляют или генерируют адреса для выборки до двух операндов размером в квадрослово из двух блоков памяти, в то время как устройство управления одновременно извлекает следующую четверку команд из третьего блока памяти. Параллельно вычислительные устройства могут обрабатывать ранее считанные операнды, а устройство управления подготавливать переход.

Пока ядро процессора занято вышеописанными действиями, каналы DMA могут в фоновом режиме обновлять содержимое внутренней памяти квадрословами данных либо из внешнего порта, либо из портов линков.

Вычислительное ядро процессора достигает исключительно высокой производительности при цифровой обработке сигналов благодаря использованию следующих особенностей:

- вычислительный конвейер;
- пара вычислительных устройств;
- исполнение до четырех команд за такт;
- выборка/запись до восьми слов памяти за такт.

Два идентичных вычислительных устройства (X и У) выполняют арифметические операции как с плавающей, так и с фиксированной точкой. Эти устройства выполняют до 6-ти операций с плавающей точкой или до 24-х операций с фиксированной точкой за такт. Модули J и К выполняют SISD обработку, модули X и У могут выполнять SISD и SIMD обработку. При этом модули X и У могут управляться одной общей командой. Модули X и У имеют сложную структуру и включают в себя разнообразные вычислительные блоки.

2.1 Вычислительные модули

Ядро процессора содержит два вычислительных устройства, называемых вычислительными модулями. Каждый вычислительный модуль содержит регистровый файл и четыре независимых вычислительных блока: ALU, CLU, умножитель и сдвиговое устройство. Для того, чтобы можно было удовлетворить самые разнообразные запросы пользователя, вычислительные блоки умеют обрабатывать данные в нескольких форматах представления с фиксированной и плавающей точкой.

Эти форматы перечислены ниже и показаны на рисунке 5.

2.1.1.1 Форматы данных с фиксированной точкой

В число таких форматов входят: 64-битное длинное слово (Long Long), 32-битное обычное слово (Integer), 32-битное комплексное (16-битная действительная и 16-битная мнимая часть), 16-битное короткое слово (Short) и 8-битное однобайтное слово (Char). Для коротких слов арифметики с фиксированной точкой учетверенные операции над данными, выровненными на границу квадрослова, обеспечивают быструю обработку вектора данных. Байтовые операции поддерживаются также для данных, выровненных на границу октослова (т.е. 256-битного слова).

2.1.1.2 Форматы данных с плавающей точкой

В число таких форматов входят: 32-битное обычное слово (float), 64-битное двойное слово (double) и 40-битное расширенное слово. Операции с плавающей точкой выполняются с одинарной, двойной и расширенной точностью. Формат обычного слова с плавающей точкой соответствует стандартному IEEE-формату, а 40-битное число формата с увеличенной точностью размещается в двойном слове (64 бита), занимая дополнительно к 32 битам восемь наименее значащих битов (Least Significant Bits – LSBs) мантиссы для достижения большей точности.

Каждый вычислительный модуль имеет многопортовый регистровый файл, содержащий регистры общего назначения для обмена данными между вычислительными блоками и шинами данных, а также для хранения промежуточных результатов. Ко всем этим регистрам можно обращаться как к обычному, двойному или квадрорегистру. Все операции вычислительных модулей исполняются на двух стадиях конвейера. Результат операции доступен только на второй стадии, поэтому если следующая команда вычислительного модуля использует результат текущей команды как операнд-источник, то всегда возникает такт простоя процессора (пузырь).

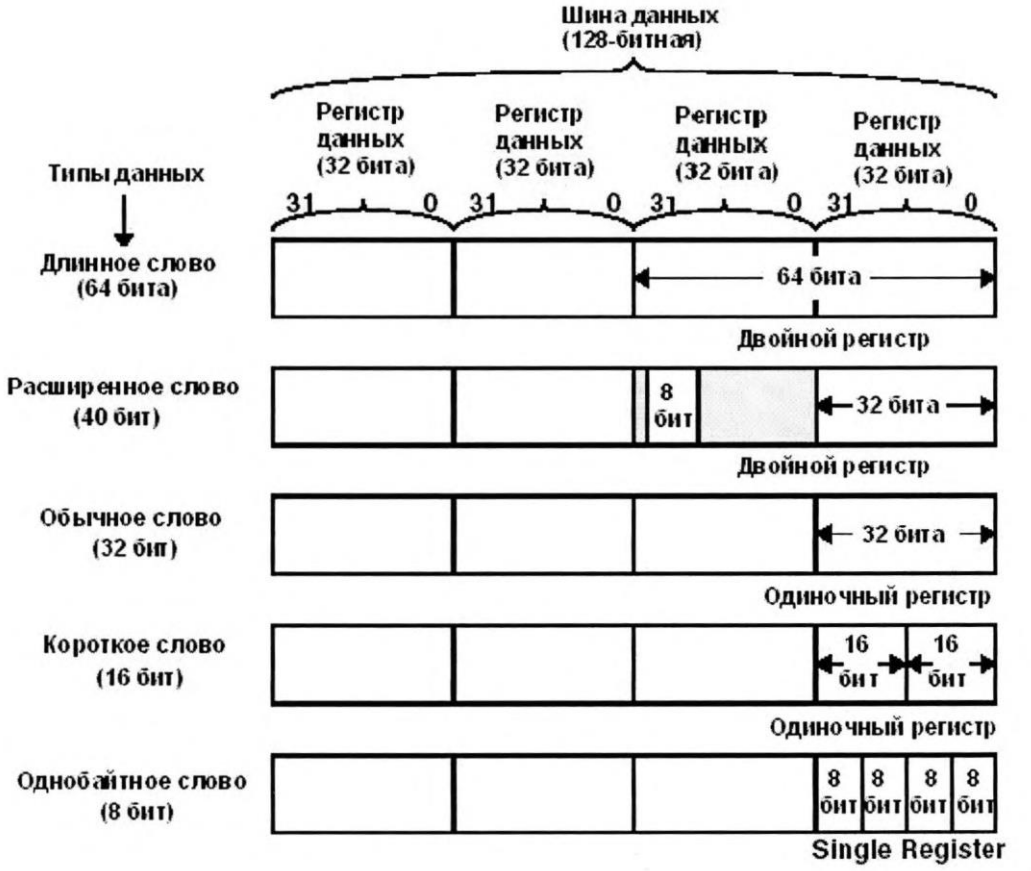


Рисунок 5 – Определения форматов слов

2.1.2 *Арифметико-логическое устройство (ALU)*

ALU выполняет арифметические операции над данными с фиксированной и с плавающей точкой, а также логические операции над данными с фиксированной точкой. Источником и приемником большинства операций ALU является регистровый файл вычислительного блока. Операнды ALU могут иметь ширину 32 или 64 бита, что соответствует одиночным операциям над типами Int, Float, Double, Long Long. Для типов меньшей разрядности (Short, Char) поддерживается векторная обработка. Очевидно, что количество элементов в векторах для типа Int равно двум, для типа Short – двум или четырем, а для типа Char – четырем или восьми. Учитывая, что вычислительные модули X и Y могут работать как один сдвоенный модуль, длина обрабатываемых векторов может быть увеличена в два раза, т.е. за такт процессор способен обрабатывать 4 типа Int, 8 типов Short или 16 типов Char. Все это соответствует максимальной ширине шины памяти в 128 бит.

2.1.3 *Коммуникационно-логическое устройство (CLU)*

Каждый вычислительный модуль содержит вычислительное устройство специального назначения, называемое коммуникационно-логическим устройством (CLU). Команды CLU предназначены для поддержки различных алгоритмов, используемых в коммуникационных приложениях, а именно алгоритмов:

- декодирования по Витерби;
- декодирования турбо кода;
- сжатия частотной полосы сигнала (despreading) для систем множественного доступа с кодовым разделением (CDMA);
- кросс-корреляций, используемых для отыскания пути.

Особенностью данного блока является то, что он имеет собственный 32-словный набор регистров общего назначения. Таким образом команды CLU могут использовать операнды из РОН вычислительного модуля и из РОН собственного блока.

2.1.4 *Умножитель с накоплением (умножитель)*

Блок умножителя выполняет умножение чисел с фиксированной или плавающей точкой, а также операции умножения с накоплением для чисел с фиксированной точкой. Умножитель поддерживает несколько типов данных для форматов с фиксированной и плавающей точкой. Для плавающей точки – это стандартные форматы одинарной и двойной точности, а также формат с расширенной точностью. Источником и приемником для большинства операций является регистровый файл вычислительного блока. Умножитель также поддерживает операции умножения над комплексными числами, представленными в виде пары 16-разрядных коротких слов, упакованных в одно 32-разрядное слово. Младшие значащие биты этого слова представляют собой действительную часть комплексного числа, а старшие значащие биты представляют мнимую часть. Команды умножения с накоплением используют в качестве приемника специальные регистры-аккумуляторы.

2.1.5 *Побитное сдвиговое устройство (сдвиговое устройство)*

Сдвиговое устройство выполняет логические и арифметические сдвиги, манипуляции с битами, заполнение полей и извлечение полей. Сдвиговое устройство выполняет операции с одним 64-битным, одним или двумя 32-битным, двумя или четырьмя 16-битными, а также с четырьмя или восемью 8-битными операндами с фиксированной точкой. Операции сдвигового устройства включают в себя:

- сдвиги и циклические сдвиги влево и вправо;
- операции манипуляции с битами, включая: установку бита, очистку бита, переключение бита и проверку;
- операции манипуляции с битовыми полями, включая извлечение полей, заполнение полей с использованием регистра BFOTMP (внутреннего регистра сдвигового устройства);
- операции с однобитным FIFO для поддержки потоков битов с полями переменной длины;
- поддержка операций преобразования фиксированной и плавающей точки (такими как извлечение порядка, определение числа ведущих единиц и нулей).

2.1.6 *Целочисленное арифметико-логическое устройство (IALU)*

Целочисленные ALU могут исполнять стандартные автономные операции ALU с регистровыми файлами IALU, операции загрузки/сохранения регистров и пересылки из регистра в регистр, а также формировать адреса при обмене данными между памятью и регистрами. Процессор имеет пару IALU (J-IALU и K-IALU), что позволяет формировать адреса одновременно для двух параллельно исполняемых транзакций по 128 бит. Наличие целочисленных ALU позволяет вычислительным операциям исполняться с максимальной эффективностью, поскольку вычислительные устройства могут быть заняты исключительно обработкой данных.

Каждое IALU имеет многопортовый 32-словный регистровый файл. Любая вычислительная операция IALU исполняется за один такт. При генерации адресов целочисленные ALU поддерживают пред-модификацию регистров без их обновления и постмодификацию с обновлением. Кольцевые буферы реализованы аппаратно. Целочисленные ALU поддерживают следующие типы команд:

- обычные команды IALU;
- команды пересылки данных [из регистра в регистр];
- команды загрузки константы в регистр;

- команды загрузки/сохранения данных с модификацией значением регистра;
- команды загрузки/сохранения данных с модификацией непосредственным значением.

При косвенной адресации (команды с модификацией) значение одного из регистров регистрового файла может быть модифицировано значением другого регистра файла или непосредственным 8- или 32-битным значением либо до (пред-модификация), либо после (постмодификация) выполнения обращения к памяти. При адресации кольцевых буферов каждому из четырех первых регистров IALU может быть сопоставлено значение длины для выполнения автоматической адресации по ее модулю внутри такого буфера; границами кольцевых буферов могут служить любые адреса памяти. Кольцевые буферы делают возможной эффективную реализацию линий задержки и других структур данных, используемых обычно в цифровых фильтрах и преобразованиях Фурье. Переход адресного указателя через границу кольцевого буфера обрабатывается процессором автоматически, что позволяет сократить накладные расходы и упростить программную реализацию алгоритмов.

Целочисленные ALU поддерживают также бит-реверсивную адресацию, ориентированную на алгоритм БПФ. Бит-реверсивная адресация реализуется с использованием сложения с обратным переносом, подобного обычному сложению, но у которого бит переноса берется из старших битов и направляется в младшие.

IALU обеспечивают гибкость в пересылках данных обычными, двойными или квадрословами. Каждая команда может исполняться с производительностью одна на такт. Обычно задержки, обусловленные зависимостью между командами IALU, отсутствуют, но если таковые имеются, то их длительность может достигать трех тактов. Задержки могут появляться, если текущая команда производит загрузку данных из памяти в регистр IALU или пересылку регистра в регистр IALU, а следующая команда использует загружаемый регистр в качестве операнда-источника. В таком случае возникает три такта задержки до готовности операнда. Необходимо отметить, что IALU обладает практически такими же вычислительными возможностями по обработке данных типа Integer как и вычислительный модуль. Наиболее проблемным местом является только отсутствие в IALU операции умножения.

2.1.7 *Устройство управления потоком команд*

Устройство управления предоставляет адреса для выборки команд из памяти. УУ совместно с IALU позволяет вычислительным операциям выполняться с максимальной эффективностью. Эффективность ветвления обеспечивается устройством управления за счет использования буфера целевых адресов перехода (BTB), позволяющего сократить задержки на исполнение условных и безусловных переходов. На УУ возлагается две задачи: декодирование извлеченных из памяти команд, т.е. выделение команд из полей командной строки, и отсылка их на соответствующие исполнительные устройства (вычислительные блоки, целочисленные ALU или УУ), а также управление ходом исполнения программы. Команды устройства управления подразделяются на два типа:

- команды управления ходом исполнения программы. Эти команды используются для смены текущего адреса исполнения (перехода) и для условного исполнения индивидуальных команд;
- команды непосредственного расширения. Эти команды используются для расширения числовых полей, указываемых в непосредственных операндах для УУ и IALU.

Команды управления ходом исполнения программы, в свою очередь, подразделяются на две категории:

- явные переходы и вызовы, основанные на непосредственно указываемом

в коде команды адресном операнде. К примеру, по команде 'if <cond> jump 100;' всегда осуществляется переход по адресу 100, если только вычисленное значение выражения <cond> есть «истина»;

- косвенные переходы, базирующиеся на адресе, предоставляемом регистром. (Команды, используемые для указания на условное исполнение командной строки, представляют собой подкатегорию косвенных переходов). К примеру, 'if <cond> cjmp;' является переходом по адресу, содержащемуся в регистре CJMP.

Команды управления ходом исполнения должны располагаться в первом поле командной строки. Непосредственными расширениями могут обладать команды IALU и УУ (в последнем случае только команды управления ходом исполнения).

Эти команды не задаются программистом явно, а порождаются автоматически в случае, если размер непосредственных данных, использованных в командах, достаточно велик. Программист должен поместить команду, которая требует непосредственного расширения, в первое поле командной строки и оставить свободное поле в этой строке (задействовав только три поля), с тем чтобы ассемблер мог поместить непосредственное расширение во второе поле командной строки. В одной командной строке может присутствовать только одно непосредственное расширение.

Процессор достигает своей высокой скорости исполнения за счет использования 9-ти стадий конвейера. Каждая стадия соответствует одному такту частоты процессора. Упрощенная структура конвейера приведена на рисунке 6.

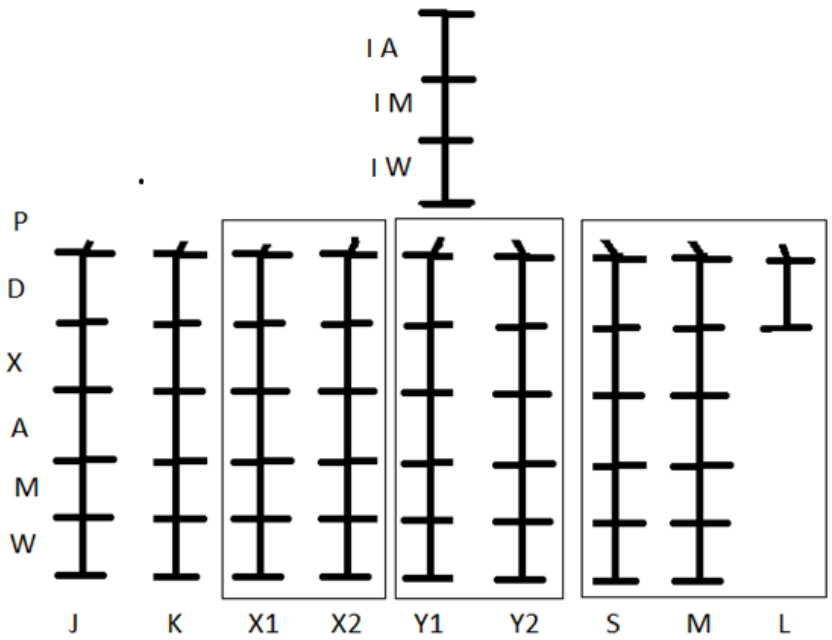


Рисунок 6 – Структура конвейера процессора

На рисунке 6 горизонтальным линиям конвейера соответствуют регистры для хранения информации между стадиями конвейера, а вертикальные линии соответствуют задержке обрабатывающей логики. Самая верхняя горизонтальная линия соответствует регистру IP – указателю на текущую команду устройства управления. IP содержит адрес, который отправляется в подсистему памяти для выборки команды.

- стадия адреса (стадия IA) соответствует логике декодирования адреса, а также логике арбитража доступа к памяти;
- следующая стадия IM соответствует чтению данных из накопителя в буфер памяти;

- на стадии IW происходит передача команды из буфера памяти в буфер выравнивания команд IAB. Шина команд имеет ширину 128бит, что соответствует выборке четырех команд за такт;
- на стадии P одна линия конвейера *чтения* команд разветвляется на девять линий (ветвей) конвейера *исполнения* команд.

Видно, что модули J и K имеют по одной линии конвейера, модули X и Y – по две линии конвейера каждый, а модуль управления включает три ветви S, M, L. Ветвь L короткая, т.к. она обрабатывает команды-расширители константного поля, и по сути вся обработка этой команды сводится к определению какой линии конвейера соответствует расширение и передачи данных на эту линию. Для модулей обработки J, K, X, Y последняя горизонтальная линия конвейера соответствует РОН, в который записывается результат операции. Название стадий конвейера обработки данных D, X, A, M, W выбрано исходя из логики выполнения команды загрузки из памяти модулей J и K.

- D – декодирование команды;
- X – исполнение команды (вычисление адреса доступа);
- A – передача адреса в подсистему памяти;
- M – чтение данных из накопителя в буфер;
- W – передача данных из буфера памяти в регистр ядра.

В зависимости от типа команды, операции, выполняемые на данных стадиях конвейера, будут различны. Так для модулей X и Y наиболее важными являются стадии A, M, W:

- на стадии A происходит декодирование команды;
- на стадии M первый такт исполнения;
- на стадии W второй (заключительный такт исполнения).

Стадии M и W для модулей X и Y еще называются стадиями исполнения и имеют дополнительное обозначение как EX1 и EX2.

Одной из задач УУ является отслеживание зависимостей по операндам во всех линиях конвейера и приостановка конвейера в случае, если операнды не готовы. Все линии конвейера продвигаются синхронно. При этом максимум 6 линий из 9-ти могут исполнять одновременно команды.

Для УУ основной линией конвейера является линия S, на которой происходит обработка команд переходов. Для такого глубоко конвейеризированного процессора скорость выполнения переходов является одним из ключевых факторов эффективности исполнения программы. Переходы могут быть безусловными и условными. Условные переходы могут быть двух типов: переходы, анализируемые на стадии X конвейера и на стадии W. Последние (на W) соответствуют анализу флагов операций в модулях X и Y. Можно видеть, что в случае выполнения перехода на стадии X процессор потеряет пять тактов, прежде чем следующая команда достигнет стадии X. Для ветвлений на стадии W таких тактов будет уже восемь. Для уменьшения потерь, связанных с переходами, УУ содержит буфер целевых адресов перехода (BTB) и имеет механизм статического прогнозирования переходов (выполняется пользователем). Если переход помечен как прогнозируемый, то он выполняется на стадии D линии S конвейера. Информация о переходе заносится в строку BTB. При первом выполнении данного перехода мы потеряем четыре такта, однако при последующих его выполнениях потери будут равны 0.

Также одной из задач УУ является обработка запросов прерываний. Процессор имеет четыре внешних линии запроса прерываний общего назначения (IRQ3-0), а также может генерировать прерывания от двух таймеров, каналов DMA, линк-портов, по арифметическим исключениям, а также генерировать векторные прерывания межпроцессорного взаимодействия и программные пользовательские прерывания. Есть возможность программным образом обеспечить обработку вложенных

прерываний. Прерывания имеют низкую латентность и не могут сбросить с конвейера текущие команды, уже начавшие свое исполнение. Прерывание векторизуется непосредственно по заданному пользователем адресу в регистровом файле таблицы векторов прерываний.

Запрос прерывания поступает в УУ и если прерывания разрешены, то стадия P конвейера прекращает поставлять команды на линии обработки команд, а УУ помещает в регистр IP адрес вектора прерывания (сбрасывая содержимое конвейера чтения команд) и запускает чтение нового потока.

2.1.7.1 Исполнение квадрокоманд

Процессор может исполнять до четырех команд за такт, выбираемых из одного блока памяти со скоростью 128 бит/такт. Способность исполнять несколько команд в единственном такте есть следствие архитектуры статической суперскалярности. Она не является суперскалярной архитектурой в строгом смысле поскольку то, какие команды исполняются в одном и том же такте, указывается в командной строке программистом или компилятором, т.е. программно, а не аппаратно самим кристаллом. Отсутствует также изменение порядка исполнения команд. Зависимость по регистрам проверяется, однако, аппаратно и, где это необходимо, генерируются такты приостановки. Код располагается в памяти вполне компактно, и какие-либо ограничения по выравниванию командных строк отсутствуют.

2.1.7.2 Относительные адреса для перемещаемости

Большинство команд перехода в процессоре поддерживает относительную адресацию (относительно значения программного счетчика PC), что позволяет легко создавать перемещаемый код. Кроме того, ссылки на данные в своем большинстве также относительны (относительно значений регистров), что позволяет программам обеспечивать доступ к блокам данных с использованием базового регистра.

2.1.7.3 Вложенные вызовы и прерывания

Адреса возврата из вложенных вызовов и прерываний (вместе с другими регистрами, если необходимо) сохраняются специальными командами в стеке во внутрикристальной памяти, что позволяет обеспечить большую общность при использовании языков высокого уровня. Для не вложенных вызовов и прерываний нет необходимости сохранять адрес возврата во внутренней памяти, что делает обращения к коротким невложенным подпрограммам более эффективными.

2.1.7.4 Переключение контекста

Процессор предоставляет возможность сохранения и восстановления командами загрузки/сохранения до восьми регистров за такт в стеке. Эта способность быстрого сохранения/восстановления делает возможной эффективную обработку прерываний и быстрое переключение контекста. Она также позволяет процессору обходиться без аппаратного стека или дублирующих регистров для регистровых файлов или регистров статуса.

2.2 Внутренняя память и внутренние шины

Внутрикристальная память состоит из шести блоков по 4 Мбит. Каждый блок содержит 128К слов, обеспечивая высокую пропускную способность, достаточную для работы обоих вычислительных блоков, выборки потока команд и внешнего ввода/вывода даже при очень высокой интенсивности. Процессор обеспечивает одновременные обращения к программе, двум операндам данных, а также системное обращение (со стороны SOC-шины), если это обращение к различным блокам памяти,

без ограничений по памяти или шинам. Блоки памяти могут хранить как команды, так и данные взаимозаменяемым образом.

Каждый блок памяти организован как 128К слов по 32 бита в каждом. Доступ конвейеризован, чтобы выдержать время доступа в один такт, которое требуется для ядра, DMA или внешней шины. При каждом обращении может быть считано/записано до четырех слов. Шесть блоков памяти представляет собой ресурс, который должен разделяться между вычислительными блоками, целочисленными ALU, устройством управления, внешним портом и портами линков. Если в течение некоторого такта более чем одно устройство пытается обратиться к одной и той же памяти, одному из устройств-претендентов доступ предоставляется, а другое придерживается с тем, чтобы предоставить ему участие в последующем арбитраже на следующем такте (см. раздел "Протокол арбитража шины" в Спецификации процессора). Такого рода конфликты оказывают лишь незначительное воздействие на производительность ввиду очень высокой пропускной способности внутренних шин.

Важное преимущество большой внутрикристальной памяти состоит в том, что, используя пересылку данных внутрь и наружу кристалла с помощью DMA, разработчик системы может реализовать высокую степень детерминизма по времени исполнения. Предсказуемое и детерминированное время исполнения представляет собой центральное требование в системах реального времени и цифровой обработки сигналов.

2.2.1 *Внутренние шины*

Ядро процессора имеет три шины (I-шина, J-шина и K-шина), каждая из которых связана со всеми блоками внутренней памяти через интерфейс коммутационной матрицы. Эти шины имеют ширину в 128 бит, что позволяет пересылать до четырех команд или четырех выровненных слов данных за такт по каждой шине. Внутрикристальные системные устройства используют для доступа к памяти SOC- и S-шины. К каждому блоку памяти на каждом такте может быть предоставлен доступ только одному устройству, так что если приложение обращается для разных целей (выборка команд, выполнение команд загрузить/сохранить целочисленных J-ALU и K-ALU, а также внешний доступ) к разным сегментам памяти, то все транзакции будут осуществляться без приостановки.

Отдельная шина «система-на-кристалле» (SOC-шина) связывает внешние интерфейсы (внешний порт, DMA, порты линков, порт JTAG и др.) с системой внутренней памяти через SOC-интерфейс и S-шину. SOC-шина состоит из 128-битной шины данных и 32-битной адресной шины. Она работает на половине частоты ядра процессора. Весь обмен данными между внутренней памятью или ядром и внешним портом проходит через эту шину.

2.2.2 *Внутренние пересылки*

Большинство регистров процессора считается универсальными регистрами (Ureg). Имеются команды пересылки данных между двумя Ureg-регистрами, между Ureg и памятью, а также команда непосредственной загрузки Ureg. В число универсальных регистров включены управляющие регистры, регистры статуса, а также регистры данных из регистровых файлов. Пересылки выполняются с теми же затратами времени, что и загрузка/сохранение для внутренней памяти. Доступ ко всем регистрам может быть осуществлен посредством команд пересылки из регистра в регистр или со стороны внешнего агента (хоста или другого процессора на той же кластерной шине). Однако, посредством команд загрузить/сохранить или команд непосредственной загрузки можно получить доступ только к регистрам ядра.

2.2.3 *Обращение к данным*

В каждой команде пересылки задается количество слов, к которому производится обращение в каждом блоке памяти. Обращение может производиться к двум блокам памяти на каждом такте средствами двух IALU генерирующих адреса. Размеры данных и регистров, а также синтаксис, определяющий их при обращении, рассматривается в разделе «Регистры регистровых файлов».

2.2.4 *Квадрообращения*

В командах задается количество слов (одно, два или четыре), которое должно быть загружено или сохранено. Квадрослова могут быть выровнены на границу квадрослова, а двойные слова – на границу двойного слова. Это, однако, необязательно при загрузке данных в вычислительные блоки, поскольку буфера выравнивания данных (Data Alignment Buffer – DAB) автоматически выравнивают квадрослова, которые не выровнены в памяти.

Из каждого блока памяти может быть выдано до четырех слов в каждый вычислительный блок, если, разумеется, новые данные не требуются на каждом такте. Такая пропускная способность данных оставляет резервные такты для ввода/вывода в блоки памяти. Использование такого доступа к данным дает выигрыш в приложениях с высокими требованиями по вводу/выводу, поскольку допускает ввод/вывод без ухудшения производительности ядра процессора.

2.2.5 *Начальная загрузка*

Внутренняя память процессора может быть загружена из 8-битного EPROM с использование механизма начальной загрузки при включении системы. Процессор может быть также загружен с использованием внешнего агента (master) или через один из портов линков Указание источника загрузки осуществляется через внешние контактные площадки. Информацию о загрузке процессора см. в Спецификации процессора.

2.3 Масштабируемость и работа в составе многопроцессорной системы

Процессор спроектирован для применения в составе многопроцессорных систем. В качестве основной многопроцессорной архитектуры поддерживается архитектура кластера, объединяющего до четырех процессоров на общей шине, которые совместно используют глобальную память, а также интерфейс к хост-процессору или к другим кластерам. Такие кластеры могут служить элементами большой многопроцессорной системы, образующей такую конфигурацию, как тороид, решетка, дерево, коммутационная матрица или что-то иное. Разработчик системы может использовать собственные средства межкластерной связи или задействовать внутрикристальные коммуникационные порты. Процессор обладает следующими возможностями поддержки работы в составе многопроцессорной системы:

- внутрикристальный арбитр шины для упрощения многопроцессорной работы;
- глобальная доступность внутренней памяти и регистров;
- семафоры;
- мощные внутрисхемные средства эмуляции многопроцессорной работы.

2.3.1 *Эмуляция и поддержка тестирования*

В состав процессора входит JTAG-порт для системного тестирования, поддерживающий стандарт IEEE 1149.1 Объединенной рабочей группы по автоматизации тестирования (Joint Test Action Group – JTAG). Этот стандарт определяет метод последовательного сканирования состояния ввода/вывода на контактах каждого

компонента в системе. Последовательный порт JTAG используется также внутрисхемным эмулятором для процессора для получения доступа к внутрикристальным процессорным средствам эмуляции

2.4 Синтаксис и структура командной строки

Процессор является статически суперскалярным процессором DSP-типа, который исполняет от одной до четырех команд, располагающихся в *командных полях командной строки.* За малыми исключениями команды исполняются со скоростью одна командная строка (четыре командных поля) за один такт ядра процессора (core clock – CCLK). На рисунке 7 показана структура командного поля и командной строки.

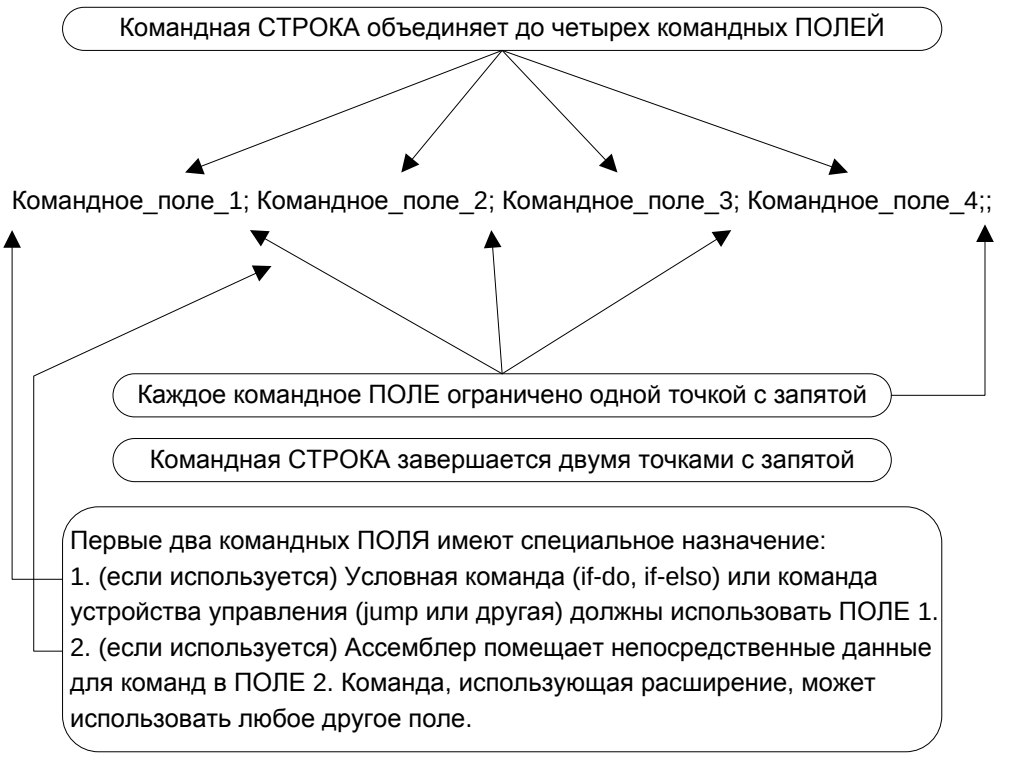


Рисунок 7 – Структура командного поля и командной строки

Есть несколько важных моментов, которые следует отметить относительно структуры командного поля командной строки, а также того, как эта структура связана с исполнением команд:

- каждая командная строка объединяет до четырех 32-битных командных полей;
- командные поля ограничены одной точкой с запятой;
- командные строки завершаются двумя точками с запятой;
- четыре команды из командной строки могут исполняться параллельно;
- для команд, которые используют 32-битные данные (к примеру, 32-битные явные адреса или 32-битные непосредственные данные) требуются два командных поля для исполнения размещения; одно поле для собственно команды и одно поле для непосредственного расширения;
- для команд устройства управления и команд непосредственного расширения требуются конкретные командные поля. Поле для непосредственного расширения отводится ассемблером; непосредственное расширение не является командой, явно написанной программистом.

Команда – это 32-битное слово, которое принуждает одно или несколько исполнительных устройств процессора провести операцию. Процессор исполняет или приостанавливает исполнение команд, расположенных в одной командной строке, одновременно. Независимо от размера (от одной до четырех команд) командные строки не обязаны быть выровнены по границам квадрослов и могут следовать в памяти одна за другой: новая командная строка начинается в следующем слове за словом, которым заканчивается предыдущая командная строка. Конец командной строки идентифицируется установленным старшим битом командного слова.

2.4.1 *Соглашения о натации команд*

Язык ассемблера процессора основывается на алгебраическом синтаксисе, что облегчает кодирование и читаемость. Синтаксис команд процессора выделяет операцию, которую исполняет процессор и опцию (т.е. режим), с которой он ее исполняет. Операции включают в себя вычисления, пересылки данных, управление потоком исполнения. Опции включают в себя выбор между режимом Одна-Команда-Одни-Данные (SISD) и режимом Одна-Команда-Множественные-Данные (SIMD), выбор формата данных, выбор размера слова, включение насыщения и включение усечения. Все элементы управления исполнением команды включены в синтаксис команды процессора – нет никаких битов режима, которые следует устанавливать в управляющих регистрах этого процессора.

В данном руководстве команды представляются в обобщенном формате. В этом формате показаны все элементы, из которых должен быть сделан выбор, и все необязательные элементы, возможные в данной команде. Соглашения для них таковы:

- X | Y | Z – список элементов, разграниченных вертикальной чертой «|», указывает на то, что синтаксис допускает выбор одного из элементов. Один элемент из списка обязан быть выбран. Вертикальная черта не является составной частью синтаксиса команды;
- {опция} – элемент или список элементов, заключенных в фигурные скобки, указывает на то, что элемент(ы) необязателен. Элемент может быть включен, а может быть и опущен. Фигурные скобки не являются составной частью синтаксиса команды;
- () [] , ; ;; – круглые и квадратные скобки, запятая, точка с запятой, двойная точка с запятой и другие символы являются обязательными элементами синтаксиса команды и должны появляться там, где они показаны в обобщенном синтаксисе за одним исключением. Пустые скобки (не выбраны никакие опции) не могут появиться в команде.

2.4.1.1 Rm Rmd Rmq

Имена регистров являются заменяемыми элементами в обобщенном синтаксисе и отмечаются курсивом. Имена регистров указывают, что синтаксис требует обычного *(Rm),* двойного *(Rmd)* или квадро- *(Rmq)* регистра. Дополнительную информацию о правилах именования регистров, выборе вычислительного блока и формате данных см. в разделе «Регистры регистровых файлов».

2.4.1.2 <Imm#>

Непосредственные данные (литеральные значения) в обобщенном синтаксисе появляются как <Imm#> с числовым значением вместо символа «#» указывающим на ширину значения в битах.

К примеру, следующая команда в обобщенном синтаксисе:

{X|Y|XY} {S|B}Rs = MIN|MAX (Rm, Rn) {({U}{Z})} ;

Может кодироваться как одна из следующих команд:

XR3 = MIN (R2, R1) ;

YBR2 = MAX (Rl, R0) (UZ) ; XYSR2 = MAX (R3, R4) (U);

2.4.2 *Поддержка безусловного исполнения*

Процессор поддерживает безусловное исполнение до четырех команд параллельно. Эта поддержка позволяет программисту сочетать одновременные вычисления с пересылками данных, переходами и циклами. Эти операции могут комбинироваться с рядом ограничений. Следующий пример кода показывает три командных строки, содержащие два, четыре и одно командное поле соответственно:

XR3:0=Q[J5+=J9]; YR1:0=R3:2+R1:0;;

XR3:0=Q[J5+=J9]; YR3:0=Q[K5+=K9]; XYR7:6=R3:2+R1:0; XYR8=R4*R5;; J5=J9-J10;;

Важно отметить, что приведенные команды исполняются безусловно. Их исполнение не зависит от вычисляемых условий.

2.4.3 *Поддержка условного исполнения*

Все команды могут исполняться условно (механизм, известный также как предикативное исполнение). Поле предиката присутствует в первом командном поле командной строки, и все остальные команды в этой строке либо исполняются, либо нет, в зависимости от вычисленного значения предиката. В условной вычислительной команде исполнение всей командной строки может зависеть от предиката, заданного в начале командной строки. Условные команды принимают одну из следующих форм:

IF Условие;
DO, Команда; DO, Команда; DO, Команда;;
IF Условие, Команда_устройства_управления;
ELSE, Команда; ELSE, Команда; ELSE, Команда;;

Такой синтаксис позволяет подчинить одному предикату до трёх условных команд. Дополнительную информацию см. в разделе «Отработка условных команд».

2.4.4 *Правила запараллеливания команд*

Процессор исполняет от одной до четырех 32-битных команд на строку. Компилятор или программист определяют, какие команды могут исполняться параллельно, т.е. в одной и той же строке до начала выполнения программы (что и приводит к названию статическая суперскалярность). Архитектура процессора накладывает некоторые ограничения на использование ряда команд и различных комбинаций команд.

Заметьте, что все ограничения относятся к комбинации команд внутри одной и той же строки. Ограничений на комбинации строк нет. Есть, однако, ситуации, в которых определенные комбинации строк могут вызывать такты приостановки главным образом из-за конфликта по данным (операнд команды в строке n+*1* есть результат команды в строке #*n*, который еще не готов к моменту своей выборки).

Таблица 1 и таблица 2 описывают правила запараллеливания команд для процессора. В следующих разделах содержатся более подробные сведения о каждом типе ограничений, которые иллюстрируются примерами:

- Общие ограничения;
- Ограничения на команды вычислительного блока;
- Ограничения на команды IALU;
- Ограничения на команды устройства управления.

Правила запараллеливания команд в таблицах 1 и 2 представляют собой ограничения по использованию ресурсов для команд, занимающих поля в одной и той же командной строке. По горизонтали перечислены *ресурсы* – архитектурные

компоненты процессора, которые активны при исполнении команды, а также количество доступных (однотипных) ресурсов. По вертикали перечислены *типы команд* – описательные названия классов команд. Для ресурсов “1” означает, что какая-то команда задействует одну единицу ресурса, а “2” означает, что команда задействует две единицы ресурса. Наряду с описательными названиями для большинства классов перечислены также типичные команды.

Важно отметить, что таблицы 1 и 2 описывают статические ограничения для процессора. Статические ограничения отличаются от динамических ограничений тем, что проверка выполнения статических ограничений может быть произведена ассемблером. К примеру, ассемблер помечает команду XR3:0=Q[J0 += 3];; как ошибочную, поскольку модификатор не является кратным четырем – это статическое нарушение.

Проверка динамических ограничений не может быть произведена ассемблером, поскольку эти ограничения формулируются в терминах времени исполнения, как это имеет место, например, с некорректными указателями. При обнаружении ядром попытки нарушения динамического (времени исполнения) ограничения, будет возбуждено исключение. Что бы ни случилось, процессор не попадет в состояние взаимоблокировки (deadlock’a), хотя в регистры или в память могут оказаться записанными непредсказуемые результаты.

В качестве примера динамического ограничения исследуем команду XR3: 0 = Q [J0+= 4];;. Хотя эта команда кажется ассемблеру корректной, она может нарушить аппаратное ограничение в случае, если значение J0 окажется не кратным четырем. Ассемблер не может предсказать, как код будет обходиться с J0 до момента исполнения этой команды, поэтому, если такое нарушение произойдет во время исполнения, оно будет динамическим нарушением.

Далее, таблицы 1 и 2 охватывают ограничения, обусловленные взаимодействием команд, входящих в одну и ту же командную строку, но, как правило, в них не включены ограничения, накладываемые на одиночную команду. Примером ограничений первого рода могут служить две команды в командной строке, пытающиеся использовать одно и то же устройство. Примером ограничения, касающегося индивидуальной команды, является запрет на использование регистра, недопустимого для данной команды. Команда XR0 = CB[J5+=1];; запрещена, так как доступ к кольцевым буферам может осуществляться только с помощью регистровых целочисленных ALU от J0 до J3 и от K0 до K3.

Для большинства типов команд в соответствующих строках таблиц 1 или 2 приведены команды и ресурсы, которые она задействует. Использование ресурсов для команд пересылки данных анализировать сложнее. Использование ресурсов для этих команд вычисляется путем сложения ресурсов источника, ресурсов приемника и базовых ресурсов, где базовые ресурсы определяются типом команды пересылки. Командами пересылки являются пересылки *Ureg*-регистров (регистр в регистр), непосредственная загрузка (непосредственные значения в регистр), загрузка из памяти (память в регистр) и сохранение в памяти (регистр в память). Понятие ресурса источника определяются ресурсным регистром и применимо только тогда, когда источник сам по себе является регистром (пересылка Ureg-регистров и сохранение).

Ресурсы приемника могут быть двух типов:

- постмодифицируемый адресный указатель (например, XR0 = [J0 += 2];);
- регистр-приемник – только в случае, когда приемником является регистр (пересылка *Ureg*-регистров, загрузка из памяти и непосредственная загрузка).

Если какая-то комбинация базовых ресурсов, ресурсов источника и приемника превышает имеющиеся в наличии, то комбинация является недопустимой.

Рассмотрим, например, следующую команду: **XR3:0 = Q[K31+0x40000];;**

Это команда загрузки из памяти, или, точнее, команда загрузки с использованием регистров K-IALU и 32-битного смещения. Рассматривая таблицу, находим, что базовые ресурсы, задействованные командой, составляют два поля в командной строке – команда K-IALU и второе поле строки (для непосредственного расширения, поскольку для кодирования смещения 0x40000 требуется больше, чем 8 бит). Приемником служит XR3:0, т.е. регистр вычислительного блока X. В строке 'X-регистровый файл, Dreg = XR31-0' таблицы 1 под подзаголовком *Ureg*-пересылка и загрузка (приемник-регистр)' указано, что команда использует также порт вычислительного блока X и входной порт блока X.

Следующая команда Ureg-пересылки предоставляет другой пример:

XYR0 = CJMP;;

В этом примере задействованы следующие ресурсы:

- одно поле команды;
- базовый ресурс – команда IALU (неважно, будь то J-IALU или K-IALU) и ресурс *Ureg*-пересылки (базовый) для команды IALU;
- ресурс источника – порт ввода/вывода устройства управления;
- ресурс приемника – порт вычислительного блока X и входной порт блока X, а также порт вычислительного блока Y и входной порт блока Y;

Для сравнения, команда **R3:0 = j7:4 ;;** использует поле команды, ресурс команды IALU (неважно, J или K), порты вычислительных блоков X и Y, а также входные порты блоков X и Y и, наконец, выходной порт J-IALU.

Таблица 1 – Правила запараллеливания для команд регистрового файла, DAB, J/K-IALU и команд доступа к порту

	Ресурсы																	
	Исп. полей команд	1-ое поле команды[1]	2-ое поле команды[2]	IALU: команды	J-IALU	K-IALU	J-IALU: порт В/В	K-IALU: порт В/В	X: порты В/В[3]	X: входные порты	X: выходные порт	X: DAB	Y: порты В/В[3]	Y: входные порты	Y: выходные порт	Y: DAB	Устр. упр.: порт В/В	SOC-интерфейс
⇒ Доступно ресурсов ⇒	4	1	1	2	1	1	1	1	2	2	1	1	2	2	1	1	1	1
⇓ Типы команд ⇓																		
IALU-арифметика																		
J-IALU *Js = Jm Op Jn|Imm8*	1			1	1													
J-IALU, 32-бит непосред. *Js = Jm Op Imm32*	2		1	1	1													
K-IALU *Ks = Km Op Kn|Imm8*	1			1		1												
K-IALU, 32-битн.непосред. *Ks = Km Op Imm32*	2		1	1		1												
Пересылки данных (полный ресурс = команда. + Ureg'и)																		
Пересылка Ureg[4] *Ureg = Ureg*	1			1														
Непоср.загрузка (полный ресурс = команда. + Ureg)																		
Непоср. 15-битн.загрузка *Ureg = Imm15*	1			1														
Непоср. 32- битн.загрузка *Ureg = Imm32*	2		1	1														

	Ресурсы																	
	Исп. полей команд	1-ое поле команды[1]	2-ое поле команды[2]	IALU: команды	J-IALU	K-IALU	J-IALU: порт В/В	K-IALU: порт В/В	X: порты В/В[3]	X: входные порты	X: выходные порт	X: DAB	Y: порты В/В[3]	Y: входные порты	Y: выходные порт	Y: DAB	Устр. упр.: порт В/В	SOC-интерфейс
⇒ Доступно ресурсов ⇒	4	1	1	2	1	1	1	1	2	2	1	1	2	2	1	1	1	1
⇓ Типы команд ⇓																		
Загрузка из памяти (полный ресурс = команда. + Ureg)																		
J-IALU загрузка[5] *Ureg = [Jm +|+= Jn|imm8]*	1		1	1	1													
J-IALU загрузка[5], 32-битн. смещение *Ureg = [Jm +|+= imm32]*	2		1	1	1													
K-IALU загрузка[5] *Ureg = [Km +|+= Kn|imm8]*	1			1		1												
K-IALU загрузка[5], 32-битн. смещение *Ureg = [Km +|+= imm32]*	2		1	1		1												
Сохранение в памяти (полный ресурс = команда. + Ureg)																		
J-IALU сохранение *[Jm +|+= Jn|imm8] = Ureg*	1			1	1													
J-IALU сохранение, 32-битн. смещение *[Jm +|+= imm32] = Ureg*	2		1	1	1													
K-IALU сохранение *[Km +|+= Kn|imm8] = Ureg*	1			1		1												
K-IALU сохранение, 32-битн. смещение *[Km +|+= imm32] = Ureg*	2		1	1		1												

	Ресурсы																	
	Исп. полей команд	1-ое поле команды[1]	2-ое поле команды[2]	IALU: команды	J-IALU	K-IALU	J-IALU: порт B/B	K-IALU: порт B/B	X: порты B/B[3]	X: входные порты	X: выходные порт	X: DAB	Y: порты B/B[3]	Y: входные порты	Y: выходные порт	Y: DAB	Устр. упр.: порт B/B	SOC-интерфейс
⇒ Доступно ресурсов ⇒	4	1	1	2	1	1	1	1	2	2	1	1	2	2	1	1	1	1
⇓ Типы команд ⇓																		
Пересылка и сохранение Ureg. Ресурсы источника (регистр-источник)																		
J-IALU *Ureg* = J30-0|JB3-0|JL3-0							1											
K-IALU *Ureg* = K30-0|KB3-0|KL3-0								1										
X: регистровый файл *Dreg* = XR31-0									1		1							
Y: регистровый файл *Dreg* = XR31-0													1		1			
XY: регистровые файлы (SIMD) *Ureg* = XYR31-0									1		1		1		1			
Устройство управления *Ureg* = CJMP|RETI|RETS|…[6]																	1	
SOC-интерфейс[5] *Ureg* = *SOC Ureg*[7]																		1

	Ресурсы																	
	Исп. полей команд	1-ое поле команды[1]	2-ое поле команды[2]	IALU: команды	J-IALU	K-IALU	J-IALU: порт В/В	K-IALU: порт В/В	X: порты В/В[3]	X: входные порты	X: выходные порт	X: DAB	Y: порты В/В[3]	Y: входные порты	Y: выходные порт	Y: DAB	Устр. упр.: порт В/В	SOC-интерфейс
⇒ Доступно ресурсов ⇒	4	1	1	2	1	1	1	1	2	2	1	1	2	2	1	1	1	1
⇓ Типы команд ⇓																		
Пересылка и загрузка Ureg. Ресурсы приемника (регистр-приемник)																		
J-IALU *Ureg* = J30-0|JB3-0|JL3-0							1											
K-IALU *Ureg* = K30-0|KB3-0|KL3-0								1										
X: регистровый файл *Dreg* = XR31-0									1	1								
Y: регистровый файл *Dreg* = XR31-0													1	1				
XY: регистровые файлы (SIMD) *Ureg* = XYR31-0									1	1			1	1				
Устройство управления *Ureg* = CJMP|RETI|RETS|…[6]																	1	
SOC-интерфейс[5] *Ureg* = *SOC Ureg*[7]																		1
Загрузка Ureg из памяти Ресурсы приемника (регистр-приемник)																		
X: регистровый файл DAB/SDAB X*Dreg* = DAB q[addr] X*Dreg* = XR31-0									1	1		1						
Y: регистровый файл DAB/SDAB Y*Dreg* = DAB q[addr] Y*Dreg* = YR31-0													1	1		1		
XY: регистровые файлы, DAB/SDAB XY*Dreg* = DAB q[addr] XY*Dreg* = XYR31-0									1	1		1	1	1		1		

Если в командной строке присутствует условная команда, то она должна использовать 1-е поле команды.

Если в командной строке присутствует непосредственное расширение, то оно должно использовать 2-е поле команды.

Две команды пересылки регистров *(Ureg=Ureg)* могут появиться в одной командной строке, если только пересылаются *Ureg*-регистры ядра. Если происходит пересылка *Ureg*-регистра, принадлежащего SOC, то в командной строке она может быть только единственной.

Команды загрузки и пересылки, у которых регистры-приемники лежат в одном квадро-регистре (например, XR0 и XR2 оба лежат в квадро-регистре XR3:0), могут появиться в одной командной строке, если только у них источники – внутренние. Регистры-приемники загрузки/пересылки не могут появиться в одной командной строке, если соответствующие источники – внешние (т.е. внешняя память или *Ureg*-регистры SOC).

Таблица 2 – Правила запараллеливания для команд устройства управления

	Ресурсы												
	Исп. полей команд	1-ое поле команды[1]	2-ое поле команды[2]	X: команда выч.бл-а	X: ALU	X: CLU	X: умножитель	X: сдвиговое устр.	Y: команда выч.бл-а	Y ALU	Y CLU	Y умножитель	Y: сдвиговое устр.
⇒ Доступно ресурсов ⇒	4	1	1	2	1	1	1	1	2	1	1	1	1
⇓ Типы команд ⇓													
Команды устройства управления													
Условный Jump/Call, 15-битн.непоср. IF *cond*, JUMP|CALL *Imm15*	1	1											
Условный Jump/Call, 32-битн.непоср. IF *cond*, JUMP|CALL *Imm32*	2	1	1										
Др. условные команды, косвенные переходы, операции со статич.флагами	1	1											
X: операции вычислительного блока													
X: команда ALU, кроме квадро-вывода *XDreg = Dreg + Dreg*	1			1	1								
X: команды[3] CLU	1			1		1		*					
X: команда умножителя, кроме квадро-вывода *XDreg = Dreg * Dreg*	1			1			1						
X: команда сдвигового устр., кроме MASK, FDEP, XSTAT[3]	1			1		*		1					
X: команда ALU, с квадро-выводом (сложение/вычитание, EXPAND, MERGE)	1			1	1		*						
X: команда умножителя, с квадро-выводом[4]	1			1	*		1						
X: команды сдвигового устройства: MASK, FDEP, XSTAT3	1			2									

	Ресурсы												
	Исп. полей команд	1-ое поле команды [1]	2-ое поле команды [2]	X: команда выч.бл-а	X: ALU	X: CLU	X: умножитель	X: сдвиговое устр.	Y: команда выч.бл-а	Y ALU	Y CLU	Y умножитель	Y: сдвиговое устр.
=> Доступно ресурсов =>	4	1	1	2	1	1	1	1	2	1	1	1	1
Типы команд													
X: операции вычислительного блока													
Y: команда ALU, кроме квадро-вывода Y*Dreg = Dreg + Dreg*	1								1	1			
Y: команды[3] CLU	1								1		1		*
Y: команда умножителя, кроме квадро-вывода Y*Dreg = Dreg * Dreg*	1								1			1	
Y*:* команда сдвигового устр., кроме MASK, FDEP, YSTAT[3]	1								1		*		1
Y: команда ALU, с квадро-выводом (сложение/вычитание, EXPAND, MERGE)	1								1	1		*	1
Y: команда умножителя, с квадро-выводом[4]	1								1	*		1	
Y: команда сдвигового устройства: MASK, FDEP, YSTAT[3]	1								2		*		
X и Y: операции вычислительных блоков (SIMD)													
XY: команда ALU, кроме квадро-вывода XY*Dreg = Dreg + Dreg*	1			1	1				1	1			
XY: команда[3] CLU	1			1		1			1		1		*
XY: команда умножителя, кроме квадро-вывода XY *Dreg = Dreg * Dreg*	1			1			1		1			1	
XY: команда сдвигового устр., кроме MASK, FDEP, XYSTAT[3]	1			1				1	1		*		1
XY: команда ALU, с квадро-выводом (сложение/вычитание, EXPAND ,MERGE)	1			1	1			1	1	1		*	1
XY: команда умножителя, с квадро- выводом[4]	1			1			1	1	1	*		1	
XY: команда сдвигового устройства: MASK, FDEP, XYSTAT[3]	1			2					2		*		

2.4.5 *Общие ограничения*

Существует общее ограничение, которое распространяется на все типы команд: две команды не могут осуществлять запись в один и тот же регистр. Это ограничение проверяется статически ассемблером.

Например: XR0 = R1 + R2 ; XR0 = R5 * R6 ;;

/* Недопустимо. Эти команды не могут быть в одной и той же командной строке*/;

Как следствие, регистром-приемником команды загрузки не может быть тот же регистр, который обновляется в той же командной строке другой командой.

Например: XR0 = [J17 + 1] ; R0 = R3 * R8 ;;

/* Недопустимо */

Команда загрузки/сохранения, которая использует адресацию с постмодификацией и обновлением, не может осуществлять загрузку в тот же регистр, что используется как индекс Jm/Km (указатель в память).

Например: J0 = [J0 += 1] ;;

/* Недопустимо; J0 не может быть использован и как приемник (Js), и как индекс (Jm) с постмодификацией (+=) для команды загрузки/сохранения */

Никакая команда не может осуществлять запись в регистр cjmp в той же командной строке, что и команда call (которая также обновляет регистр cjmp).

Например: if ALE, CALL label ; J6 = J0 + J1 (CJMP) ;;

/* Недопустимо */

Существует два способа обновления счетчика цикла, но комбинирование их недопустимо.

Например: IF LC0E; DO ... ; LC0 = [J0 + J1] ;;

/* Недопустимо */

2.4.5.1 Ограничения на команды вычислительного блока

Существует два вычислительных блока, и команды могут быть выданы одному из них или обоим:

- команды в формате XRs = *Rm* op *Rn* выдаются вычислительному блоку X;
- команды в формате YRs = *Rm* op *Rn* выдаются вычислительному блоку Y;
- команды в формате *Rs = Rm* op *Rn* или XY*Rs = Rm* op *Rn* выдаются обоим вычислительным блокам X и Y.

На рисунке 8 показаны потоки данных входных и выходных операндов для вычислительных блоков. Глядя на входные операнды (стрелки на рисунке 8, входящие в ALU, CLU, умножитель и сдвиговое устройство), можно заключить, что вычислительные потоки данных накладывают следующие ограничения на параллелизм вычислительных команд:

- только две вычислительные команды могут быть выданы вычислительному блоку в командной строке;
- вычислительному устройству может быть выдана только одна команда в командной строке.

Глядя на выходные операнды (стрелки на рисунке 8, исходящие из ALU, CLU, умножителя и сдвигового устройства), можно заключить, что вычислительные потоки данных накладывают особые ограничения на параллелизм вычислительных команд: в каждый момент времени только одна из команд с квадро-выводом (ALU, CLU и умножителя) может быть исполнена.

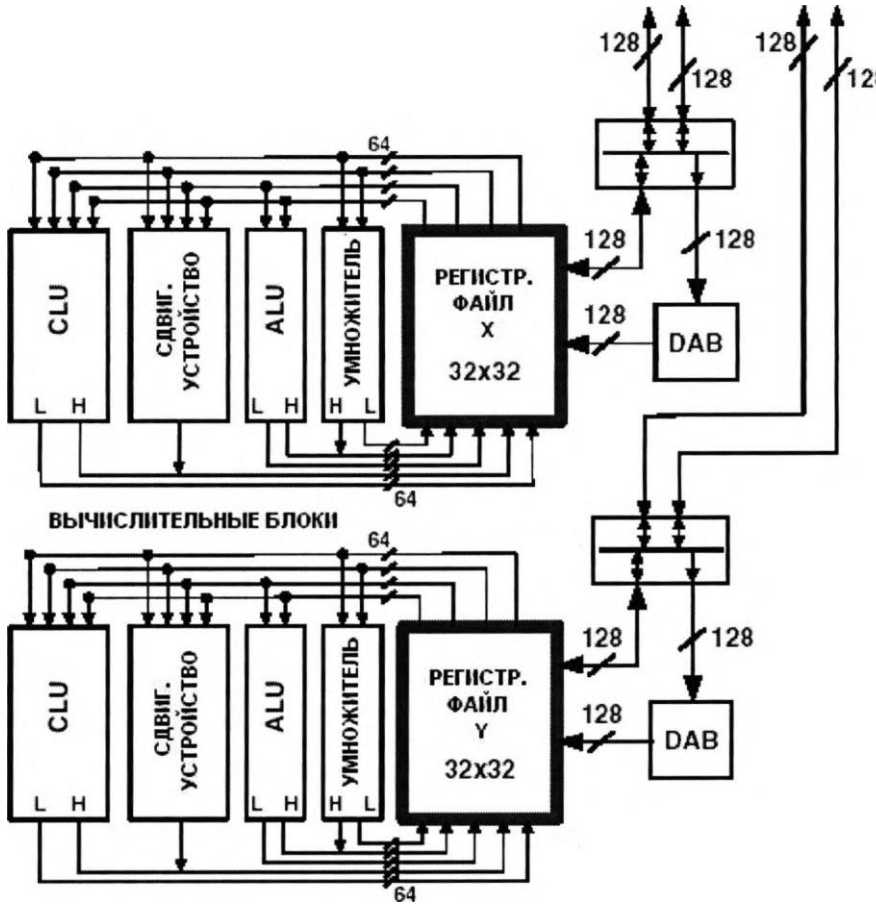


Рисунок 8 – Регистровые файлы данных в вычислительных блоках X и Y

Ниже приводятся примеры, следующие из вышеперечисленных условий на выдачу команд вычислительным блокам. Ассемблер статически проверяет все эти ограничения на потоки данных входных и выходных операндов.

Каждому вычислительному блоку может быть выдано до двух команд, из чего следует, что в одной строке может быть выдано максимум четыре команды вычислительных блоков. В рамках этого правила команды типа Rs = Rm op Rn считаются как одна команда на каждый вычислительный блок.

Например: R0 = R1 + R2 ; R3 = R4 * R5 ;;

/* Допустимо; всего четыре команды */

XR0 = R1 +R2; XR3 = R4 * R5; XR6 = LSHIFT R1 BY R7 ;;

/* Недопустимо; три команды на вычислительный блок */

В одном такте каждому устройству (ALU, CLU, умножителю или сдвиговому устройству) может быть выдана только одна команда, поэтому разные команды должны выдаваться разным устройствам.

Например: XR0= R1+ R2; XR6= R1 + R2 ;; /**/

XR0= R1+ R2; YR6= R1 + R2 /*Допустимо*/

Если исполняется одна из перечисленных ниже команд сдвигового устройства, то она должна быть единственной командой в командной строке для данного вычислительного блока. Это команды fdep, mask, getbits, putbits и команды доступа к регистрам xstat/ystat.

Например: XR0 += MASK R1 BY R2 ; XR6 = R1 + R2 ;;
/* Недопустимо; 3-х операндная команда сдвигового устройства в одной командной строке с операцией ALU; обе выдаются вычислительному блоку X */

Только одно вычислительное устройство (ALU, CLU или умножитель) может использовать две шины результатов. Устройство использует две шины результатов либо в случае, когда результат является квадрословом, либо в случае, когда имеется два результата (команды парного сложения/вычитания- R0 = R1 + R2, R5 = R1 - R2;). В командной строке можно поместить еще одну команду.

Например: R0 = R1 + R2, R5 = R1 - R2 ; XR6 = R1 * R2 ;; /* Допустимо */
R0 = R1 + R2, R5 = R1 - R2 ; XR11:8 = MR3:0 ;;
/* Недопустимо; две команды использует по две шины */

Однако, вычислительные операции CLU, использующие две шины результатов, и команда сдвигового устройства не могут быть размещены в одной командной строке.

Например: R3:0 = THR3:0; R6 = LSHIFT R1 BY R7;;
/* Недопустимо; CLU и сдвиговое устройство претендуют на одну и ту же шину результата */

Одновременно с командой сдвигового устройства загрузки/сохранения регистра x/ystat не может выдаваться никакая другая команда вычислительного блока.

В умножителе опция CR (сбросить и установить бит округления) и опция I (целое, а не дробное) не могут быть использованы в одной команде умножения с накоплением.

Опция CR умножителя не может быть использована только в двух следующих командах:

MR3:2|MR1:0+|-= Rm * Rn 32-битное дробное умножение с накоплением;

MR3:2|MR1:0 += Rm ** Rn Комплексное умножение с накоплением.

Два типа команд, связанных с CLU, исполняются на устройствах, отличных от CLU. Команды загрузки регистров решетки (tr), истории решетки (thr) и управляющего коммуникационного регистра (cmctl) исполняются на вычислительном сдвиговом устройстве, а команды permute исполняются на вычислительном устройстве ALU.

2.4.5.2 Ограничения на команды IALU

Имеется четыре типа команд IALU:

- загрузка из памяти, сохранение в памяти, например: R0 = [J0 + 1];
- операции IALU, например: J0 =J1 + J2;
- загрузка данных, например: R1 = 0xABCD;
- пересылка Ureg-регистров, например: XR0 = YR0;

При выдаче команд IALU действуют ограничения, описанные далее.

В одной командной строке может быть выдано до одной команды J-IALU и до одной команды K-IALU. Например:

R0 = [J0 + = 1] ; R1 = [K0 += 1] ;; /* Рекомендуется, чтобы J0 и K0 указывали на разные блоки памяти, чтобы избежать циклов приостановки */

[J0 += 1] = XR0 ; [K0 += 1] = YR0;; J0= [J5 + 1] ; XR0 = [K6 + 1] ;; R1 = 0xABCD ; R0 = [J0 += 1] ;; /* Одна команда загрузки данных (in K-IALU) и одна операция J-IALU*/

XR0 = YR0 ; XR1 = [J0 += 1]; YR1 = [K0+= 1] ;; /* Недопустимо; три команды IALU */

XR0 = [J0 + 1] ; YR0 = [J1 + 1] ;; /* Недопустимо; обе команды используют одно и то же IALU (J-IALU) */

XR0 = [J0 + 1] ; J5 = J1 + 1 ;; /* Недопустимо; обе команды используют одно и то же IALU (J-IALU) */

Два обращения к одному и тому же адресу памяти, выполняемые в одной командной строке, если одно из этих обращений является записью, могут дать непредсказуемые результаты.

Загрузка из внешней памяти разрешена только в регистровые файлы вычислительного блока и IALU.

Чтение из мультипроцессорной широковещательной зоны недопустимо.

Пересылка из регистра в регистр: если один из регистров представляет собой смесь регистров вычислительных блоков, то другой не может быть регистром вычислительного блока. Например:

XYR1:0 = XR11:8 ; /* Недопустимо */

XR11:8 = XYR1:0 ; /* Недопустимо */

XYR1:0 = J11:8 ; /* Допустимо */

J11:8 = XYR1:0 ; /* Допустимо */

Обращение с использованием буфера выравнивания данных (DAB) должно осуществляться посредством загрузки квадрослова. Оно не может осуществляться посредством использования «смешанных» групп Ureg- регистров. Например:

R3:0 = DAB Q[J0+= 4] ;; /* Допустимо; широковещательная загрузка */

R1:0 = DAB Q[J0+= 4] ;; /* Недопустимо; попытка загрузить в смешанный регистр */

Обращение с использованием DAB и обращение к кольцевому буферу в памяти разрешается только [в режиме] постмодификации с обновлением. Например:

XR1:0 = CB L[J2 + 2] ;; /* Недопустимо */

Обращение к группам регистров от 0x20 до 0x3F (DMA, внешний порт, порт линков, контроллер прерываний, порт JTAG и регистры автоDMA) может осуществляться только посредством пересылок Ureg-регистров.

В пересылках из регистра в регистр, XY-регистр не может использоваться в качестве источника транзакции, за тем исключением, когда таковыми являются и источник, и приемник. Например:

R1:0 = R11:10 ;; /* Допустимо */

J1:0 = R11:10 ;; /* Недопустимо */

R3:0 = J3:0 ;; /* допустимо */>

Допускается одновременное осуществление до двух команд загрузки в регистровый файл одного и того же вычислительного блока; допускается также одновременное осуществление до одной команды загрузки в регистровый файл и одной команды сохранения из регистрового файла одного и того же вычислительного блока.

Регистровый файл вычислительного блока имеет один входной порт и один входной/выходной порт. Если выдается две команды сохранения, то ни одна из них не выполняется.

Например:

[J0 + 1] = XR0 ; [К0 + 1] = XR1 ;; /* Недопустимо; попытка использовать два выходных порта */

R0 = [J0 + 1] ; R1 = [К1 + 1] ;; /* Допустимо; используется по два входных порта в выч. блоках X и Y /

R0 = [J0 + 1] ; [К1 + 1] = XR1 ;; /* Допустимо */

Пересылка Ureg-регистров внутри регистрового файла одного и того же вычислительного блока не может быть совмещена с какой-либо другой операцией сохранения в этот же регистровый файл. Например:

XR3:0 = R7:4 ; [J17 + 2] = YR4 ;; /*Допустимо; различные регистровые файлы*/

XR3:0 = R7:4 ; XR10 = [J17 + 2] ;; /* Допустимо; одна Ureg-пересылка и одна загрузка в выч.блок X */

XR3:0 = R7:4 ; [J17 + 2] = XR4 ;; /* Недопустимо; одна Ureg-пересылка и одно сохранение из выч.блока X */

R3:0 = R31:28 ;; /* Допустимо — SIMD Ureg-пересылка /

R3:0 = R31:28 ; [J17 + 2] = YR8 ;; /* Недопустимо —SIMD Ureg-пересылка (в обоих per.файлах) и одно сохранение из выч.блока Y */

Позволительна только одна DAB-загрузка на вычислительный блок. Например:

XR3:0 = DAB Q[J0 + = 4] ; XR7:4 = DAB Q[K0 += 4] ;; /* Недопустимо */

XR3:0 = DAB Q[J0 + = 4] ; YR7:4 = DAB Q[K0 += 4] ;; /* Допустимо */

Позволительно выполнение только одной операции загрузки/сохранения для однопортовых регистровых файлов. Однопортовыми регистровыми файлами являются:

- регистры SOC-интерфейса: группы от 0x20 до 0x3F,
- регистры процессорного ядра: группы 0x0A,0x1B, 0x1E,0x1F.

Регистры JK ALU (группы 0x0C - 0x0F), а также регистры устройства управления (группа 0x1A) имею раздельные порты чтения и записи, и допускают одновременное выполнение не более одной операции чтения и не более одной операции записи в одном такте.

Например,

k0 = j0; j0 = k0;; /* Допустимо */

cjmp = k0; k0 = cjmp;; /* Допустимо */

j0 = [j1+5]; [k1+4] = j0;; /* Допустимо */

q[j27+0] = j19:16; j19 = cjmp;; /* Допустимо */

cjmp = j19; j19:16 = q[j27+0];; /* Допустимо */

[j1+0] = j0; [k1+1] = j1;; /* Недопустимо – два чтения */

j0 = [j1+0]; j1 = xr0;; /* Недопустимо – две записи */

Адрес обращения к памяти должен быть выровнен по размеру данных.

Например, квадрословное обращение должно быть выровнено на границу квадрослова. Длинное (двусловное) обращение должно быть выровнено на четный адрес. Это не относится к загрузке в вычислительный блок через DAB. Дополнительно, непосредственный модификатор адреса должен быть кратен четырем при квадрословном доступе и двум при двусловном. Например:

XR3:0 = Q[J0 += 3] ;; /* Недопустимо */ XR3:0 = Q[J0 += 4] ;; /* Допустимо */

В следующих операциях J-IALU, использующих адресацию кольцевых буферов или бит-реверсивную адресацию, индекс Jm может быть только индексом J0, J1, J2 или J3:

Js = Jm + | - Jn (CB)

Ureg = CB [L] [Q] (Jm + | += Jn | Imm) CB [L] [Q] (Jm + / += Jn | Imm) = Ureg Ureg = DAB [L] [Q] (Jm + | += Jn | Imm) Ureg = BR [L] [Q] (Jm + | += Jn | Imm) BR [L] [Q] (Jm + | += Jn | Imm) = Ureg Ureg = BR [L] [Q] (Jm + | += Jn | Imm)

Те же ограничения применимы к командам K-IALU, которые используют адресацию кольцевых буферов или бит-реверсивную адресацию. Индекс Km может быть только индексом K0, K1, K2 или K3.

В командах загрузки и сохранения адрес памяти не может быть регистром.

Например, адрес не может быть адресом отображенного в память регистра в диапазоне от 0x1E0000 до 0x1FFFFF. Например:

Q[J2 +0] = XR3:0 ;; /* Недопустимо, если J2 в диапазоне от 0x1E0000 до 0x1FFFFF */

Если одно IALU используется для получения доступа к регистру другого IALU, то не должно быть команды непосредственной загрузки в той же строке. Например:

Q[J2 + 0] = K3:0 ; XR0 = 100 ;; /* Недопустимо */

Q[K2 + 0] = K3:0 ; XR0 = 100 ;; /* Допустимо */

Следующие транзакции запрещены:

Ureg_отладки = Ureg_устройства_управления ;; /* Недопустимо*/

Ureg_устройства_управления = Ureg_отладки ;; /* Недопустимо*/

Ureg_отладки = Ureg_отладки ;; /* Недопустимо */

/* Ureg_устройства_управления: группы 0x1A, 0x30-0x39 и 0x3B */

/* Ureg_отладки: группы 0x1B, 0x3D-0x3F */

2.4.5.3 Ограничения на команды устройства управления

На одну командную строку может приходиться одна команда устройства управления и одно непосредственное расширение, при этом командой устройства управления может быть переход, косвенный переход и другие команды. Ассемблер статически проверяет все следующие ограничения:

- команда устройства управления должна занимать первое из четырех полей командной строки;
- непосредственное расширение рассматривается как одно из четырех полей в командной строке;
- не может быть двух командных строк, которые заканчиваются в пределах одного квадрослова и притом каждая содержит команду перехода с установленным битом прогноза. Например:

  IF MLE, JUMP + 100 ;; /* начальный адрес 1000 */
  IF NALE JUMP -50 ;
  XR0 = R5 + R6 ; J0 = J2 + J3 ; YR4 = [K3 + 40] ;; /* Допустимо; первая командная строка заканчивается в слове с адресом 1001; вторая – в слове с адресом 1005 */

  IF MLE, JUMP + 100 ;; /* начальный адрес 1000 */
  IF NALE JUMP - 50 ;; /* Недопустимо; обе строки лежат в одном и том же квадрослове */

У команды с логическими операциями над статическими флагами SCFx += operation Condition не может быть логических операций между статическими флагами и вычислительного блока и (XCF0/1, YSF0/1 и XYSF0/1) условиями невычислительных блоков (J-IALU или K-IALU).

Опция Не-обновлять-флаги (NF) может использоваться независимо для каждой команды в строке, если только в этой строке нет условной команды. Для условных командных строк либо все поля команд используют опцию nf, либо не использует ни одно.

2.5 Регистры вычислительных блоков

Ядро процессора содержит два вычислительных блока – вычислительный блок X и вычислительный блок Y. В каждом из них имеется регистровый файл и четыре независимых вычислительных устройства. Этими устройствами являются ALU, умножитель, сдвиговое устройство и CLU. Так как выполнение всех вычислительных команд в процессоре зависит от форматов входных и выходных данных, а также от

того, исполняется ли данная команда на одном вычислительном блоке или на обоих, важно понимать, как правильно использовать регистры вычислительных блоков процессора. В данной главе описаны регистры вычислительных блоков, показано, как правила именования регистров определяют формат данных и место исполнения команды, а также определены все доступные форматы данных.

В разделе описываются:

- Регистры регистровых файлов;
- Числовые форматы.

В состав каждого вычислительного блока входит многопортовый регистровый файл данных общего назначения объёмом в 32 слова. Он служит для пересылки данных между вычислительными устройствами и шинами данных, а также для хранения промежуточных результатов вычислений. На рисунке 9 изображено, как каждый регистровый файл обеспечивает взаимодействие между внутренними шинами и вычислительными устройствами внутри вычислительных блоков.

Как показано на рисунке 9, перед записью в регистровый файл данные могут быть пропущены через буфер выравнивания данных (DAB). DAB представляет собой два квадрословных FIFO и обеспечивает выравнивание данных в регистрах в случаях дву- и квадро-регистрового чтения невыровненных данных из памяти. Более подробную информацию о DAB см. в главе «IALU».

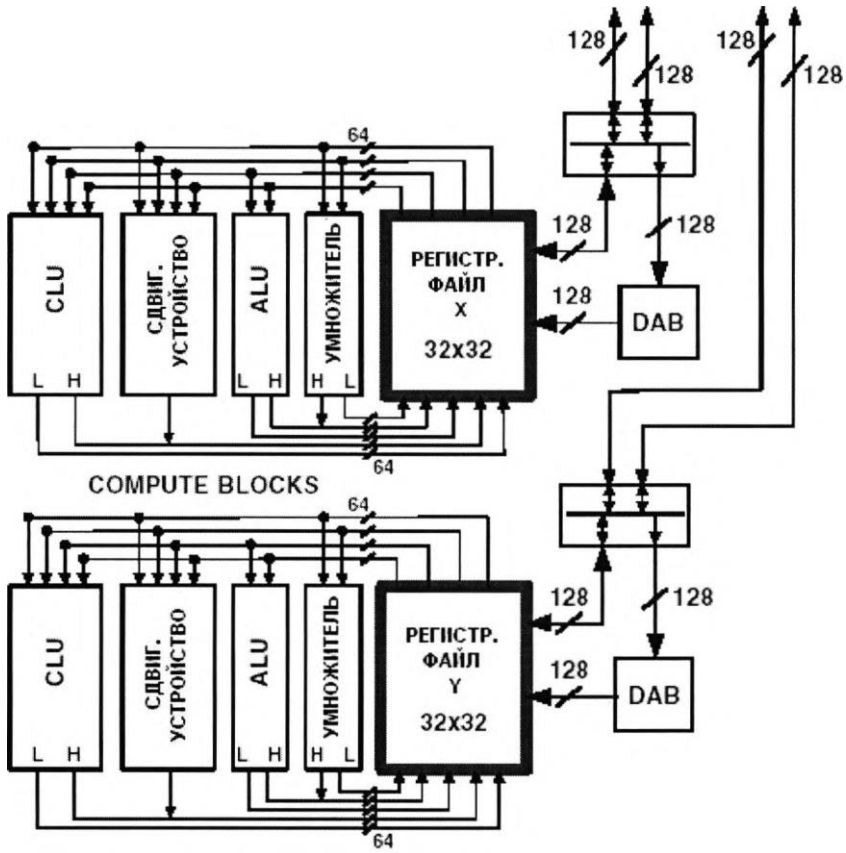


Рисунок 9 – Регистровые файлы вычислительных устройств X и Y

В каждом вычислительном блоке имеется два типа регистров: отображённые в память регистры и не отображённые в память регистры. В каждом вычислительном блоке отображёнными в память регистрами являются все регистры общего назначения регистрового файла данных: XR31-0 и YR31-0. Будучи отображёнными в память, эти регистры доступны для устройств на внешней шине.

Важно различать отображённые в память и не отображённые в память регистры, так как отображённые в память регистры являются *универсальными (Ureg).* Эти регистры называются универсальными, потому что доступ к ним можно получить через внутренние шины ядра процессора; таким образом, Ureg-регистры доступны для всех ведущих как на внутренней шине (например, вычислительные блоки, IALU и др.), так и на внешней шине (например, хост-процессор или другие процессоры). *Ureg*-регистры вычислительных блоков могут быть использованы в дополнительных операциях, неприменимых к другим *Ureg*-регистрам. Для того, чтобы отличать регистры регистровых файлов вычислительных блоков от других Ureg-регистров, регистры XR31-0 и YR31-0 называют также *регистрами данных (Dreg).* Не отображённые в память регистры (например, MR, PR и др.) доступны только в пределах исполнительных устройств, в которых используются.

Тот факт, что большинство регистров процессора отображено в память, оказывается весьма полезным при работе в многопроцессорной системе. Отображённые в память регистры имеют собственные абсолютные адреса, что делает их доступными для других процессоров через мультипроцессорное адресное пространство или для любого другого ведущего на шине в системе.

В вычислительных блоках имеется несколько регистров, не отображённых в память. Этим регистрам не приписаны никакие абсолютные адреса. Не отображённые в память регистры – это специальные регистры, используемые определёнными командами в каждом вычислительном блоке. Не отображённые в память регистры вычислительных блоков включают в себя:

- статусные регистры вычислительных блоков (XSTAT и YSTAT);
- регистры побочных результатов операций (XPR1-0 и YPR1-0) ALU;
- регистры результата (XMR3-0 и YMR3-0) и переполнения результата (XMR4 и YMR4) умножителя;
- регистры временного однобитного FIFO переполнения (XBFOTMP и YBFOTMP) сдвигового устройства;
- регистры результата (XTR31-0 и YTR31-0) и истории решётчатых операций (XTHR3:0 и YTHR3:0), а также коммуникационный управляющий регистр (XCMCTL и YCMCTL) CLU.

Таблица 3 – Регистр состояния вычислительного модуля XSTAT и YSTAT

Бит	Имя	Назначение
0	AZ	ALU. Признак нулевого результата операции
1	AN	ALU. Признак отрицательного результата операции
2	AV	ALU. Признак переполнения результата операции
3	AC	ALU. Признак переноса
4	MZ	Умножитель. Признак нулевого результата операции
5	MN	Умножитель. Признак отрицательного результата операции
6	MV	Умножитель. Признак переполнения результата операции
7	MU	Умножитель. Признак исчезновения (underflow) результата операции
8	SZ	Сдвигатель. Признак нулевого результата операции
9	SN	Сдвигатель. Признак отрицательного результата операции
10	BF0	Сдвигатель. Флаг блока операций с плавающей запятой
11	BF1	Сдвигатель. Флаг блока операций с плавающей запятой
12	AI	Ошибочная (Invalid) операция АЛУ с плавающей запятой
13	MI	Ошибочная (Invalid) операция умножителя с плавающей запятой
14	TROV	CLU. Переполнение операции trellis

Бит	Имя	Назначение
15	TRSOV	CLU. Переполнение операции trellis. Stiky бит
16	MIN_MAX_NUM	Разрешение вычисления номера максимума или минимума команд АЛУ MAX или MIN
17	-	
18	-	
19	-	
20	UEN	Разрешение (если 1) генерации исключительной ситуации при возникновении флага исчезновения (underflow) результата
21	OEN	Разрешение (если 1) генерации исключительной ситуации при возникновении флага переполнения (overflow) результата
22	IVEN	Разрешение (если 1) генерации исключительной ситуации при возникновении флага ошибочного (invalid) результата при операциях с плавающей запятой
23	-	
Stiky биты		
24	AUS	Underflow в АЛУ с плавающей запятой
25	AVS	Переполнение в АЛУ с плавающей запятой
26	AOS	Переполнение в АЛУ с фиксированной точкой
27	AIS	Ошибочный результат в АЛУ с плавающей запятой
28	MUS	Underflow в умножителе с плавающей запятой
29	MVS	Переполнение в умножителе с плавающей запятой
30	MOS	Переполнение в умножителе с фиксированной точкой
31	MIS	Ошибочный результат в умножителе с плавающей запятой

Все не отображённые в память регистры вычислительных блоков используются для специальных целей. Регистры X/YSTAT (описаны в таблице 3, см. также раздел «Загрузка/Пересылка статусных регистров вычислительных блоков (X/YSTAT)») содержат флаги статуса соответствующих вычислительных блоков. Эти флаги устанавливаются и сбрасываются в соответствии с результатом исполнения команды в ALU, умножителе, сдвиговом устройстве или CLU вычислительного блока. Регистры X/YPR1-0 содержат побочные результаты выполнения команд SUM, ABS, VMAX и VMIN арифметико-логического устройства. При необходимости регистры X/YMR3-0 используются для хранения результатов операций умножения с фиксированной точкой, а регистр X/YMR4 – для хранения значений переполнения в результате этих операций. Регистры X/YBFOTMP используются для временного хранения или размещения возвращаемого значения переполнения в командах GETBITS и PUTBITS.

В процессоре регистры X/YTR31-0 и X/YTHR3:0 решётчатых операций используются в командах DESPREAD и XCORRS. Коммуникационный управляющий регистр (X/YCMCTL) служит возможным источником для значения CUT в команде XCORRS.

2.5.1 *Регистры регистровых файлов*

Регистровые файлы вычислительных блоков X и Y содержат тридцать два 32-битных регистра, с помощью которых реализуются взаимодействия внутренних шин процессора и вычислительных устройств вычислительных блоков. Регистры регистровых файлов (XR31-0 и YR31-0) являются одновременно и универсальными регистрами *(Ureg),* и регистрами данных *(Dreg).*

Входные данные для вычислительных операций обычно берутся из регистрового файла. Однако, в некоторых случаях входные данные могут быть взяты из внутренних регистров вычислительных блоков. Результаты сохраняются в регистровом файле, кроме тех случаев, когда они сохраняются во внутренних регистрах вычислительных блоков.

Примечание – регистр может быть указан как выходной операнд (в котором сохраняется результат) лишь один раз в командной строке, однако, синтаксис ассемблера допускает многократное использование одного и того же регистра в качестве входного операнда (содержащего входные данные) для нескольких команд в одной командной строке (содержащей до четырёх полей команд).

Регистры регистровых файлов связаны на аппаратном уровне таким образом, что при каждой вычислительной операции происходит проверка зависимостей по регистрам, чтобы обеспечить использование корректных значений. При осуществлении доступа к регистру во время вычислительной операции, процессор выполняет такую проверку, чтобы удостовериться, что данный регистр не связан какой-либо иной зависимостью. Более подробную информацию по командным строкам и зависимостям см. в разделах «Синтаксис и структура командной строки» и «Правила запараллеливания команд».

Синтаксис языка ассемблера процессора допускает множество вариантов именования регистров. Правила именования регистров позволяют специфицировать множество деталей, связанных с вычислительными командами. Используя правила именования регистров в команде, можно указать:

- используемые вычислительные блоки;
- ширину регистров;
- размеры операндов;
- форматы данных.

На рисунке 10 показаны правила именования регистров и отмечены поля, определяющие эти характеристики.

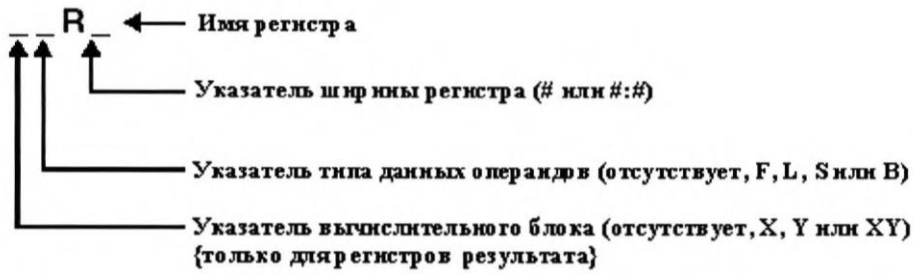


Рисунок 10 – Правила именования регистров регистровых файлов

Примечание – Синтаксис языка ассемблера позволяет также использовать целочисленный, дробный, а также действительный или комплексный типы данных. Указание одного их этих типов данных происходит с помощью опций команд и не является частью правил именования регистров регистровых файлов.

2.5.1.1 Указание вычислительных блоков

Как показано на рисунке 10, правила именования регистров позволяют определить, какому вычислительному блоку принадлежат регистры, с которыми вы работаете.

Префиксы X и Y перед именами регистров указывают вычислительный блок, в котором находится этот регистр: X – только вычислительный блок X, Y – только

вычислительный блок Y и XY (или без префикса) – оба. Следующие команды ALU приведены как примеры использования правил именования регистров.

```
XR0 = R1 + R2 ;; /*Эта команда выполняется на вычислительном устройстве X */
```

Эта команда использует регистры XR0, XR1 и XR2.

```
YR1 = R5 + R6 ;; /* Эта команда выполняется на вычисл. устройстве Y */
```

Эта команда использует регистры YR1, YR5 и YR6.

```
XYR0 = R0 + R2 ;; /* Эта команда выполняется на вычисл. устройствах X и Y */
```

Эта команда использует регистры XR0, XR2, YR0 и YR2.

```
R0 = R22 + R3 ;; /* Эта команда выполняется на вычисл. устройствах X и Y */
```

Эта команда использует регистры XR0, XR22, XR3, YR0, YR22 и YR3.

Префикс выбора вычислительных блоков позволяет указать, выполнять ли команду на одном, или на обоих вычислительных блоках, обеспечивая тем самым выбор одного из режимов выполнения команды: Одна Команда - Одни Данные (SISD) или Одна Команда - Множественные Данные (SIMD). Использование SIMD предоставляет мощное средство оптимизации исполнения в случаях, когда один и тот же алгоритм используется для обработки нескольких каналов данных. Синтаксис SIMD (XYRs = Rm op Rn) указывает процессору, что одна и та же команда должна быть выполнена на обоих вычислительных блоках.

Стоит обратить внимание, что SISD и SIMD не являются включаемыми или выключаемыми режимами, которые дают задержку при переключении. Исполнение в режиме SISD или SIMD возможно всегда, так как указание на работу в одном из этих режимов осуществляется просто в именах регистров.

Необязательные элементы команд приводятся в фигурных скобках {}. Варианты значений, которые могут принимать элементы команд, перечисляются через вертикальную черту |. Следующий пример определения синтаксиса и удовлетворяющей ему команды демонстрирует разницу в нотации синтаксиса и кода при указании вычислительного блока:

```
{X|Y|XY}Rs = Rm + Rn ;;

/* необязательные параметры указываются в фигурных скобках */
/* варианты значения параметра разделены вертикальными чертами */

XYR0 = R1 + R0 ;;

/* код, фигурные скобки и вертикальные черты отсутствуют */
```

2.5.1.2 Указание ширины регистров

Как показано на рисунке 10, правила именования регистров позволяют выбрать ширину регистра, с которым вы работаете.

Каждый конкретный регистр регистровых файлов (XR31-0 и YR31-0) является 32-битным. Чтобы обеспечить поддержку размеров данных, больших, чем 32-битные слова, синтаксис языка ассемблера позволяет объединять регистры для хранения больших слов. Правила именования регистров, для указания ширины регистров, таковы:

- *Rs, Rm* или *Rn* обозначают *одиночный регистр,* содержащий 32-битное слово (или меньший объём данных).
  Примерами имён одиночных регистров могут быть R1, XR2 и др.
- *Rsd, Rmd* или *Rnd* обозначает *двойной регистр,* содержащий 64-битное слово (или меньший объём данных).
  Примерами имён двойных регистров могут быть R1:0, XR3:2 и др. Здесь

последний номер регистра должен быть либо равным нулю, либо кратным двум.

- *Rsq, Rmq* или *Rnq* обозначает *квадрорегистр,* содержащий 128-битное слово (или меньший объём данных).
  Примерами имён квадро-регистров могут быть R3:0, XR7:4 и др. Здесь последний номер регистра должен быть либо равным нулю, либо кратным четырем.

Имена регистров, перечисленные выше, понимаются как шаблоны имён регистров и выделяются курсивом в коде. Такая запись используется вместо указания всех возможных имён регистров в команде. Следующий пример описания синтаксиса и удовлетворяющей ему команды демонстрирует разницу в нотации синтаксиса и кода при указании ширины регистра:

```
{X | Y | XY) Rsd = Rmd + Rnd ;;

/* подставляемые имена регистров, курсивом выделены шаблоны */

XR1:0 = R3:2 + R1:0 ;;

/* код, подстановка произведена */
```

2.5.1.3 Указание типов данных операндов

Как показано на рисунке 10, правила именования регистров позволяют указать размеры операндов, а также выбрать формат данных, размещаемых в регистре: данные с фиксированной точкой или данные с плавающей точкой.

Одиночные, двойные и квадро-регистры регистровых файлов (Rs, Rsd, Rsq) содержат операнды (входные и выходные данные) команд. В зависимости от размеров операнда, а также от его представления в формате с фиксированной или плавающей точкой, один регистр может содержать более одного операнда.

Для указания размера операндов в регистре регистрового файла в имени регистра используется префикс, определяющий размер, не превышающий размер самого регистра. Для данных с фиксированной точкой такими префиксами являются:

- B – указывает на однобайтное (8-битное) слово данных. В этом случае данные в 32-битном регистре интерпретируются как четыре 8-битных слова. Примерами имён регистров с однобайтными операндами могут быть BR1, BR1:0 и BR3:0.
- S – указывает на короткое (16-битное) слово данных. В этом случае данные в 32- битном регистре интерпретируются как два 16-битных слова. Примерами имён регистров с короткими операндами могут быть SR1, SR1:0 и SR3:0.
- отсутствие префикса указывает на обычное (32-битное) слово данных. Примерами имён регистров с обычными операндами могут быть R0, R1:0 и R3:0.
- L – указывает на длинное (64-битное) слово данных. Примером имени регистра с длинными операндами может быть LR1:0.

Примечание – Префиксы B, S и L применимы только к командам ALU и сдвигового устройства. Выбор размеров операндов в командах умножителя происходит несколько иным способом. Более подробно см. в разделе «Операции умножителя»

Чтобы различать данные в регистре в формате с фиксированной и плавающей точкой, в имени регистра используется префикс F, указывающий, что данные в регистре представлены в формате с плавающей точкой. Процессор поддерживает следующие размеры данных в формате с плавающей точкой:

- отсутствие префикса указывает на данные с фиксированной точкой;
- FRs, FRm или FRn (данные с плавающей точкой в одиночном регистре) – указывает на обычное 32-битное слово данных в формате IEEE. Примером имени регистра с обычными операндами в формате с плавающей точкой может быть FR3;
- FRsd, FRmd или FRnd (данные с плавающей точкой в двойном регистре) – указывает на расширенное (40-битное) слово данных либо на данные стандартной двойной точности в формате IEEE (64-битное слово). Примером имени регистра с расширенными (двойной точности) операндами в формате с плавающей точкой может быть FR1: 0.

Надо отметить, что размеры операндов влияют на исполнения команд. Например, команда SRsd = Rmd + Rnd ;; представляет собой сложение четырёх коротких операндов, расположенных в двух парах регистров. Ниже приведён конкретный пример команды такого вида, а на рисунке 11 показаны результаты выполнения этой команды.

SR1:0 = R31:30 + R25:24 ;;

На рисунке 11 показано, что команда выполняет операцию над всеми 64-мя битами, однако, над каждой группой из 16 бит в отдельности.

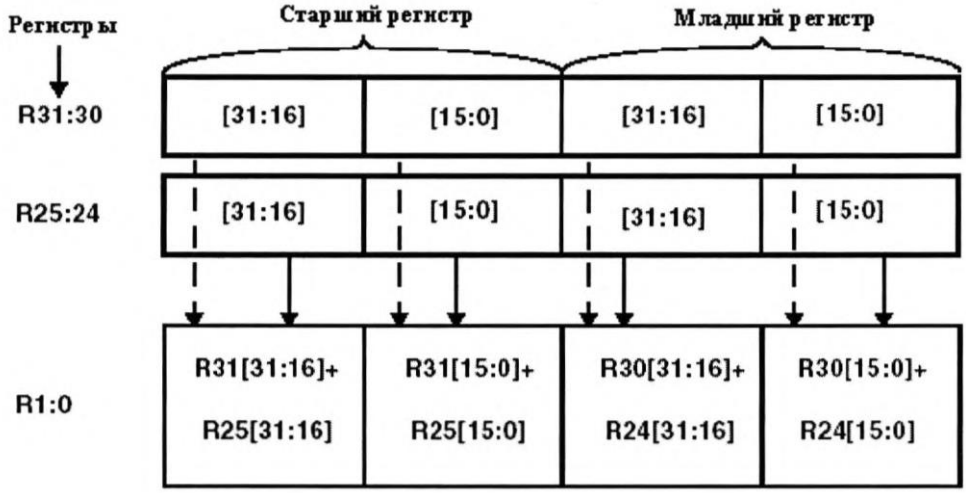


Рисунок 11 – Сложение четырёх коротких операндов в двойных регистрах

2.5.1.4 Сводка правил именования регистров

Регистры регистровых файлов данных используются в вычислительных командах, а также в командах загрузки/сохранения. Синтаксис этих команд описан в разделах:

- Арифметико-логическое устройство (ALU);
- Коммуникационно-логическое устройство (CLU);
- Умножитель;
- Сдвиговое устройство.

Далее на примере команды ALU показаны соглашения, принятые в правилах именования регистров регистрового файла данных.

{X | Y | XY} {F}Rsd = Rmd + Rnd ;;

Здесь:

- {X|Y |XY} – префиксы X, Y или XY (отсутствие префикса имеет тот же

смысл, что и префикс XY) в именах регистров указывают на каком(их) вычислительном(ых) блоке(ах) должна быть выполнена данная команда. Фигурные скобки означают, что этот элемент имени регистра опционален, а вертикальные черты – что может быть выбран лишь один вариант значения этого элемента.

- {F} – префикс F указывает на то, что операция должна быть проведена над данными в формате с плавающей точкой. Отсутствие этого префикса означает, что данные представлены в формате с фиксированной точкой.
- Rsd – тот факт, что результат сохраняется в двойном регистре, обозначен буквой d. Имя регистра в этом случае имеет вид R#:#, где второй номер кратен двум (как например, в R1: 0).
- Rmd, Rnd – входные данные представляют собой двойные регистры. Разные буквы т и п означают, что указанные регистры должны быть различными.

Ниже приведены примеры к правилам именования регистров. На рисунке 12 имя регистра XBR3 означает, что в операции используются четыре 8-битных слова в формате с фиксированной точкой, расположенные в регистре данных R3 вычислительного блока X. На рисунке 13 имя регистра XSR3 означает, что в операции используются два 16-битных слова в формате с фиксированной точкой, расположенные в регистре данных R3 вычислительного блока X. На рисунке 14 имя регистра XR3 означает, что в операции используется одно 32-битное слово в формате с фиксированной точкой, расположенное в регистре данных R3 вычислительного блока X. А имя регистра XFR3 на рисунке 14 означает, что в операции используются данные в формате с плавающей точкой.

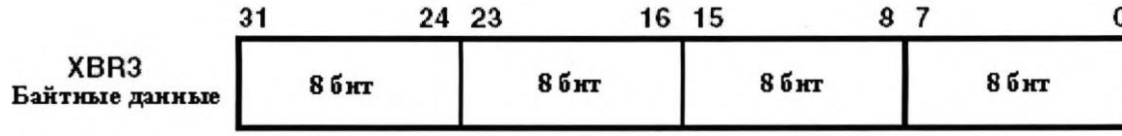


Рисунок 12 – Регистр R3 вычислительного блока X, байтовые операнды

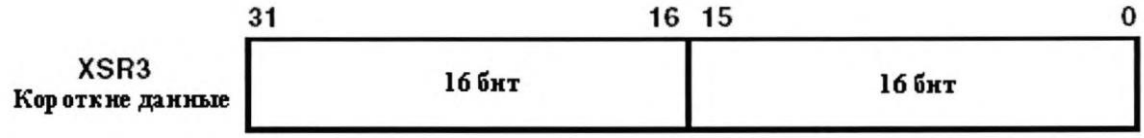


Рисунок 13 – Регистр R3 вычислительного блока X, короткие операнды

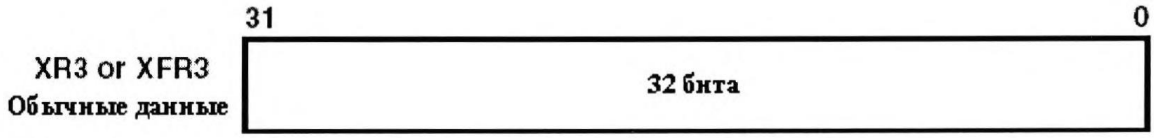


Рисунок 14 – Регистр R3 вычислительного блока X, обычные операнды

На рисунках 15, 16 и 17 показано использование различных размеров операндов в двойных регистрах, аналогичных приведённым на рисунках 12, 13 и 14.

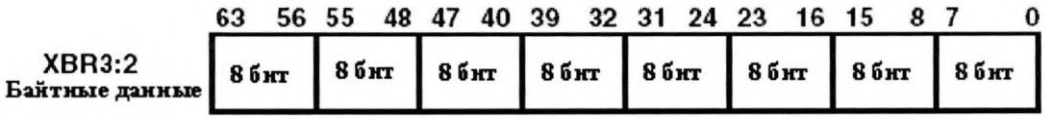


Рисунок 15 – Регистр R3:2 вычислительного блока X, байтовые операнды

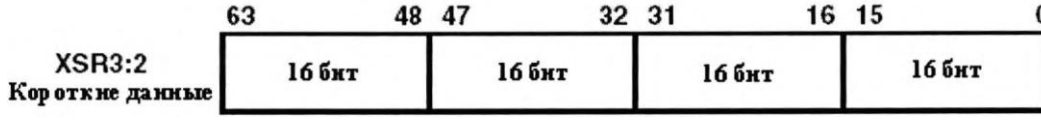


Рисунок 16 – Регистр R3:2 вычислительного блока X, короткие операнды

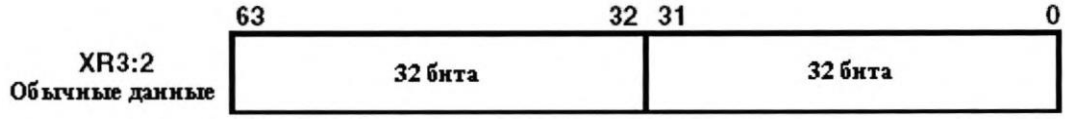


Рисунок 17 – Регистр R3:2 вычислительного блока X, обычные операнды

Примеры на рисунках 18 и 19 демонстрируют регистры, содержащие единственный операнд.

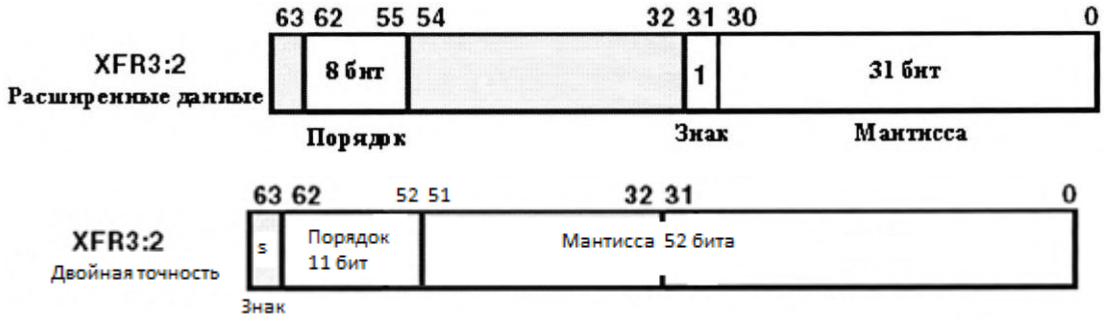


Рисунок 18 – Регистр R3:2 вычислительного блока X, расширенный операнд и операнд стандартной двойной точности (в формате с плавающей точкой)

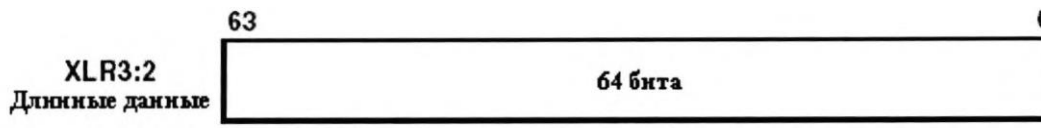


Рисунок 19 – Регистр R3:2 вычислительного блока X, длинный операнд

2.5.2 *Числовые форматы*

Процессор поддерживает 32-битный и 64-битный форматы данных с плавающей точкой одинарной и двойной точности, определённые в стандарте IEEE 754/854. Дополнительно процессор поддерживает формат расширенной точности, а именно 40-битную версию 32-битного формата, но с восемью дополнительными битами в мантиссе. Формат расширенной точности может использоваться только в случае использования старого программного обеспечения, поддерживавшего данный формат. Данный формат не имеет никаких преимуществ по сравнению с форматом стандартной двойной точности, т.к. использует тот же объем данных и обрабатывается за то же время.

Процессор также поддерживает 8-, 16-, 32- и 64-битные данные в формате с фиксированной точкой – как дробные, так и целочисленные – которые могут быть как знаковыми (в дополнительном представлении), так и беззнаковыми.

2.5.2.1 Формат данных с плавающей точкой одинарной точности в стандарте IEEE

Стандарт IEEE 754/854 определяет 32-битный формат данных с плавающей точкой одинарной точности, как показано на рисунке 20. Число, представленное в

этом формате, состоит из бита знака *s,* 24-битной мантиссы и 8-битного беззнакового порядка *e.*

Для нормализованных чисел мантисса состоит из 23-битной дробной части *f* и скрытого бита, равного 1 и неявно подразумеваемого в позиции перед битом f22 мантиссы. Предполагается, что двоичная точка находится как раз между этим скрытым битом и битом f22. Самым младшим битом (LSB) дробной части является f0; битом LSB порядка является e0.

Скрытый бит повышает эффективную точность мантиссы формата с плавающей точкой до 24 бит при фактически присутствующих в формате 23 битах. Этот бит также гарантирует, что мантисса любого нормализованного в соответствии со стандартом IEEE числа всегда больше или равна 1 и меньше 2.

Для обычных чисел в формате одиночной точности порядок может принимать значения в диапазоне 1<= *e* <= 254. Порядок в этой записи смещён на +127 (254/2). Истинное несмещённое значение порядка может быть вычислено путём вычитания 127 из *e.*

Рисунок 20 – 32-битный формат данных с плавающей точкой одинарной точности в стандарте IEEE

Стандарт IEEE также определяет несколько специальных типов данных формата с плавающей точкой одинарной точности:

- Значение порядка 255 (все биты равны 1) с ненулевой дробной частью представляет не-число (NAN). NAN обычно используется в качестве контрольных меток в потоке данных, значений неинициализированных переменных и результатов некорректных операций, таких как 0 *∞ (ноль умножить на бесконечность).
- Бесконечность имеет представление в виде числа с порядком 255 и нулевой дробной частью. Так как числа в данном формате имеют знак, то и положительная, и отрицательная бесконечность имеет представление.
  Ноль представляется в виде числа с нулевым порядком и нулевой дробной частью. Также как и с бесконечностью, предусмотрен как положительный, так и отрицательный ноль.

Все типы данных с плавающей точкой одинарной точности в стандарте IEEE, поддерживаемые процессором, а также их интерпретация приведены в таблице 4.

Таблица 4 – Типы данных с плавающей точкой одинарной точности в стандарте IEEE

Тип данных	**Порядок**	**Дробная часть**	**Значение**
NAN	255	Ненулевая	Не определено
Бесконечность	255	0	$(-1)^S$ бесконечность
Обычные	1 <= e <= 254	Любая	$(-1)^S(1.f22\text{-}0)2^{e-127}$
Ноль	0	0 (любая)	$(-1)^S$ ноль

Процессор совместим с форматом данных с плавающей точкой одинарной точности в стандарте IEEE во всех отношениях, кроме:

- процессор не имеет флага неточности;
- NAN во входных данных приводит к исключению по некорректному операнду, в результате операции возвращается тихий NAN. Когда один или оба входных операнда NAN, результатом операции будет значение 0xFFFF_FFFF;
- денормализованные операнды обнуляются на входах вычислительных устройств и не порождают исключения по потере значимости. Любое денормализованное число или число, потерявшее значимость, получившееся в результате арифметических операций, сбрасывается в ноль и порождает исключение по потере значимости;
- поддерживаются режимы округления к нулю и ближайшему числу. Режимы округления к ± бесконечности не поддерживаются.

2.5.2.2 Формат с плавающей точкой расширенной точности

Формат с плавающей точкой расширенной точности использует 40 бит с тем же 8-битным порядком, что и в формате с одинарной точностью, но с 32-битной мантиссой. Этот формат показан на рисунке 21. Во всех других отношениях формат с плавающей точкой расширенной точности аналогичен формату одинарной точности в стандарте IEEE.

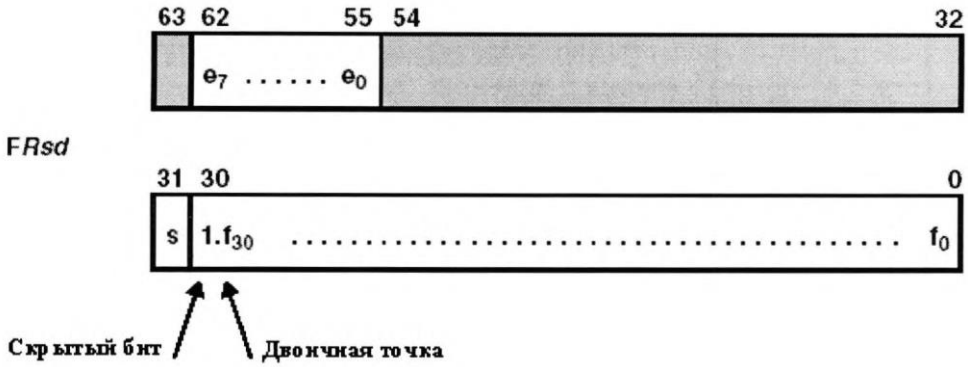


Рисунок 21 – 40-битный формат с плавающей точкой повышенной точности (расширенное слово)

Биты 63 и 54-32 (в старшем регистре) не используются.

2.5.2.3 Формат данных с плавающей точкой двойной точности в стандарте IEEE

Стандарт IEEE 754/854 определяет 64-битный формат данных с плавающей точкой двойной точности, как показано на рисунке 22. Число, представленное в этом формате, состоит из бита знака s, 53-битной мантиссы и 11-битного беззнакового порядка e.

Для нормализованных чисел мантисса состоит из 52-битной дробной части f и скрытого бита, равного 1 и неявно подразумеваемого в позиции перед битом f51 мантиссы. Предполагается, что двоичная точка находится как раз между этим скрытым битом и битом f51. Самым младшим битом (LSB) дробной части является f0; битом LSB порядка является e0.

Скрытый бит повышает эффективную точность мантиссы формата с плавающей точкой до 53 бит при фактически присутствующих в формате 52 битах.

Этот бит также гарантирует, что мантисса любого нормализованного в соответствии со стандартом IEEE числа всегда больше или равна 1 и меньше 2.

Для обычных чисел в формате одиночной точности порядок может принимать значения в диапазоне 1<= e <= 2046. Порядок в этой записи смещён на +1023 (2046/2). Истинное несмещённое значение порядка может быть вычислено путём вычитания 1023 из e.

Рисунок 22 – 64-битный формат данных с плавающей точкой двойной точности в стандарте IEEE

Как и для одинарной точности, стандарт IEEE также определяет несколько специальных типов данных формата с плавающей точкой двойной точности:

- значение порядка 2047 (все биты равны 1) с ненулевой дробной частью представляет не-число (NAN);
- бесконечность имеет представление в виде числа с порядком 2047 и нулевой дробной частью;
- ноль представляется в виде числа с нулевым порядком и нулевой дробной частью.

Все типы данных с плавающей точкой двойной точности в стандарте IEEE, поддерживаемые процессором, а также их интерпретация приведены в таблице 5.

Таблица 5 – Типы данных с плавающей точкой двойной точности в стандарте IEEE

Тип данных	**Порядок**	**Дробная часть**	**Значение**
NAN	2047	Ненулевая	Не определено
Бесконечность	2047	0	$(-1)^S$ бесконечность
Обычные	1 <= e <= 2046	Любая	$(-1)^S(1.f51\text{-}0)2^{e-}$
Ноль	0	0 (любая)	$(-1)^S$ ноль

Процессор совместим с форматом данных с плавающей точкой двойной точности в стандарте IEEE во всех отношениях, кроме:

- Процессор не имеет флага неточности.
  NAN во входных данных приводит к исключению по некорректному операнду, в результате операции возвращается тихий NAN. Когда один или оба входных операнда NAN, результатом операции будет значение 0xFFFF_FFFF_FFFF_FFFF.
- Денормализованные операнды обнуляются на входах вычислительных устройств и не порождают исключения по потере значимости. Любое денормализованное число или число, потерявшее значимость, получившееся в результате арифметических операций, сбрасывается в ноль и порождает исключение по потере значимости.
- Режимы округления к ближайшему и округления к нулю поддерживаются. Режимы округления к ± бесконечности не поддерживаются.

2.5.2.4 Форматы с фиксированной точкой

Процессор поддерживает дробный и целочисленный форматы с фиксированной точкой для 16-, 32- и 64-битных данных. Процессор поддерживает целочисленный знаковый формат с фиксированной точкой для 8-битных данных. Во всех этих форматах числа могут быть как знаковыми (в дополнительном представлении), так и беззнаковыми. В дробном формате двоичная точка предполагается слева от самого старшего значащего бита. В целочисленном формате считается, что двоичная точка находится справа от LSB. Следует отметить, что в дополнительном представлении знаковый бит имеет коэффициент -1.

Данные в 8- и 16-битном форматах всегда упакованы в регистрах следующим образом: одиночный регистр содержит четыре 8-битных слова или два 16-битных, двойной регистр содержит восемь 8-битных слов или четыре 16-битных и, наконец, квадро-регистр содержит шестнадцать 8-битных слов или восемь 16-битных слов.

На выходах ALU данные обычно имеют ту же ширину и формат, что и на входах. Однако, умножитель может дать результат либо того же размера, что и входные операнды, либо удвоенного размера по сравнению с размером входных операндов. Перемножение двух 32-битных операндов даёт либо 32-, либо 64-битный результат. Перемножение двух 16- битных операндов даёт либо 16-, либо 32-битный результат; перемножение 16-битных операндов производится исключительно в рамках SIMD квадроумножения двойных регистров, с результатом, размещающимся соответственно либо в двойном регистре (когда частные 16-битные), либо в квадро-регистре (когда частные 32-битные). Если операнды беззнаковые (целочисленные или дробные), то и результат беззнаковый. Все эти форматы показаны на рисунках 33, 34, 37 и 38.

Если один из входных операндов умножителя знаковый, а второй – нет, то результат будет знаковый. Если оба операнда знаковые, результат также знаковый. Если входные операнды дробные, то результат автоматически сдвигается влево на один бит. LSB обнуляется, а бит 62 перемещается в знаковый разряд. Если оба операнда знаковые, то обычно биты 63 и 62 результата равны. За одним исключением, когда минимальная отрицательная дробь умножается на себя. Таким образом, сдвиг влево удаляет обычно избыточный бит знака, повышая тем самым точность сохраняемой наиболее значимой части произведения.

Также, если данные на умножителе имеют дробный формат, то сдвиг влево на один бит нормализует произведение и приводит его вновь к дробному формату. Знаковые форматы использующие и не использующие левый сдвиг показаны на рисунках 35 и 36.

Умножитель имеет 80-битный регистр-аккумулятор для накопления 64-битных произведений. Более подробную информацию по умножителю и регистру-аккумулятору см. в разделе «Умножитель».

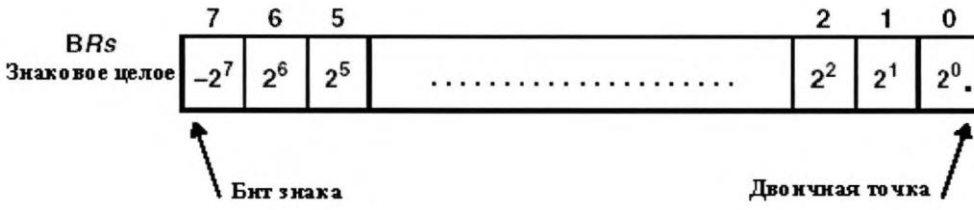


Рисунок 23 – 8-битный формат с фиксированной точкой, знаковое целое (однобайтное слово)

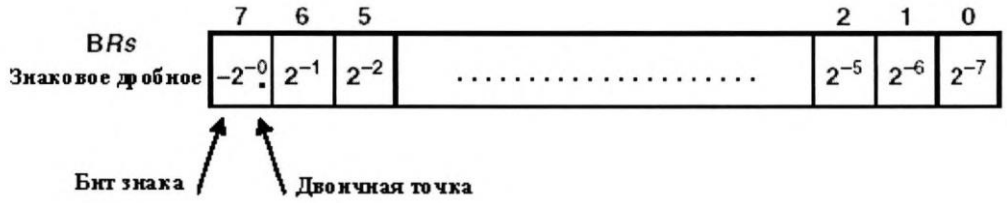


Рисунок 24 – 8-битный формате фиксированной точкой, знаковое дробное (однобайтовое слово)

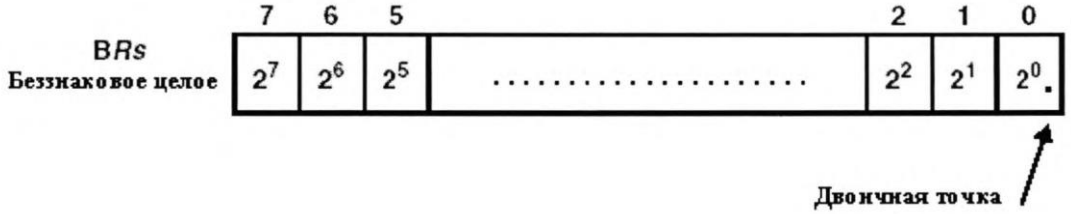


Рисунок 25 – 8-битный формат с фиксированной точкой, беззнаковое целое (однобайтное слово)

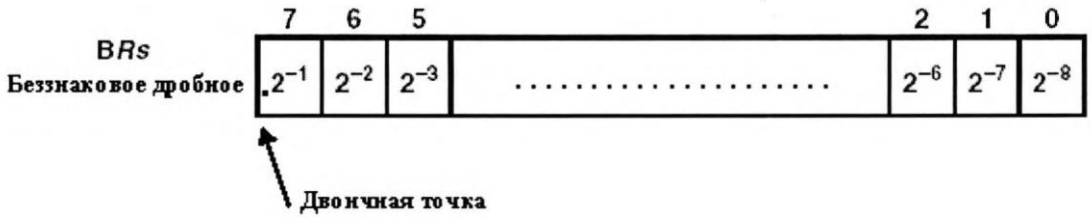


Рисунок 26 – 8-битный формат с фиксированной точкой, беззнаковое дробное (однобайтное слово)

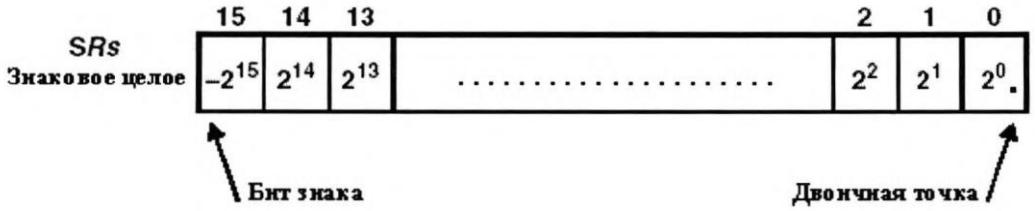


Рисунок 27 – 16-битный формат с фиксированной точкой, знаковое целое (короткое слово)

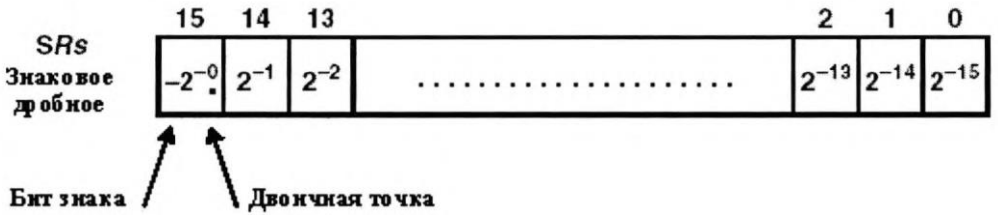


Рисунок 28 – 16-битный формат с фиксированной точкой, знаковое дробное (короткое слово)

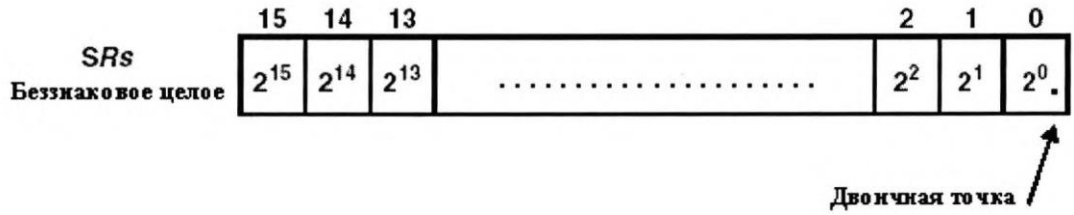


Рисунок 29 – 16-битный формат с фиксированной точкой, беззнаковое целое (короткое слово)

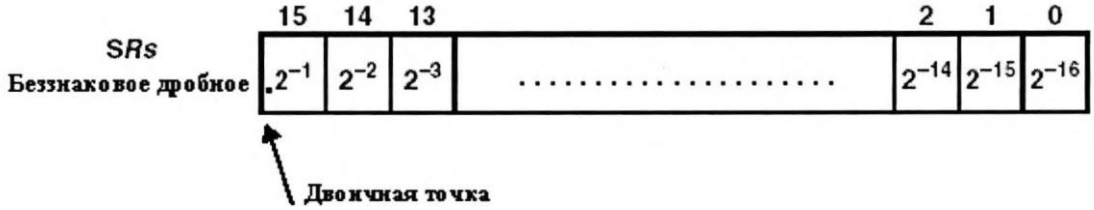


Рисунок 30 – 16-битный формат с фиксированной точкой, беззнаковое дробное (короткое слово)

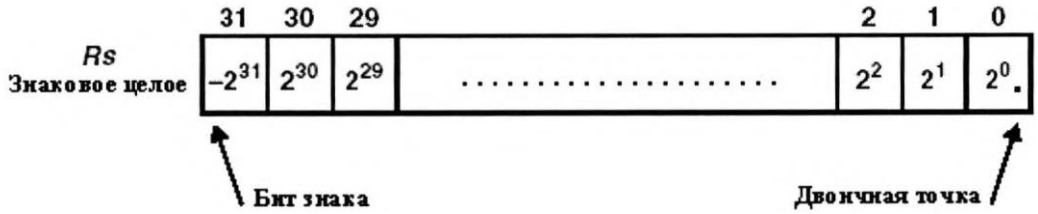


Рисунок 31 – 32-битный формат с фиксированной точкой, знаковое целое (обычное слово)

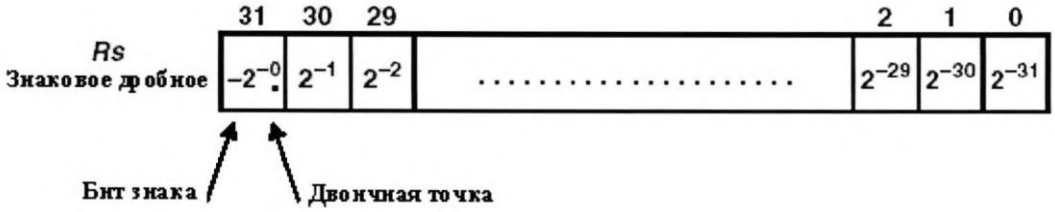


Рисунок 32 – 32-битный формат с фиксированной точкой, знаковое дробное (обычное слово)

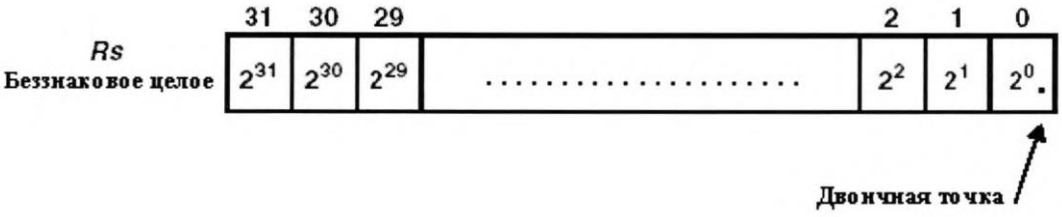


Рисунок 33 – 32-битный формат с фиксированной точкой, беззнаковое целое (обычное слово)

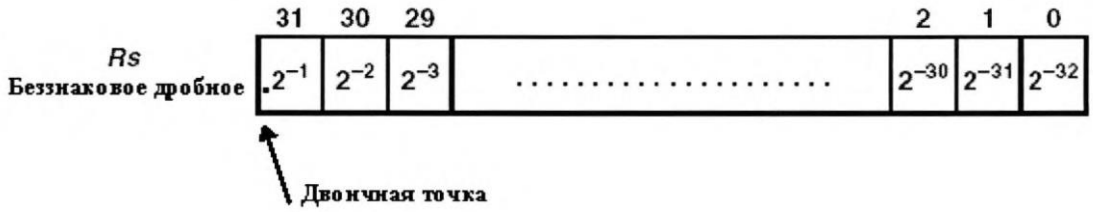


Рисунок 34 – 32-битный формат с фиксированной точкой, беззнаковое дробное (обычное слово)

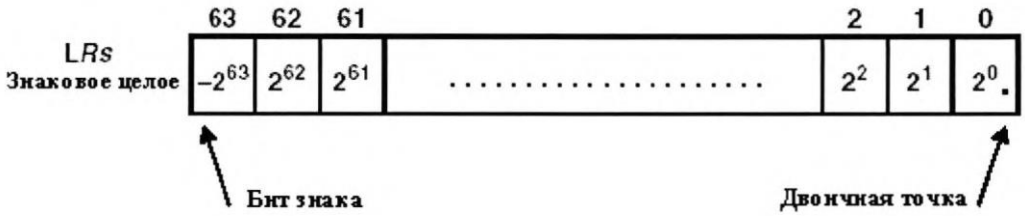


Рисунок 35 – 64-битный формат с фиксированной точкой, знаковое целое (длинное слово)

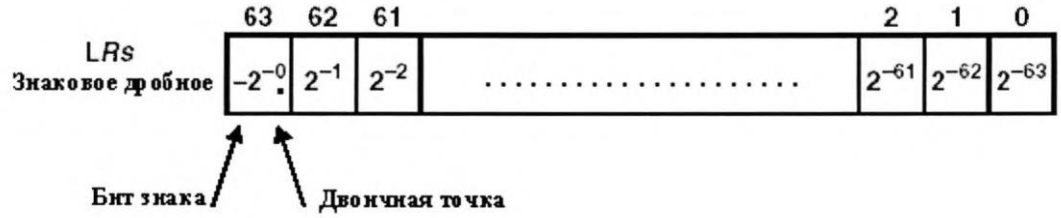


Рисунок 36 – 64-битный формат с фиксированной точкой, знаковое дробное (длинное слово)

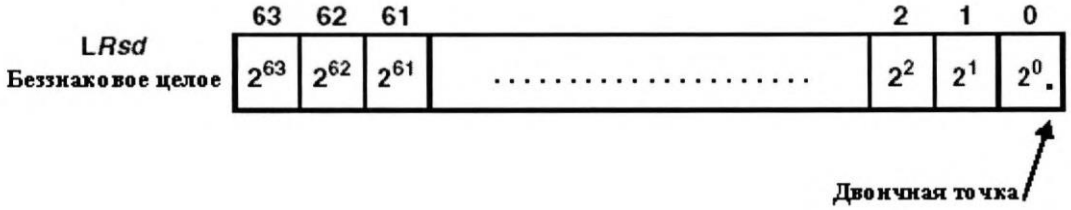


Рисунок 37 – 64-битный формат с фиксированной точкой, беззнаковое целое (длинное слово)

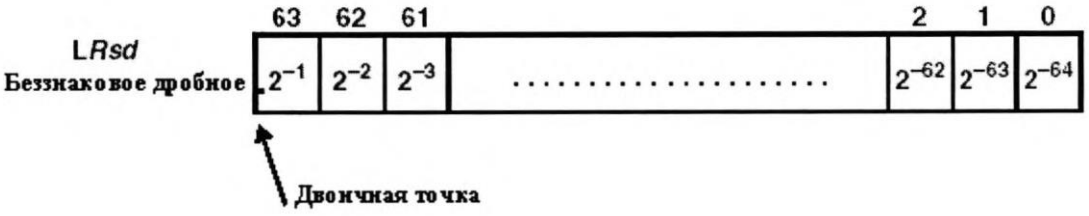


Рисунок 38 – 64-битный формат с фиксированной точкой, беззнаковое дробное (длинное слово)

3 Арифметико-логическое устройство (ALU)

Ядро процессора содержит два вычислительных устройства, известных как вычислительные блоки. Каждый вычислительный блок содержит регистровый файл и четыре независимых вычислительных устройства: ALU, CLU, умножитель и сдвиговое устройство. Арифметико-логическое устройство (ALU) выделено на рисунке 39. ALU берёт входные данные из регистрового файла и сохраняет выходные данные также в регистровом файле.

Это устройство выполняет все арифметические операции (сложение/вычитание) процессора над данными в формате с фиксированной и плавающей точкой, а также логические операции процессора над данными в формате с фиксированной точкой. ALU также выполняет операции преобразования данных, такие как операции расширения/сжатия над данными в формате с фиксированной точкой.

Не все операции ALU применимы к данным и в формате с фиксированной точкой, и в формате с плавающей точкой. 64-битное арифметико-логическое устройство каждого вычислительного блока поддерживает следующие операции ALU и типы данных соответственно:

- арифметические операции с фиксированной и плавающей точкой: сложение (+), вычитание (-), нахождение минимума (MIN), максимума (MAX), максимума по Витерби (VMAX), сравнение (COMP), отсечение по границе (CLIP) и нахождение абсолютного значения (ABS);
- арифметические операции только с фиксированной точкой: инкремент (INC), декремент (DEC), продольное сложение (SUM), побочный результат продольного сложения (PRx=SUM), подсчет единиц (ONES) и инкремент указателя однобитного FIFO (BFOINC);
- операции преобразования данных (повышение/понижение разрядности) только с фиксированной точкой: расширение (EXPAND), сжатие (COMPACT) и слияние (MERGE);
- логические операции: AND, AND NOT, OR, XOR и PASS;
- арифметические операции только с плавающей точкой: преобразование к формату с плавающей точкой (FLOAT), преобразование к формату с фиксированной точкой (FIX), копирование знака (COPYSIGN), масштабирование (SCALB), приближённое обратное или деление (RECIPS), приближённый квадратный корень или обратный квадратный корень (RSQRTS), извлечение мантиссы (MANT), извлечение порядка (LOGB), расширение операнда (EXTD) и преобразование операнда из расширенного формата в обычный (SNGL).

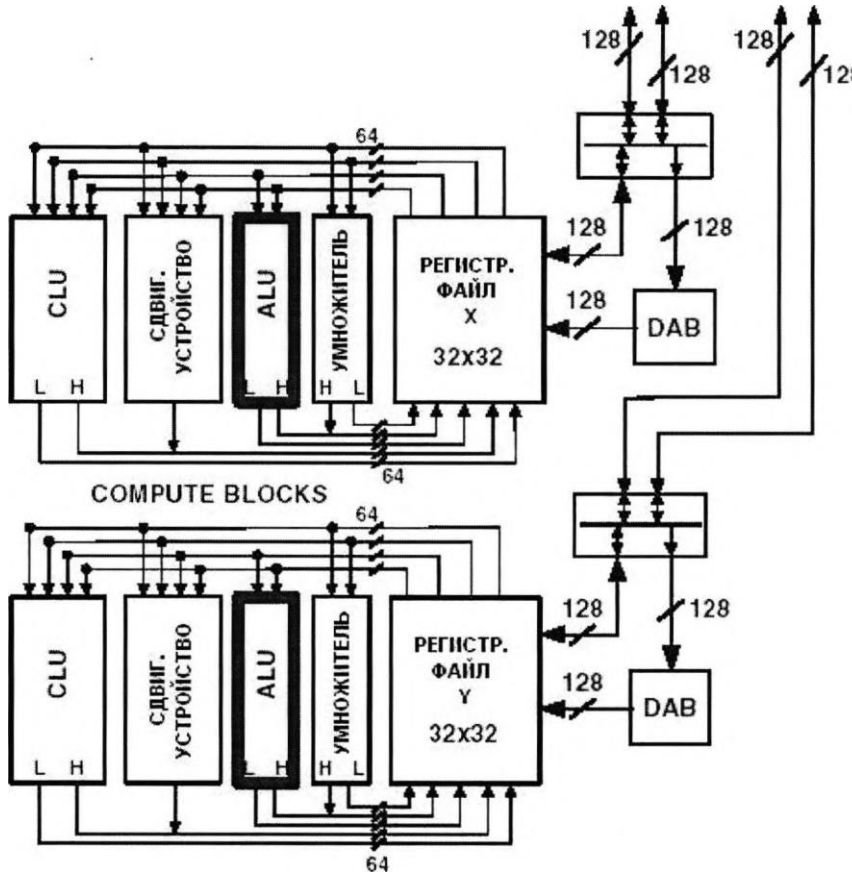


Рисунок 39 – ALU вычислительных блоков X и Y[1]

Просматривая возможные операнды каждой команды, можно увидеть, какие классы операций ALU какие типы данных поддерживают:

Арифметические операции с фиксированной точкой поддерживают:
- 8-битные (однобайтные) входные операнды;
- 16-битные (короткие) входные операнды;
- 32-битные (обычные) входные операнды;
- 64-битные (длинные) входные операнды;
- 8-, 16-, 32- и 64-битные результаты на выходе.

Операции преобразования данных с фиксированной точкой поддерживают преобразование:
- 8-битных (однобайтных) данных в 16-битные (короткие);
- 16-битных (коротких) данных в 32-битные (обычные);
- 32-битных (обычных) данных в 16-битные (короткие);
- 64-битных (длинных) данных в 32-битные (обычные);
- 128-битных (квадро) данных в 64-битные (длинные).

Логические операции поддерживают:
- 32-битные (обычные) входные операнды;
- 64-битные (длинные) входные операнды;
- 32- или 64-битные результаты на выходе.

Арифметические операции с плавающей точкой поддерживают:
- 32-битные входные операнды (по стандарту IEEE);

- 40-битные (расширенные) входные операнды;
- 64-битные (двойной точности) входные операнды (по стандарту IEEE);
- 32-, 40- или 64-битные результаты на выходе;
- Векторные операции над двумя парами 32-битных операндов (по стандарту IEEE)

Правила именования регистров определяют размер и тип данных входных операндов и результата в командах. Более подробную информацию о выборе размера и типа данных в командах ALU можно найти в разделе «Регистры регистровых файлов».

Оставшаяся часть этой главы посвящена описанию команд ALU, их опций и результатов с использованием синтаксиса команд. Объяснение соглашений по синтаксису команд ALU и других устройств, можно найти в разделе «Синтаксис и структура командной строки». Список команд ALU и их синтаксис приведён в разделе «Сводка команд ALU».

3.1 Операции ALU

ALU выполняет арифметические операции над данными в формате с фиксированной и плавающей точкой, а также логические операции над данными в формате с фиксированной точкой. Процессор использует регистры вычислительных блоков для хранения входных операндов и результата операций ALU. Регистрами регистровых файлов вычислительных блоков являются регистры XR31 - XR0 и YR31 - YR0. В ALU имеется один двойной регистр специального назначения - регистр PR - хранящий побочные результаты. Регистр PR используется процессором при выполнении различных вариаций команд SUM, ABS, VMAX и VMIN. Дополнительную информацию о регистровых файлах и правилах именования регистров для выбора типов и ширины данных можно найти в разделе «Регистры регистровых файлов». Следующие примеры представляют собой команды ALU, демонстрирующие арифметические операции.

```
XR2 = R1 + R0;;
```

/* Это операция сложения 32-битных операндов с фиксированной точкой в XR1 и XR0; результат помещается в XR2*/

```
YLR1:0 = ABS(R3:2 - R5:4);;
```

/* Это операция взятия абсолютного значения разности 64-битных операндов с фиксированной точкой в R3:2 и R5:4; результат помещается в YLR1:0; буква "L" в имени регистра результата указывает, что входные и выходные операнды должны интерпретироваться как 64-битные длинные данные */

```
XYFR2 = (R1 + R0)/2;;
```

/* Это операция сложения и деления пополам 32-битных операндов с плавающей точкой XR1+XR0 и YR1+YR0; результаты помещаются в XR2 и YR2 соответственно: это операция типа Одна-Команда, Множественные -Данные (SIMD), выполняющаяся одновременно на обоих вычислительных блоках */

Если в одном регистре хранятся множественные входные операнды (например, короткие или однобайтные слова данных), то процессор обрабатывает их параллельно. Например, предположим, что регистр YR0 содержит значение 0x00050003, а регистр YR1 – значение 0x00040008 (как показано на рисунке 40). После выполнения команды YSR2 = R0 - R1;;, регистр содержит значение 0x0001FFFB (0x01 в старшей части и -0x05 в младшей части).

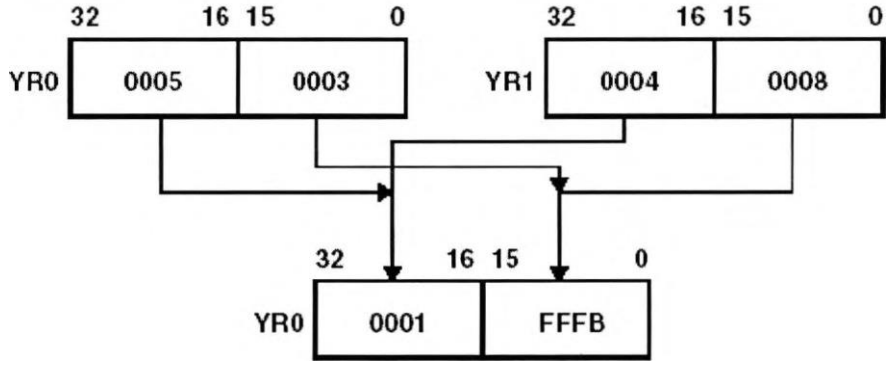


Рисунок 40 – Входные операнды для параллельного вычитания

Все команды ALU устанавливают флаги состояния для отображения статуса завершения операции, если только не использована опция Не-обновлять-флаги (NF). Так как в одной команде может быть проведено множество операций над множеством операндов, значение флага состояния вычисляется как битовое ИЛИ соответствующих флагов всех операций. Команда, показанная на рисунке 40, устанавливает флаг YAN (вычислительный блок Y, результат ALU отрицательный), так как одна из двух операций вычитания дала отрицательный результат. Более подробная информация о состоянии ALU представлена в разделе «Статус выполнения ALU».

3.1.1 *Операции с константами для команд вычислительных модулей*

В базовом варианте команд, только команды целочисленного IALU имеют возможность работы с непосредственным литералом в поле команды. В процессоре ВН28 добавлена возможность работы с константами для ALU модуля. Особенно это удобно для выполнения SIMD вычислений с длинными векторами.

Использование константы возможно для всех команд ALU вместо операнда Rn. То есть команды, использующие только один операнд Rm, не могут использоваться с константой. Исключением является векторная команда одинарной точности сложения-вычитания.

{X|Y|XY}SFRsd = Rmd + Rnd, SFRad = Rmd - Rnd {({mX}{nX} {NF}) };

Её нельзя использовать с константой ввиду отсутствия практического смысла.

3.1.2 *Опции команд ALU*

Большинству команд ALU сопоставлены опции, позволяющие гибко управлять режимом исполнения данной команды. Важно отметить, что эти опции влияют на то, как именно выполняются команды, а также тот факт, что существуют опции, применимые лишь с определёнными группами команд – не все опции используемы со всеми командами. Опции команд указываются в круглых скобках в конце поля команды. Список всех опций, используемых с конкретными командами ALU, можно найти в разделе «Сводка команд ALU».

Опции команд ALU:

- ( ) операция со знаковыми операндами, без насыщения, округление к ближайшему чётному, с дробными операндами;
- (S) операция со знаковыми операндами, с насыщением;
- (U) операция с беззнаковыми операндами, без насыщения, округление к ближайшему чётному;
- (SU) операция с беззнаковыми операндами, с насыщением;

- (X) операция расширения для команды ABS;
- (T) операция со знаковыми операндами, с усечением;
- (TU) операция с беззнаковыми операндами, с усечением;
- (Z) знаковый результат даёт ноль в команде MIN/MAX;
- (UZ) беззнаковый результат даёт ноль в команде MIN/MAX;
- (I) операция со знаковыми операндами, целочисленная;
- (IU) операция с беззнаковыми операндами, целочисленная;
- (IS) операция со знаковыми операндами, с насыщением, целочисленная;
- (ISU) операция с беззнаковыми операндами, с насыщением, целочисленная;
- (NF) не обновлять флаги.

Следующие примеры представляют собой команды ALU, демонстрирующие арифметические операции с применением опций.

```
XR2 = R1 + R0 (S) ;;
/* Это операция сложения с насыщением 32-битных (обычное слово) операндов с фиксированной точкой. */
YLFR1:0 = ABS (R3:2-R5:4) (T) ;;
/* Это операция взятия абсолютного значения разности двух операндов с плавающей точкой с усечением. */
XYFR2 = (R1+R0) /2 ( ) ;;
/* Это операция сложения и деления пополам 32-битных операндов с плавающей точкой без усечения и с округлением к ближайшему чётному; тот же эффект достигается, если круглые скобки опущены. */
```

3.1.2.1 Опция Знаковое/Беззнаковое

Процессор всегда представляет 8-, 16-, 32- или 64-битные числа в формате с фиксированной точкой, используя до четырёх 32-битных регистров данных. Данные в формате с фиксированной точкой могут быть беззнаковыми или в дополнительном представлении. Данные ALU в формате с плавающей точкой всегда знаковые величины. Информацию о поддерживаемых числовых форматах см. в разделе «Числовые форматы».

3.1.2.2 Опция Насыщение

Существует два типа арифметики с насыщением, которые могут быть выбраны для команды: знаковая или беззнаковая. В случае знакового насыщения, как только происходит переполнение (флаг AV установлен), результат операции заменяется на максимальное положительное или минимальное отрицательное значение. В случае беззнакового насыщения, при наступлении переполнения результат операции заменяется на максимальное значение или ноль. Здесь под максимальным и минимальным значениями понимаются максимальное и минимальное значения, представимые в используемом формате выходных данных.

Максимальным положительным (положительным переполнением) и минимальным отрицательным (отрицательным переполнением) значениями для знаковых форматов данных являются:

- однобайтный формат, максимальное положительное значение 0x7F, минимальное отрицательное значение 0x80;
- короткий формат, максимальное положительное значение 0x7FFF, минимальное отрицательное значение 0x8000;
- обычный формат, максимальное положительное значение 0x7FFFFFFF, минимальное отрицательное значение 0x80000000;

Максимальным положительным (положительным переполнением) и минимальным отрицательным (отрицательным переполнением) значениями для беззнаковых форматов данных являются:

- однобайтный формат, максимальное положительное значение 0xFF, минимальное отрицательное значение 0x0;
- короткий формат, максимальное положительное значение 0xFFFF, минимальное отрицательное значение 0x0;
- обычный формат, максимальное положительное значение 0xFFFFFFFF, минимальное отрицательное значение 0x0.

При использовании арифметики с насыщением флаги AV и AC отражают состояние ALU до проведения операции насыщения. Например, при знаковом насыщении, когда в результате операции происходит переполнение, возвращается максимальное или минимальное значение и флаг AV остаётся установленным. С другой стороны, флаги AN и AZ устанавливаются согласно результату после проведения операции насыщения, корректно отображая, таким образом, знак и равенство или неравенство нулю конечного результата. Это позволяет правильно вычислять предикаты AEQ, ALT и ALE даже при переполнении и использовании арифметики с насыщением.

3.1.2.3 Опция Расширение (ABS)

Опция X расширяет диапазон выходных значений команды ABS. Диапазон расширяется на один бит путём перевода результата в беззнаковый формат. Без опции X MSB результата всегда равен нулю, а максимальное отрицательное знаковое значение (0x80. .0) даёт переполнение. Когда опция X команды ABS включена, диапазон выходных значений расширяется до 0x0 - 0x80..0. Выходные значения являются беззнаковыми числами в указанном диапазоне.

3.1.2.4 Опция Усечение

Для всех команд ALU, поддерживающих усечение для операций с плавающей точкой, опция T позволяет выбрать режим округления результата. Процессор поддерживает два способа округления: округление к нулю и округление к ближайшему чётному. Эти способы округления соответствуют стандарту IEEE 754 и определяются следующим образом:

- Округление к ближайшему чётному (опция T не используется). Если результат до округления не может быть точно представлен в требуемом формате, округлённый результат вычисляется как число, ближайшее к результату до округления. Если результат до округления в точности посередине между двумя числами, представимыми в требуемом формате (отличающимися в LSB), округлённый результат вычисляется как число с LSB, равным 0.
- Округление к нулю (опция T включена). Если результат до округления не может быть точно представлен в требуемом формате, округлённый результат вычисляется как число, ближайшее к нулю. Это эквивалентно отсечению непредставимой части.

На практике округление вверх и вниз происходят с одинаковой частотой, поэтому такое округление не даёт систематического смещения. Так как максимальное значение числа в формате с плавающей точкой отличается только в LSB от представления бесконечности, результат операции, лежащий посередине между максимальным значением в формате с плавающей точкой и бесконечностью округляется к бесконечности.

3.1.2.5 Опция Возврат к нулю (MAX/MIN)

Опция Z в командах MAX/MIN меняет операции таким образом, что если большее (в случае MAX) или меньшее (в случае MIN) значение оказывается лежащим во втором входном регистре, а не в первом, то в качестве результата возвращается значение 0.

Без опции Z команда MAX может быть записана следующим псевдокодом:

If (Rmd >= Rnd) then Rsd = Rmd; else Rsd = Rnd;

ALU вычисляет, какой из регистров Rmd или Rnd содержит максимальное значение, и помещает его в Rsd.

С опцией Z команда MAX может быть записана следующим псевдокодом:

If (Rmd >= Rnd) then Rsd = Rmd; else Rsd = 0;

ALU вычисляет, какой из регистров Rmd или Rnd содержит максимальное значение. Если Rmd содержит максимальное значение, то ALU помещает его в регистр Rsd. Если же Rnd содержит максимальное значение, то ALU помещает ноль в регистр Rsd.

3.1.2.6 Опция Дробный/Целочисленный

Процессор всегда представляет 8-, 16-, 32- или 64-битные числа в формате с фиксированной точкой, используя до четырёх 32-битных регистров данных. Команды арифметико-логического устройства EXPAND и COMPACT могут быть применены к данным в дробном или целочисленном форматах. По умолчанию используется дробный формат. Опция I используется для указания целочисленного формата операндов. Информацию о поддерживаемых числовых форматах см. в разделе «Числовые форматы»

3.1.2.7 Опция Не-обновлять-флаги

Почти все вычислительные операции влияют на значение регистра состояния вычислительного блока в соответствии со своим статусом завершения. Зачастую в некоторых приложениях оказывается полезным защитить значение регистра состояния от изменения в результате выполнения операции. Существует множество способов сохранения слова состояния (например, сохранение содержимого регистра состояния или загрузка значений флагов состояния в регистр статических флагов (SFREG)). Опция Не-обновлять-флаги (NF) предоставляет другой способ работы со словом состояния. Вместо того, чтобы сохранять статус завершения операции для дальнейшего использования, программа может запретить переписывать слово состояния, используя опцию (NF) со следующей командой, предотвращая генерирование той статуса завершения. Существуют некоторые ограничения на то, каким образом опция (NF) может быть использована в условных командах. Более подробно см. раздел «Отработка условных команд».

3.1.2.8 Опция переставить Rmd

Опция (mX) меняет местами пару регистров Rmd в векторных командах с числами одинарной точности.

Пример:

SFR1:0 = R3:2 + R11:10 (mX);;

В регистре R1 окажется значение R2+R11, а в регистре R0 – R3+R10.

3.1.2.9 Опция переставить Rnd

Опция (nX) меняет местами пару регистров Rnd в векторных командах с числами одинарной точности.

Пример:

SFR1:0 = R3:2 + R11:10 (nX);;

В регистре R1 окажется значение R3+R10, а в регистре R0 – R2+R11.

3.1.2.10 Опция FJ

Опция (FJ) может использоваться только в команде векторного умножение чисел одинарной точности, и только в её варианте с константой. Используется для комплексного умножения. Подробнее см. в описании команды.

3.1.3 *Статус выполнения ALU*

Операции ALU изменяют флаги состояния арифметических регистров состояния вычислительных блоков XSTAT и YSTAT (таблица 3). Программы могут использовать флаги состояния для управления исполнением условных команд и порождения программных исключений. Более подробно, см. раздел «ALU-предикаты». В таблице 6 приведены флаги регистров XSTAT и YSTAT, отражающие статус завершения последней операции ALU.

Таблица 6 – Флаги состояния ALU

Флаг	**Определение**	**Изменяется...**
AZ	Признак равенства нулю в операции ALU с фиксированной точкой или потери значимости в операции с плавающей точкой	Всеми операциями ALU
AN	Признак отрицательного значения в операции ALU	Всеми операциями ALU
AV	Признак переполнения в операции ALU	Всеми арифметическими операциями
AC	Признак переноса в операции ALU	Всеми операциями с фиксированной точкой; очищается операциями с плавающей точкой
AI	Признак некорректной операции с плавающей точкой ALU	Всеми операциями с плавающей точкой; очищается операциями с фиксированной точкой

В таблице 7 приведены флаги регистров XSTAT и YSTAT, отражающие залипающий статус завершения последней операции ALU. Установленный однажды, залипающий флаг остаётся в этом положении до тех пор, пока не будет сброшен явно.

Таблица 7 – Залипающие флаги состояния ALU

Флаг	**Определение**	**Изменяется...**
AUS	Признак потери значимости в операции ALU с плавающей точкой, залипающий	Всеми операциями с плавающей точкой
AVS	Признак переполнения в операции ALU с плавающей точкой, залипающий	Всеми операциями с плавающей точкой
AOS	Признак переполнения в операции ALU с фиксированной точкой, залипающий	Всеми операциями с фиксированной точкой
AIS	Признак некорректной операции с плавающей точкой ALU, залипающий	Всеми операциями с плавающей точкой

Обновление флагов происходит в конце выполнения каждой операции, делая флаги доступными уже в следующем поле команды. Программа не может записать значение в регистр XSTAT или YSTAT явно в том же командном такте, в котором происходит обновление слова состояния.

Команды с множественными операндами (например, SFRs = Rmd + Rnd) порождают несколько результатов. В этом случае процессор вычисляет флаг состояния как битовое ИЛИ соответствующих флагов для каждого результата.

При использовании опции (NF) вычислительное устройство не обновляет значения флагов состояния в соответствии со статусом завершения операции. Разрешено использовать опцию (NF) со всеми командами ALU, за исключением команд PASS, COMP и FCOMP.

3.1.3.1 AZ - Признак равенства нулю результата в ALU

Флаг AZ устанавливается, если результат операции ALU равен нулю или в результате операции с плавающей точкой произошла потеря значимости.

Если бит UEN регистра X/YSTAT установлен, то операция, в которой произошла потеря значимости, (флаг AZ установлен в результате операции с плавающей точкой) порождает программное исключение.

3.1.3.2 AN - Признак отрицательного результата в ALU

Флаг AN устанавливается, если результат операции ALU отрицателен. Обычно флаг AN устанавливается равным MSB результата. По иным правилам флаг AN устанавливается только в случае команд, приведённых ниже:

- Rs = ABS Rm ; здесь AN равен знаку Rm (входных данных);
- FRs = ABS Rm; здесь AN равен знаку Rm (входных данных);
- Rs = ABS (Rm {+ | -} Rn) ; здесь AN равен знаку результата до применения операции ABS;
- Rs = MANT FRm; здесь AN равен знаку Rm (входных данных);
- FRs = ABS (Rm { + |-} Rn); здесь AN равен знаку результата до применения операции ABS.

Знак результата команд, приведённых выше, не указан, поскольку их результат всегда положителен.

3.1.3.3 AV - Признак переполнения в ALU

Флаг AV отражает состояние переполнения. Во всех командах ALU этот бит устанавливается, когда результат операции настолько велик, что не может быть представлен в требуемом формате. Проверка переполнения производится всегда в предположении, что результат знаковый, если иное не указано в команде.

В следующем примере значения в регистрах R5 и R6 равны 0x70...0 (большие положительные числа), результат команды сложения (0xD0..0) больше, чем позволяет формат данных для знаковых операций (MSB, который в дополнительном формате отражает знак, установлен).

```
R10 = R5 + R6 ;;
```

Как показано в следующем примере, команда может состоять более чем из одной операции:

```
R11:10 = EXPAND (Rm + Rn) ( I ) ;;
```

Если в этом примере сложения с расширением значения в Rm и Rn равны 0x70...0, то результатом этой операции будет 0x00...0D0...0. Переполнение определяется только для конечного результата и не вычисляется для промежуточных значений. В нашем случае переполнения нет.

Если бит OEN регистра X/YSTAT установлен, операция, вызвавшая переполнение (установлен флаг AV), порождает программное исключение.

3.1.3.4 AI - Признак некорректной операции в ALU

Флаг AI устанавливается в случае некорректной операции с плавающей точкой в соответствии со стандартом IEEE плавающей арифметики. Более подробную информацию можно найти в разделе «Формат данных с плавающей точкой одинарной точности IEEE».

Если бит IVEN регистра X/YSTAT установлен, некорректная операция плавающей арифметики (установлен флаг AI) порождает программное исключение.

3.1.3.5 AC - Признак переноса в ALU

Флаг AC используется как бит переноса в командах сложения или вычитания, которые могут быть сцеплены. Он также может быть использован как признак беззнакового переполнения в этих операциях. В случае знакового переполнения устанавливается флаг AV.

3.1.4 *ALU-предикаты*

В условных командах ALU исполнение всей командной строки зависит от выполнения предиката, указанного в начале командной этой строки. Условные команды ALU имеют вид:

IF cond; DO, instr; DO, instr; DO, instruct. ;;

Такой синтаксис позволяет подчинить одному предикату до трёх условных команд. Пропуск DO перед командой делает её безусловной.

В таблице 8 приведён список ALU-предикатов. Более подробную информацию об условных командах см. в разделе «Отработка условных команд».

Таблица 8 – ALU-предикаты

Предика	**Описание**	**Значения флагов**
AEQ	Равно нулю в ALU	AZ=1
ALT	Строго меньше нуля в ALU	AN=1
ALE	Меньше или равно нулю в ALU	AN=1илиAZ=1
NAEQ	НЕ (Равно нулю в ALU)	AZ=0
NALT	НЕ (Строго меньше нуля в ALU)	AN=0
NALE	НЕ (Меньше или равно нулю в ALU)	AN=0 и AZ=0

3.2 Статические флаги ALU

В устройстве управления регистр статических флагов (SFREG) может хранить значения флагов состояния для их дальнейшего использования в условных командах. В регистре SFREG каждому вычислительному блоку выделено два бита флагов: X/YSCF0 (предикаты XSF0, YSF0, XYSF0 или SF0) и X/YSCF1 (предикаты XSF1, YSF1, XYSF1 или SF1). Следующий пример демонстрирует загрузку значения предиката вычислительного блока в регистр статических флагов.

```
XSF0 = XAEQ ;;
/* Загрузить значение предиката AEQ вычислительного блока X в бит XSCF0 регистра статических флагов (SFREG) */
IF XSF0, DO XR5 = R4 + R3 ;;
/* предикат XSF0 истинен, если установлен статический флаг XSCF0 */
```

Более подробно о статических флагах рассказано в разделе «Отработка условных команд».

3.3 Примеры использования ALU

В листинге 1 приведено несколько примеров арифметических команд ALU. Ключевые особенности каждой команды, такие как формат операндов с фиксированной или плавающей точкой, размер операндов, а также использование регистров, перечислены в комментариях к команде.

Листинг 1 – Примеры команд ALU

```
LR5:4 = R11:10 + R1:0 ;;
/* Это операция попарного сложения 64-битных операндов с фиксированной точкой:
XR11:10 + XR1:0 и YR11:10 + YR1:0; результаты помещаются в XR5:4 и YR5:4
соответственно. */
YSR1:0 = R31:30 + R25:24 ;;
/* Это операция сложения четырёх 16-битных операндов с фиксированной точкой в
YR31:30 и четырёх операндов в YR25:24; четыре значения результата помещаются в
YR1:0. */
XR3 = R5 AND R7 ;;
/* Это операция логического и двух 32-битных операндов в XR5 и XR7; результат
помещается в XR3. */
YR4 = SUM SR3:2 ;;
/* Это операция знакового продольного суммирования четырёх 16-битных операндов
в YR3:2; результат помещается в YR4.*/
R9 = R4 + R8, R2 = R4 - R8 ;;
/* Это двойная команда (две команды в одном поле команды); первая часть команды
– это операция попарного сложения 32-битных операндов с фиксированной точкой:
XR4 + XR8 и YR4 + YR8; её результаты помещаются в XR9 и YR9 соответственно;
вторая часть - это операция попарного вычитания 32-битных операндов с
фиксированной точкой: XR4 - XR8 и YR4 - YR84; её результаты помещаются в XR2 и
YR2 соответственно. */
FR9 = R4 + R8 ;;
/* Это операция [попарного] сложения 32-битных операндов с плавающей точкой:
XR4 + XR8 и YR4 + YR8; результаты помещаются в XR9 и YR9 соответственно.*/
XFR9:8 = R3:2 + R5:4 ;;
/* Это команда сложения 40-битных (расширенное слово) операндов с плавающей
точкой в XR3:2 и XR5:4; результат помещается в XR9:8. */
```

3.3.1 *Пример параллельного сложения однобайтных данных*

На рисунке 41 показана операция сложения однобайтных операндов в двойных регистрах. Синтаксис для этой команды следующий:

```
XBR11:10 = R9:8 + R7:6 ;;
```

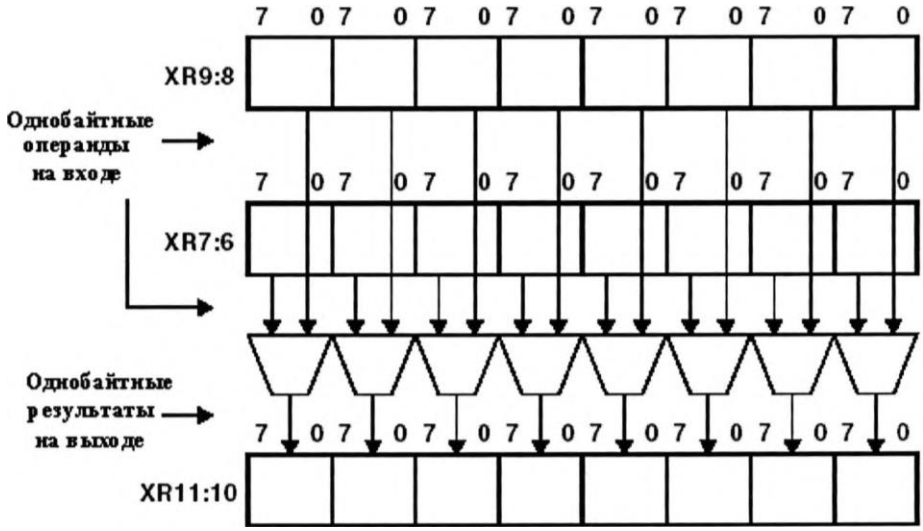


Рисунок 41 – Операнды для параллельного сложения

Важно отметить, что ALU обрабатывает все восемь операций сложения вне зависимости друг от друга, однако, вычисляет статус завершения всей операции путём битового ИЛИ соответствующих статусов каждой из операций.

3.3.2 *Пример продольного сложения однобайтных данных*

На рисунке 42 показана операция продольного сложения однобайтных операндов в двойных регистрах. Синтаксис для этой команды следующий:

XR11 = SUM BR9:8 (U);; /* беззнаковое продольное суммирование */

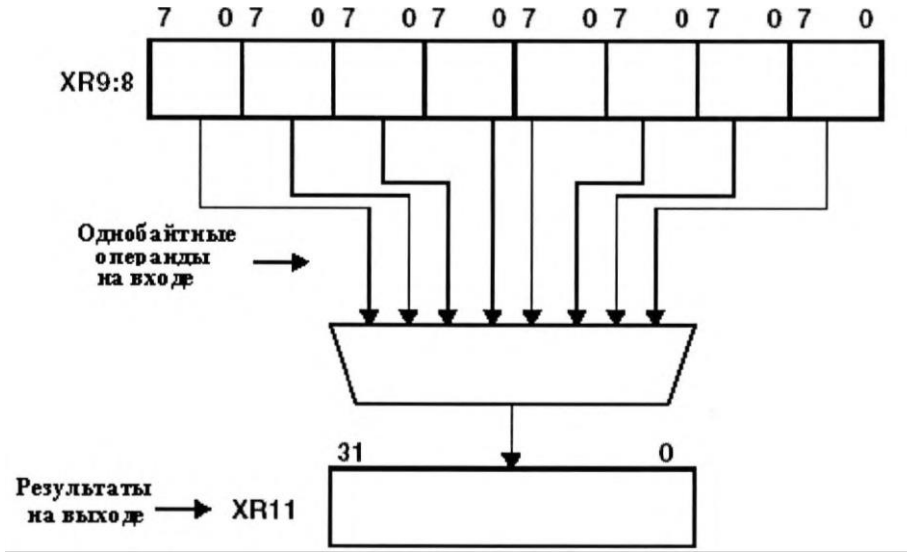


Рисунок 42 – Операнды для продольного сложения

3.3.3 *Пример использования регистра побочных результатов (PR)*

ALU имеет специальный набор команд, таких как SUM, VMAX, VMIN и ABS, которые могут использовать регистр PR1: 0. Регистр PR1: 0 является регистром ALU, не отображённым, подобно регистрам данных регистровых файлов, в память. Что касается его загрузки или сохранения, то регистр PR1: 0 может быть загружен в программе только из регистра данных или сохранён только в регистре данных; прямая

загрузка из памяти или сохранения в память регистра PR1: 0 невозможны. Для доступа к регистру PR1: 0 программа должна использовать следующие команды:

PR1:0 = Rmd ;;

Rsd = PR1:0 ;;

Команды загрузки или сохранения регистра PR всегда должны оперировать двойными регистрами, даже если используется только PRO или только PR1.

Команда SUM является одной из тех команд, которые могут использовать регистр PR1: 0 для хранения побочных результатов. При использовании регистра PR1:0 команда SUM выполняет операцию продольного сложения коротких или однобайтных операндов, добавляет полученный результат к содержимому одного из PR регистров и, наконец, сохраняет полученный побочный результат в этом регистре.

Следующая команда выполняет четыре 16-битных сложения и прибавляет результат к текущему содержимому регистра PR0.

PR0 += SUM SR5:4 ;;

На рисунке 43 показано использование регистров при параллельном/продольном сложении.

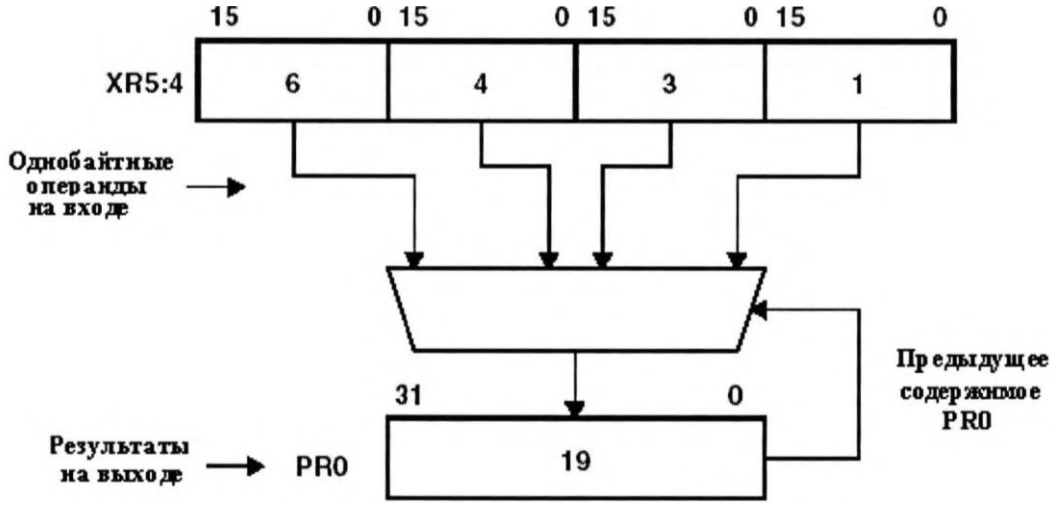


Рисунок 43 – Операнды для параллельного/продольного сложения

3.4 Сводка команд ALU

Следующие листинги демонстрируют синтаксис команд ALU:

- Листинг 2 «Команды ALU операций с фиксированной точкой»;
- Листинг 3 «Команды ALU логических операций»;
- Листинг 4«Различные команды ALU операций с фиксированной точкой»;
- Листинг 5 «Команды ALU операций с плавающей точкой»;
- Листинг 6 «Команды ALU векторных операций с плавающей точкой (вариант 1)» (данный синтаксис считается устаревшим);
- Листинг 7 «Команды ALU векторных операций с плавающей точкой (вариант 2)».

Соглашения по обозначениям имён регистров, необязательных элементов или вариантов, используемые в этих листингах, подробно описаны в разделе «Регистры регистровых файлов». Вкратце эти соглашения таковы:

- { } – необязательные элементы указаны в фигурных скобках; фигурные скобки не являются частью синтаксиса команды;
- | – вертикальная черта разделяет варианты значений элементов; вертикальные черты не являются частью синтаксиса команды;

- Rmd – имена регистров, выделенные курсивом, представляют собой подставляемые шаблоны для имён одиночных (Rs, Rm или Rn), двойных (Rsd, Rmd или Rnd) или квадро- (Rsq, Rmq или Rnq) регистров.

Каждая команда, представленная здесь, занимает одно поле команды в командной строке. Более подробную информацию о командных строках и ограничениях по комбинированию команд см. в разделах «Синтаксис и структура командной строки» и «Правила запараллеливания команд».

Листинг 2 – Команды ALU операций с фиксированной точкой

{X|Y|XY} {S|B}Rs = Rm +|- Rn {({S|SU} {NF})} ;[1]
{X|Y|XY} {L|S|B}Rsd = Rmd +|- Rnd {({S|SU} {NF})} ;[1]
{X|Y|XY} Rs = Rm + CI {-1} ;
{X|Y|XY} LRsd = Rmd + CI {-1} ;
{X|Y|XY} Rs = Rm + Rn + CI {({S|SU} {NF})} ;[1]
{X|Y|XY} Rs = Rm - Rn + CI -1 {({S|SU} {NF})} ; [1]
{X|Y|XY} LRsd = Rmd + Rnd + CI {({S | SU} {NF}) } ;[1]
{X|Y|XY} LRsd = Rmd - Rnd + CI -1 {({S | SU} {NF})} ; [2]
{X|Y|XY} {S | B}Rs = (Rm+|-Rn)/2 {({T} {U}{NF})} ; [2]
{X|Y|XY} {L | S | B} Rsd = (Rmd +|- Rnd)/2 { ({T} {U} {NF}) } ; [2]
{X|Y|XY} {S | B}Rs = ABS Rm { (NF) } ;
{X|Y|XY} {L|S|B}Rsd = ABS Rmd {(NF)} ;
{X|Y|XY} {S | B} = ABS (Rm + Rn) { ({X} {NF}) } ; [3]
{X|Y|XY} { L | S | B} Rsd = ABS (Rmd + Rnd) { ({X} {NF}) } ; [3]
{X|Y|XY} { S | B} = ABS (Rm - Rn) { ({X | U) {NF}) } ; [4]
{X|Y|XY} { L | S | B} Rsd = ABS (Rmd -Rnd) { ({X | U} {NF}) } ; [4]
{X|Y|XY} {S | B} Rs = - Rm {(NF)} ;
{X|Y|XY} {L|S|B}Rsd = - Rmd {(NF)} ;
{X|Y|XY} {S | B} Rs = MAX | MIN (Rm, Rn) { ({U} { Z } {NF}) } ; [5]
{X|Y|XY} { L | S | B}Rsd = MAX |MIN (Rmd, Rnd) { ({U} { Z } {NF}) } ; [5]
{X|Y|XY} S | BRsd = VMAX | VMIN (Rmd, Rnd) { (NF) } ;
{X|Y|XY} {S|B}Rs = INC | DEC Rm {({S | SU}{NF})} ; [1]
{X|Y|XY} {L|S|B}Rsd = INC |DEC Rmd {({S | SU}{NF})) ; [1]
{X|Y|XY} {S | B} COMP (Rm, Rn) { (U) } ; [5]
{X|Y|XY} {L|S|B}COMP(Rnd, Rnd) {(U)) ; [5]
{X|Y|XY} {S |B}Rs = CLIP Rm BY Rn { (NF) } ;
{X|Y|XY} {L|S|B}Rsd = CLIP Rmd BY Rnd {(NF)} ;
{X|Y|XY} Rs = SUM SRm | BRm {({U}{NF})} ; [6]
{X|Y|XY} Rs = SUM SRmd | BRmd {({U}{NF})} ; [6]

[1] Допустимые опции:( ) без насыщения, (S): с насыщением, знаковое, (SU): с насыщением, беззнаковое.
[2] Допустимые опции: ( ) округление к ближайшему четному, (T): знаковое, с усечением, (U): беззнаковое, округление к ближайшему чётному, (TU): беззнаковое, с усечением
[3] Допустимые опции:( ): знаковое, с насыщением, (X): расширенное для ABS с беззнаковым насыщением.
[4] Допустимые опции:( ): знаковое, с насыщением, (X): расширенное для ABS с беззнаковым насыщением, (U): беззнаковое.
[5] Допустимые опции: ( ): обычное знаковое сравнение, (U): сравнение беззнаковых чисел, (Z): результат равен нулю, если выбран Rn, в операциях MIN/MAX; иначе результат равен Rm, (UZ): беззнаковое сравнение с опцией (Z) описанной выше.
[6] Допустимые опции: ( ): знаковое, (U): беззнаковое.

{X|Y|XY} Rs = ONES Rm | Rmd {(NF)} ;
{X|Y|XY} PR1:0 = Rmd {(NF)} ;
{X|Y|XY} Rsd = PR1:0 {(NF)) ;
{X|Y|XY} Rs = BFOINC Rmd { (NF) } ;
{X|Y|XY} PRO|PR1 +=ABS(SRmd - SRnd) {({U}{NF})}
{X|Y|XY} PRO|PR1 +=ABS(BRmd - BRnd) {({U}{NF})}
{X|Y|XY} PRO|PR1 +=SUM SRm {({U}{NF})} ; [6]
{X|Y|XY} PRO|PR1 +=SUM SRmd {({U}{NF})} ; [6]
{X|Y|XY} PRO|PR1 +=SUM BRm {({U}{NF})} ; [6]
{X|Y|XY} PRO|PR1 +=SUM BRmd {({U}{NF})} ; [6]
{X|Y|XY} {S|B} Rs = Rm + Rn, Ra= Rm - Rn{ (NF) } ; (двойная операция)
{X|Y|XY} {L|S|B}Rsd = Rmd + Rnd, Rad = Rmd -Rnd { (NF)}; (двойная операция)
{X|Y|XY} {B|S}Rsd = COMPACT {S}Rm by <Imm> {({U}{R})};
{X|Y|XY} {B|S|L}Rsq = COMPACT {S|L}Rmd by <Imm> {({U}{R})};
{X|Y|XY} {B|S}Rsd = EXPAND {B|S}Rm - {B|S}Rn (ZF);
{X|Y|XY} {B|S|L}Rsq = EXPAND {B|S}Rm - {B|S}Rn (ZF);
{X|Y|XY} {S|B}Rs = CLIPZ Rm by Rn;
{X|Y|XY} {L|S|B}Rsd = CLIPZ Rmd by Rnd;
{X|Y|XY} {B|S} Rs = ROUNDUP (Rm +|- Rn)/2 {(U|NF)}
{X|Y|XY} {L|S|B} Rsd = ROUNDUP (Rmd +|- Rnd)/2 {(U|NF)}
{X|Y|XY} {S|B} Rs = SETMASK Rm {EQ}|{GE}|{GT} Rn {(U|NF)};
{X|Y|XY} {S|B|L} Rsd = SETMASK Rm {EQ}|{GE}|{GT} Rn {(U|NF)};

Листинг 3 – Команды ALU логических операций

{X|Y|XY}*Rs* = PASS *Rm* ;
{X|Y|XY}LRsd = PASS Rmd ;
{X|Y|XY}*Rs* = *Rm* AND | AND NOT | OR | XOR *Rn* {(NF)} ;
{X|Y|XY}LRsd = *Rmd* AND | AND NOT | OR | XOR *Rnd* { (NF) } ;
{X|Y|XY}*Rs* = NOT *Rm* {(NF)} ;
{X|Y|XY}*LRsd* = NOT *Rmd* {(NF)} ;

Листинг 4 – Различные команды ALU операций с фиксированной точкой

```
{X|Y|XY}Rsd = EXPAND SRm {+|- SRn} {({I | IU} {NF})} ;[1]
{X|Y|XY}Rsq = EXPAND SRmd { + |- SRnd} {({I | IU}{NF)) } ;[1]
{X|Y|XY}SRsd = EXPAND BRm {+|- BRn} (((I|IU}{NF))} ;[1]
{X|Y|XY}SRsq = EXPAND BRmd {+|- BRnd} {({I|IU)(NF))) ;[1]
{X|Y|XY}SRs = COMPACT Rmd {+|- Rnd} {({T | I | IS | ISU}{NF})} ;[2]
{X|Y|XY}BRs = COMPACT SRmd {+|- SRnd} {({T | I | IS | ISU}{NF})} ;[2]
{X|Y|XY}Rs = COMPACT LRmd {({U | IS | ISU}{NF})} ;
{X|Y|XY}LRsd = COMPACT QRmq {({U | IS | ISU}{NF})} ;
{X|Y|XY}BRsd = MERGE Rm, Rn {(NF)} ;
{X|Y|XY}BRsq = MERGE Rmd, Rnd ( (NF) } ;
{X|Y|XY}SRsd = MERGE Rm, Rn {(NF)} ;
{X|Y|XY}SRsq = MERGE Rmd, Rnd {(NF)} ;
{X|Y|XY}Rsd = PERMUTE (Rmd, Rn) {(NF)} ;
{X|Y|XY}Rsq = PERMUTE (Rmd, -Rmd, Rn) {(NF)} ;
```

Листинг 5 – Команды ALU операций с плавающей точкой

```
{X|Y|XY}FRs = Rm +|- Rn {({T}{NF})} ;[3]
{X|Y|XY}FRsd = Rmd +|- Rnd {({T}{NF})} ;[3]
{X|Y|XY}FRs = (Rm +|- Rn)/2 {({T} {NF}) } ;[3]
{X|Y|XY}FRsd = (Rmd +|- Rnd)/2 {({T} {NF} ) } ;[3]
{X|Y|XY}FRs = MAX | MIN (Rm, Rn) { (NF) } ;[4]
{X|Y|XY}FRsd = MAX | MIN (Rmd, Rnd) { (NF) } ;[4]
{X|Y|XY}FRs = ABS Rm {(NF)} ;
{X|Y|XY}FRsd = ABS Rmd { (NF) } ;
{X|Y|XY}FRs = ABS (Rm +|- Rn) {({T}{NF})} ;[3]
{X|Y|XY}FRsd = ABS (Rmd +|- Rnd) {({T} {NF} ) } ;[3]
{X|Y|XY}FRs = - Rm { (NF) } ;
{X|Y|XY}FRsd = - Rmd {(NF)} ;
{X|Y|XY}FCOMP (Rm, Rn) ;
{X|Y|XY}FCOMP (Rmd, Rnd) ;
{X|Y|XY}Rs = FIX FRm|FRmd {BY Rn} {({T}{NF})} ;
{X|Y|XY}FRs | FRsd = FLOAT Rm {BY Rn} { ( {T} {NF} ) } ;
{X|Y|XY}FRsd = EXTD Rm { (NF) } ;
{X|Y|XY}FRs = SNGL Rmd {({T}{NF})} ;
{X|Y|XY}FRs = CLIP Rm BY Rn { (NF) } ;
{X|Y|XY}FRsd = CLIP Rmd BY Rnd {(NF)} ;
{X|Y|XY}FRs = Rm COPYSIGN Rn {(NF)} ;
{X|Y|XY}FRsd = Rmd COPYSIGN Rnd { (NF) } ;
{X|Y|XY}FRs = SCALB FRm BY Rn {(NF)} ;
{X|Y|XY}FRsd = SCALB FRmd BY Rn {(NF)} ;
{X|Y|XY}FRs = PASS Rm ;
{X|Y|XY}FRsd = PASS Rmd ;
{X|Y|XY}FRs = RECIPS Rm {(NF)} ;
{X|Y|XY}FRsd = RECIPS Rmd {(NF)} ;
{X|Y|XY}FRs = RSQRTS Rm {(NF)} ;
{X|Y|XY}FRsd = RSQRTS Rmd {(NF)} ;
```

[1] Допустимые опции: ( ): дробное, (I): знаковое целое, (IU): беззнаковое целое.
[2] Допустимые опции: ( ): дробное округление, (I): целое, без насыщения, (T): дробное, с усечением, (IS): целое, с насыщением, знаковое, (ISU): целое, с насыщением, беззнаковое.
[3] Допустимые опции: ( ): округление, (T): с усечением.
[4] Допустимые опции: ( ): округление, (T): с усечением (только MIN).

{X|Y|XY}Rs = MANT FRm | FRmd {(NF)} ;
{X|Y|XY}Rs = LOGB FRm | FRmd {({S}{NF})}
{X|Y|XY}FRs = Rm + Rn, FRa = Rm - Rn {(NF)} ; (двойная операция)
{X|Y|XY}FRsd = Rmd + Rnd, FRad = Rmd - Rnd { (NF) } ; (двойная операция)

Листинг 6 – Команды ALU векторных операций с плавающей точкой (вариант 1)

{X|Y|XY}FRsd = RmH + RnH, RmL + RnL {({T}{NF})} ;
{X|Y|XY}FRsd = RmH - RnH, RmL – RnL {({T}{NF})} ;
{X|Y|XY}FRsd = (RmH + RnH) / 2, (RmL + RnL) / 2 {({T}{NF})} ;
{X|Y|XY}FRsd = (RmH - RnH) / 2, (RmL - RnL) / 2 {({T}{NF})} ;
{X|Y|XY}FRsd = ABS (RmH + RnH), ABS (RmL + RnL) {({T}{NF})} ;
{X|Y|XY}FRsd = ABS (RmH - RnH), ABS (RmL - RnL) {({T}{NF})} ;
{X|Y|XY}FRsd = FLOAT RmH by Rn, FLOAT RmL by Rn {({T}{NF})} ;
{X|Y|XY}Rsd = FIX FRmH by Rn, FIX FRmL by Rn {({T}{NF})} ;
{X|Y|XY}FRsd = CLIP RmH by RnH, CLIP RmL by RnL {({T}{NF})} ;
{X|Y|XY}FRsd = RmH COPYSIGN RnH, RmL COPYSIGN RnL {({T}{NF})} ;
{X|Y|XY}FRsd = SCALB FRmL BY Rn, SCALB FRmH BY Rn {({T}{NF})} ;
{X|Y|XY}FRsd = FLOAT RnL, FLOAT RmH {({T}{NF})} ;
{X|Y|XY}FRsd = ABS (RmH), ABS(RmL) {({T}{NF})} ;
{X|Y|XY}Rsd = MANT FRmH, MANT FRmL {({T}{NF})} ;
{X|Y|XY}FRsd = PASS RmH, PASS RmL {({T}{NF})} ;
{X|Y|XY}FRsd = -RmH, -RmL {({T}{NF})} ;
{X|Y|XY}FRsd = RECIPS RmH, RECIPS RmL {({T}{NF})} ;
{X|Y|XY}FRsd = RSQRTS RmH, RSQRTS RmL {({T}{NF})} ;
{X|Y|XY}Rsd = FIX FRmH, FIX FRmL {({T}{NF})} ;
{X|Y|XY}Rsd = LOGB FRmH, LOGB FRmL {({T}{S}{NF})} ;
{X|Y|XY}FRsd = MAX(RmH, RnH), MAX(RmL RnL) {({T}{NF})} ;
{X|Y|XY}FRsd = MIN(RmH, RnH), MIN(RmL RnL) {({T}{NF})} ;
{X|Y|XY}FCOMP (RmH, RnH), FCOMP(RmL, RnL) {({T}{NF})} ;
{X|Y|XY}FRsd = RmH + RnH, RmL + RnL, FRad = RmH - RnH, RmL - RnL {(NF)} ;

T: с округлением (T=0); с усечением (T=1) для всех команд ALU с плавающей точкой;
S задает насыщение (только LOGB).

Листинг 7 – Команды ALU векторных операций с плавающей точкой (вариант 2)

{X|Y|XY}SFRsd = Rmd +|- Rnd {({mX}{nX} {T}{NF})} ;
{X|Y|XY}SFRsd = (Rmd +|- Rnd)/2 {({mX}{nX} {T} {NF}) } ;
{X|Y|XY}SFRsd = MAX | MIN (Rmd, Rnd) {({mX}{nX} {NF}) } ;
{X|Y|XY}SFRsd = ABS Rmd {({mX}{NF}) };
{X|Y|XY}SFRsd = ABS (Rmd +|- Rnd) {({mX}{nX} {T} {NF}) } ;
{X|Y|XY}SFRsd = - Rmd {({mX} {NF}) };
{X|Y|XY}SFCOMP (Rmd, Rnd) {({mX}{nX}} ;
{X|Y|XY}SFRsd = CLIP Rmd BY Rnd {({mX}{nX} {NF}) };
{X|Y|XY}SFRsd = Rmd COPYSIGN Rnd {({mX}{nX} {NF}) };
{X|Y|XY}SFRsd = SCALB FRmd BY Rn {({mX} {NF}) };
{X|Y|XY}SFRsd = PASS Rmd {({mX}};
{X|Y|XY}SFRsd = RECIPS Rmd {({mX}{NF}) };
{X|Y|XY}SFRsd = RSQRTS Rmd {({mX}{NF}) };
{X|Y|XY}SFRsd = FLOAT Rmd BY RN {({mX}{T}{NF})};
{X|Y|XY}Rsd = FIX SFRmd BY Rn {({mX}{T}{NF})};
{X|Y|XY}SFRsd = FLOAT Rmd {({mX}{T}{NF})};

{X|Y|XY}Rsd = MANT SFRmd {({mX}{NF}) };
{X|Y|XY}Rsd = FIX SFRmd {({mX}{T}{NF})};
{X|Y|XY}Rsd = LOGB SFRmd {({mX} {S}{NF})};
{X|Y|XY}SFRsd = Rmd + Rnd, SFRd = Rmd - Rnd {({mX}{nX} {NF}) }; (двойная операция)

T: с округлением (T=0); с усечением (T=1) для всех команд ALU с плавающей точкой;
S задает насыщение (только LOGB).

4 Коммуникационно-логическое устройство (CLU)

Ядро процессора содержит два вычислительных устройства, известных как вычислительные блоки. Каждый вычислительный блок содержит регистровый файл и четыре независимых вычислительных устройства: ALU, CLU, умножитель и сдвиговое устройство. Коммуникационно-логическое устройство (CLU) выделено на рисунке 44. CLU берёт входные данные из регистрового файла или регистров TR/THR и сохраняет выходные данные также в регистровом файле или регистрах TR/THR. Большая часть команд CLU работает с регистрами решётчатых операций (TR) и регистрами истории решётчатых операций (THR).

Данное устройство выполняет специализированные функции, облегчающие написание основных телекоммуникационных алгоритмов, таких как декодирование по Витерби, декодирование турбо кодов, декодирование в системах множественного доступа с кодовым разделением каналов (CDMA), операции сжатия частотной полосы сигнала и нахождения комплексной корреляции. Эти функции могут быть также применены и в алгоритмах, не используемых в телекоммуникациях.

Глава содержит следующие разделы:

- Операции CLU;
- Примеры использования CLU;
- Сводка команд CLU.

Включение команд CLU в код упрощает программирование упомянутых алгоритмов, сохраняя при этом всю гибкость чисто программного подхода. При таком подходе легко подстраивать алгоритм под индивидуальные требования конкретного пользователя. К тому же, эти команды могут использоваться во множестве различных приложений: например, команда TMAX, включенная в набор команд для декодирования турбо кодов, также оказывается очень полезной при декодировании кодов контроля чётности с низкой плотностью.

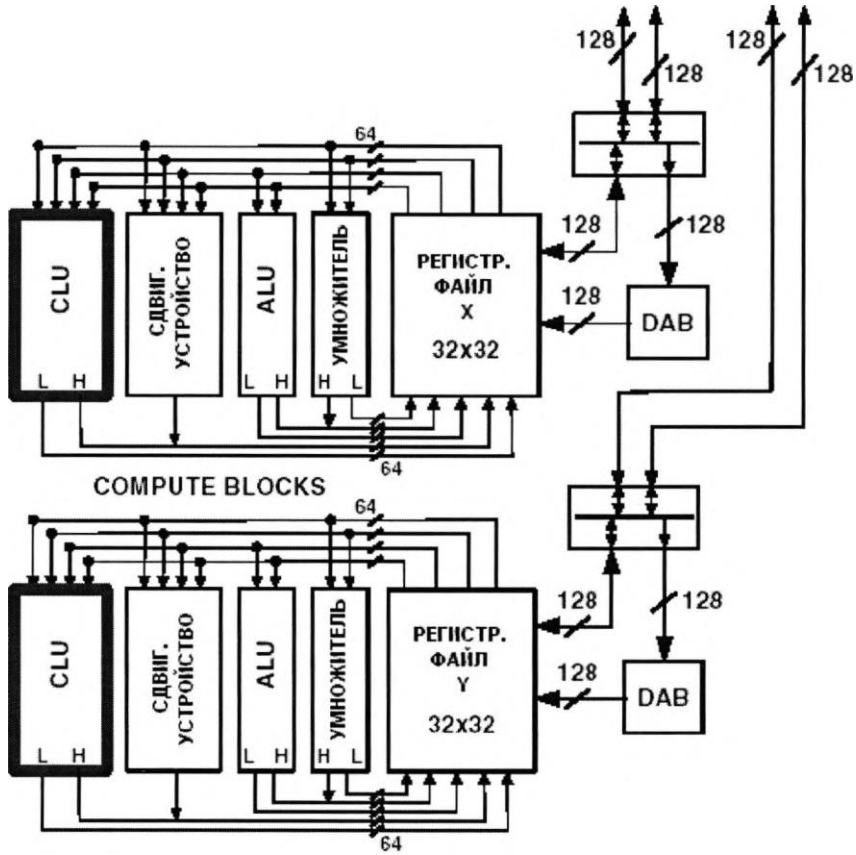


Рисунок 44 – CLU вычислительных блоков X и Y

Главным достоинством процессора является огромная скорость передачи данных, которая составляет два 128-битных обращения к памяти на каждом такте. Для операции сжатия частотной полосы сигнала это позволяет выполнять 16 комплексных операций умножения с накоплением над 16-битными комплексными данными (с 8-битной действительной и 8-битной мнимой частями) на каждом такте. Также это позволяет обсчитывать полную 16-битную решётку с 64-мя состояниями на обоих вычислительных блоках совместно каждые четыре такта.

Операции CLU производятся над данными в формате с фиксированной точкой. Соотнеся операции CLU и поддерживаемые ими действительные и комплексные типы данных, можно увидеть, что 128-битное устройство CLU в каждом вычислительном блоке реализует следующие функции:

- логарифм Якоби для декодирования турбо кодов (TMAX);
- сжатие частотной полосы сигнала в CDMA (DESPREAD);
- вычисление кросс-корреляции в CDMA (XCORRS);
- полиномиальное переупорядочивание (PERMUTE);
- решётчатая операция сложить/сравнить/выбрать (ACS).

Просматривая возможные операнды каждой команды, можно увидеть, какие операции CLU какие типы данных поддерживают:

- 32-битные комплексные данные (16-битная действительная и 16-битная мнимая части);
- 16-битные комплексные данные (8-битная действительная и 8-битная мнимая части);
- 2-битные комплексные данные (1-битная действительная и 1-битная мнимая части);
- 8- и 16-битные действительные данные.

Поддержка определённого типа данных в командах CLU реализована согласно требованиям соответствующего алгоритма. Большая часть команд не поддерживает все типы и размеры данных. Правила именования регистров определяют размер и тип данных входных операндов и результата в командах. Более подробную информацию о выборе размера и типа данных в командах CLU см. в разделе «Регистры регистровых файлов».

Оставшаяся часть этой главы посвящена описанию команд CLU, их опций и результатов с использованием синтаксиса команд. Объяснение соглашений по синтаксису команд CLU и других устройств см. в подразделе «Синтаксис и структура командной строки». Список команд CLU и их синтаксиса см. в подразделе «Сводка команд CLU».

4.1 Операции CLU

CLU выполняет коммуникационно-логические операции над данными в формате с фиксированной точкой. Процессор использует регистры вычислительных блоков для хранения входных операндов и результата операций CLU. Команды CLU оперируют тремя типами регистров:

- Rm, n, s – регистры (данных) регистрового файла;
- TRm, n, s – 32 регистра (решётчатых операций), доступных только для команд CLU;
- THR – четыре регистра (истории решётчатых операций), используемых в командах;
- ACS, DESPREAD и XCORRS для сдвигаемых данных;
- CMCTL – коммуникационный управляющий регистр, используемый в команде;
- XCORRS в качестве дополнительного источника для значения CUT.

Далее в разделе описаны:

- Функция TMAX;
- Решётчатые функции;
- Функции сжатия частотной полосы сигнала;
- Функция вычисления кросс-корреляций;
- Функция сложить/сравнить/выбрать;
- Опции команд CLU;
- Статус выполнения CLU.

Для операций декодирования турбо кодов и декодирования по Витерби CLU поддерживается размер входных данных в 8 и 16 бит для мягких значений (мягкие значения можно понимать, как вероятности значений бита, в отличие от жестких значений - 0 или 1); для выходных данных поддерживается размер в 16 и 32 бита. Необходимо внимательно выбирать размер данных, чтобы не столкнуться с ситуацией переполнения во время вычислений.

Функция DESPREAD работает с 16-битными комплексными числами. Каждое 16-битное комплексное число состоит из действительной части (биты 7-0) и мнимой части (биты 15-8). В результате всегда получается одно или два комплексных слова, каждое из которых состоит из двух коротких слов. В них биты 15-0 представляют действительную часть, а биты 31-16 – мнимую часть (аналогично представлению комплексных чисел в умножителе).

Все команды CLU устанавливают флаги состояния для отображения статуса завершения операции. Так как в одной команде может быть проведено множество операций над множеством операндов, значение флага состояния вычисляется как битовое ИЛИ соответствующих флагов всех операций. Более подробная информация о состоянии CLU представлена в разделе «Статус выполнения CLU».

4.1.1 *Функция TMAX*

Функция TMAX обычно используется при декодировании турбо кодов и других высокопроизводительных кодов, исправляющих ошибки. Функция определяется следующим образом:

TMAX(a,b) = max (a,b) + ln(1+e-|a-b|)

где второе слагаемое реализовано в виде таблицы:

ln(1+e-|a-b|)

(для больших значений |a-b|, его значение близко к ln (1) = 0).

Результат вычитания подаётся на вход таблицы TMAX по высокому уровню тактового сигнала на стадии конвейера EX1. Отрицательный результат инвертируется (в предположении, что разница между дополнением до единицы и дополнением до двух в пределах допустимой погрешности). Входным значением для таблицы 9 являются семь LSB результата операции вычитания со сравнением. Если результат операции вычитания со сравнением занимает больше семи бит, результат считается нулевым. Обратите внимание, что подразумеваемая десятичная точка всегда расположена перед пятью LSB.

Ниже (таблица 9) показаны значения TMAX. Максимальная погрешность выходного значения один LSB.

Таблица 9 – Значения TMAX

Отрицательное входное значение (двоичное)	**Положительное входное значение (двоичное)**	**Выходное значение (двоичное)**
11..111.1111X	000.0000x	000.10110
11..111.1110x	000.0001X	000.10101
11..111.1101X	000.0010x	000.10100
11..111.1100x	000.0011X	000.10011
11..111.1011X	000.0100X	000.10010
11..111.1010x	000.0101X	000.10001
11..111.100xX	000.011XX	000.10000
11..111.011XX	000.100XX	000.01110
11..111.010xX	000.101XX	000.01101
11..111.001XX	000.110xX	000.01100
11..111.000xX	000.111XX	000.01011
11..110.111XX	001.000xX	000.01010
11..110.110xX	001.001XX	000.01000
11..110.10xXX	001.01XXX	000.00111
11..110.01XXX	001.10xXX	000.00110
11..101.00XXX	001.11XXX	000.00101
11..101.1XXXX	010.0xXXX	000.00011
11..101.0xXXX	010.1XXXX	000.00010
11..100.XXXXX	011.XXXXX	000.00001
<= 11..10xX.XXXXX	>= 1XX.XXXXX	000.00000

4.1.2 *Решётчатые функции*

Решётчатые диаграммы (рисунок 45) являются широко используемым инструментом в коммуникационных системах. Например, оба алгоритма декодирования по Витерби и декодирования турбо кодов работают с решётками.

Процессор предоставляет специализированные команды для операций над решётками с количеством состояний, не превышающим восьми, и бинарными переходами между состояниями. Решётки с большим количеством состояний часто могут быть разбиты на подрешётки с восемью или менее состояниями и затем обработаны при помощи этих команд.

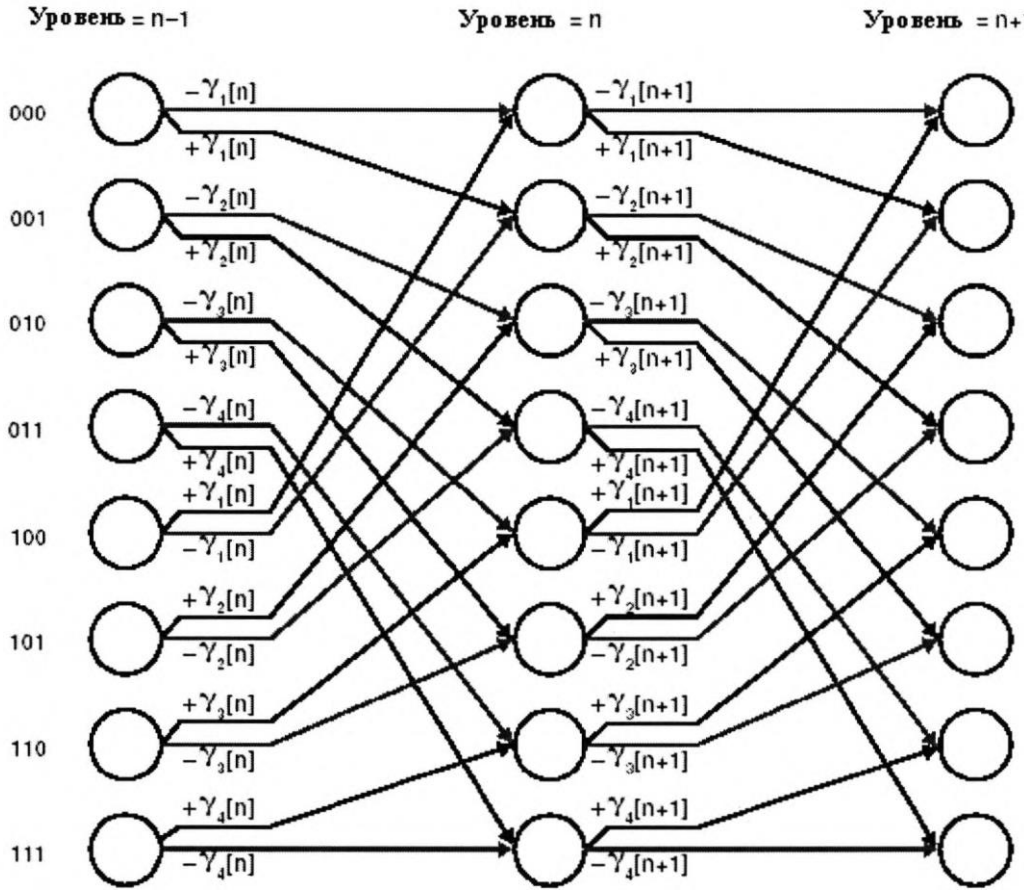


Рисунок 45 – Решётчатая диаграмма

Типичная решётчатая диаграмма (изображённая на рисунке 45) представляет множество возможных переходов между состояниями с одного уровня на другой.

На каждом уровне n требуется вычислить суммарную метрику для каждого состояния. Для конкретного состояния это значение зависит от метрики каждого из двух возможных предшествующих ему состояний (на уровне n-1) и от метрики перехода γ [n] (gamma [n]), соответствующей определенным комбинациям на входе/выходе.

Определённая симметрия в метриках переходов, изображённая на рисунке 45 типична для кодов, исправляющих ошибки, используемых на практике.

Команда ACS (сложить-сравнить-выбрать) имеет варианты для вычисления двух типов метрик состояний. В алгоритме Витерби метрики каждого из двух предшествующих состояний пересчитываются и из них выбирается максимальное значение. Например, метрика для состояния "100" (двоичная запись 4) на уровне n вычисляется следующим образом:

Метрика(“ 100”,п) = max ((Метрика(“010”,п-1)- γ3(п)), (Метрика(“ 110”,п-1)+ γ3(п))

Команда ACS параллельно вычисляет метрики для четырёх или восьми состояний, и дополнительно сохраняет информацию о предпринятых переходах для использования в процедуре прослеживания предшествующих переходов.

Во втором варианте, используемом при декодировании турбо кодов, операция MAX в описанном выше алгоритме заменяется на операцию TMAX, определённую в подразделе «Функция TMAX».

4.1.2.1 Решётчатые функции вида STRsd = TMAX(TRmd+Rmq_h,TRnd+Rmq_l);

Как показано на рисунках 46 и 47, старшая часть регистра Rmq прибавляется к значению в TRmd, младшая часть регистра Rmq прибавляется к значению в TRnd, а затем выполняется операция TMAX над результатами обеих операций сложения.

В данной команде поддерживается режим насыщения.

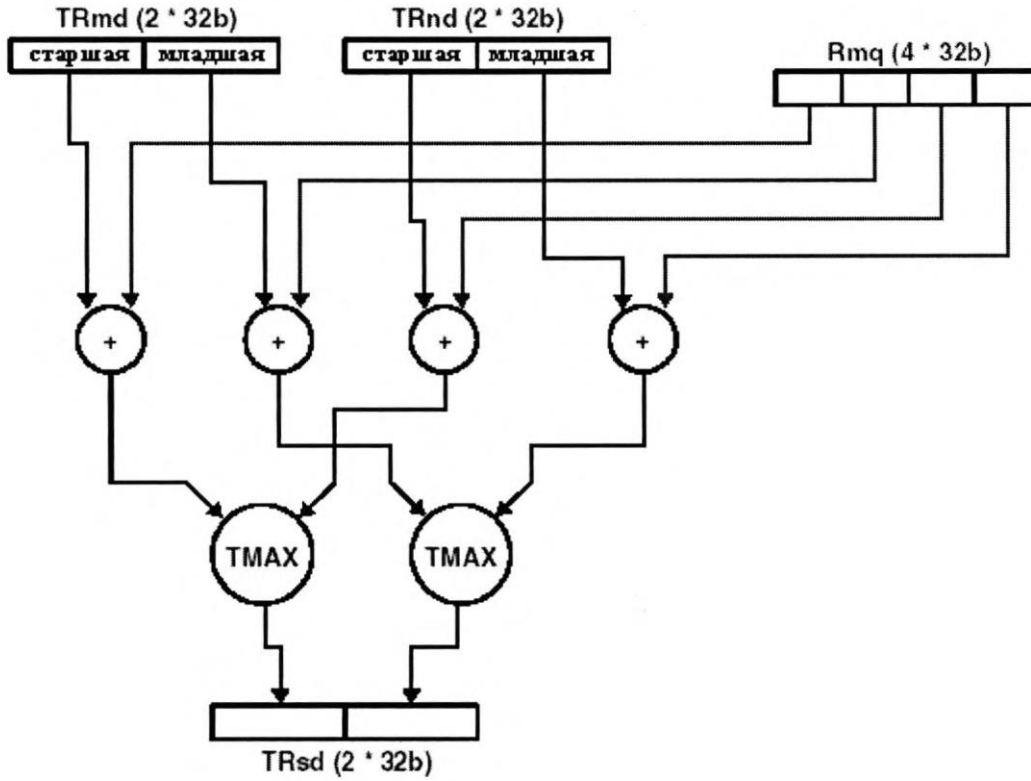


Рисунок 46 – TRsd = TMAX(TRmd + Rmq_h, TRnd + Rmq_l)

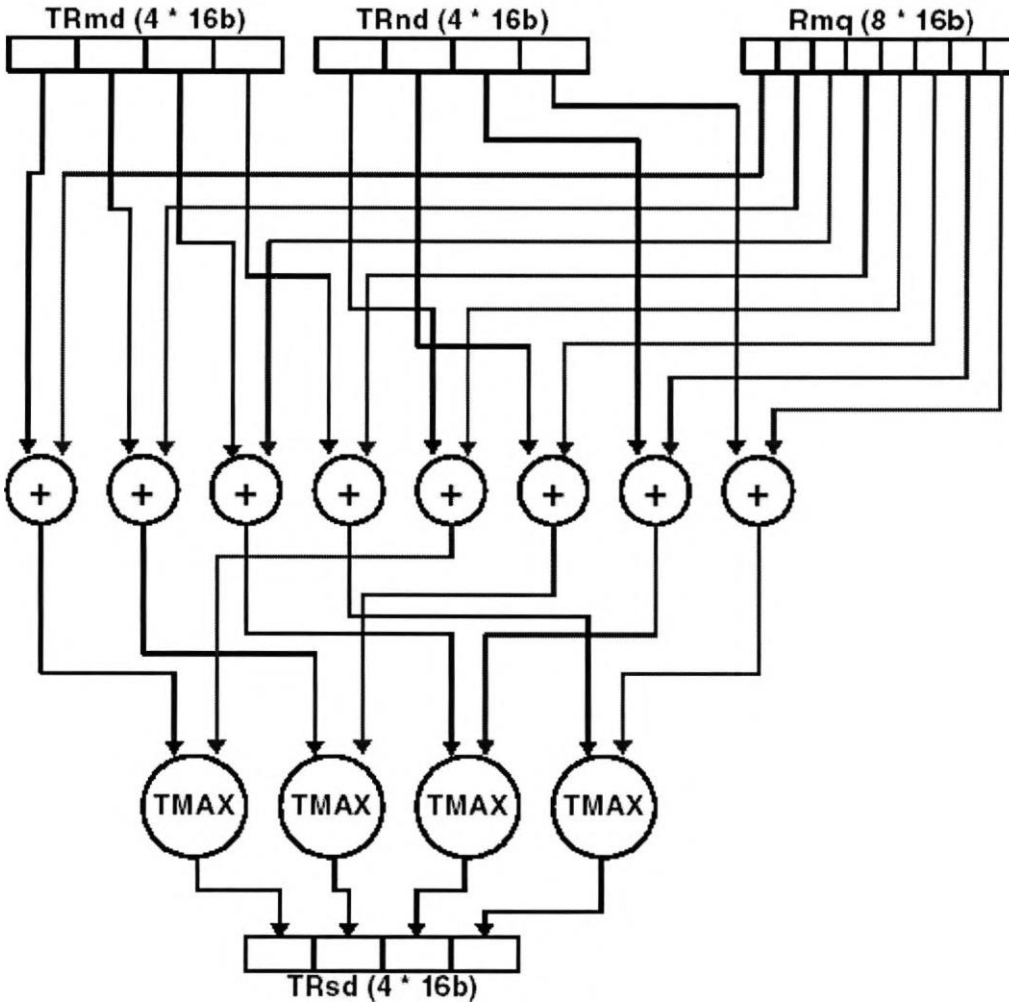


Рисунок 47 – STRsd = TMAX(TRmd + Rmq_h, TRnd + RMq_l)

4.1.2.2 Решётчатые функции вида STRsd = TMAX(TRmd-Rmq_h,TRnd-Rmq_l);

Как показано на рисунках 49 и 50, старшая часть регистра Rmq вычитается из значения в TRmd, младшая часть регистра Rmq вычитается из значения в TRnd, а затем выполняется операция TMAX над результатами обеих операций вычитания. Порядок операндов в операциях вычитания представлен на рисунке 48.

В данной команде поддерживается режим насыщения.

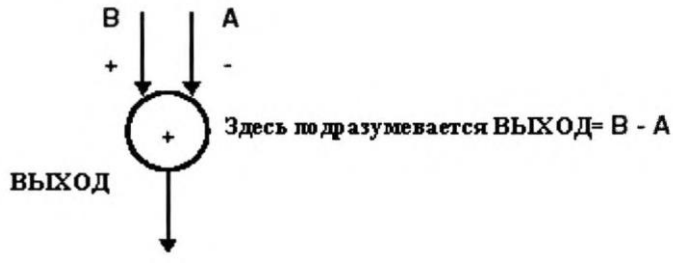


Рисунок 48 – Порядок операндов для диаграмм с вычитанием

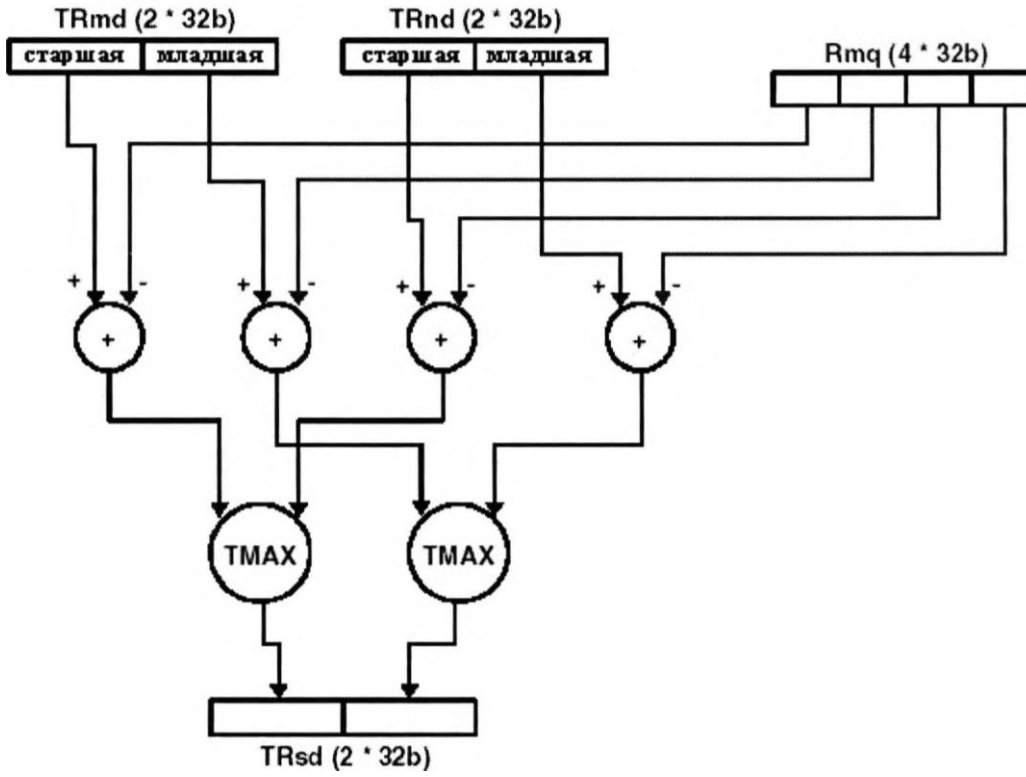


Рисунок 49 – TRsd = TMAX (TRmd - Rmq_h, TRnd - Rmq_l)

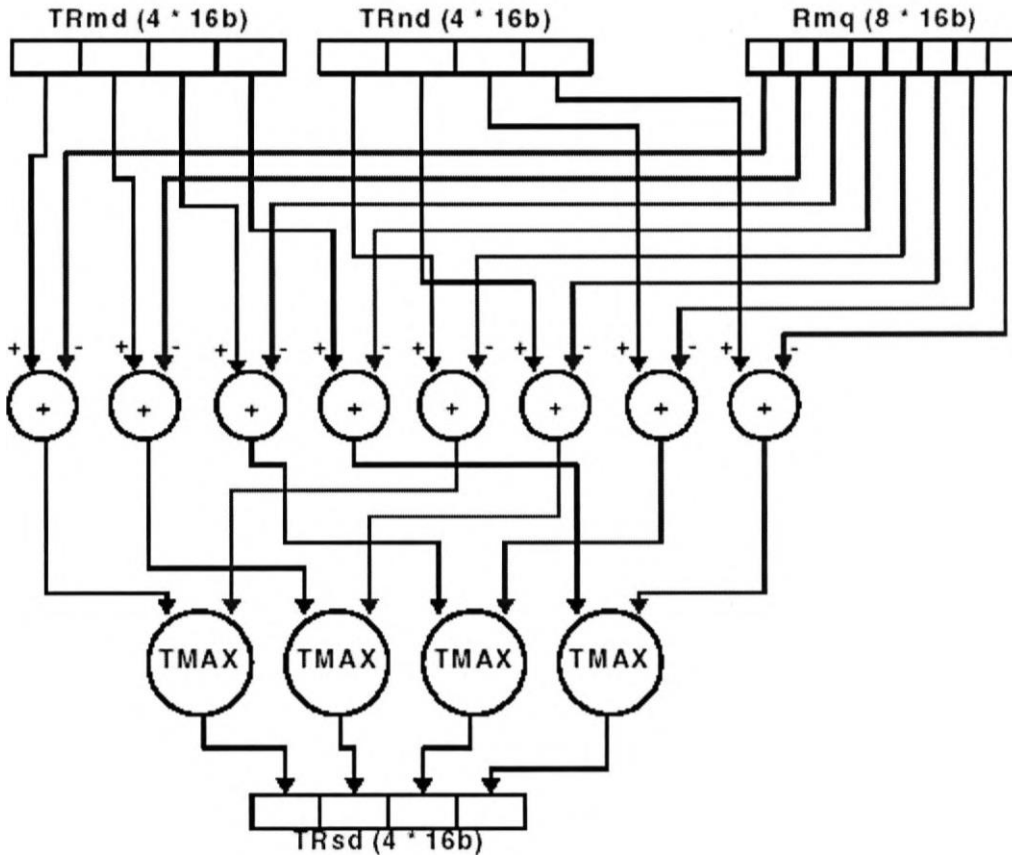


Рисунок 50 – STRsd = TMAX(TRmd - Rmq_h, TRnd-Rmq_l)

4.1.2.3 Решётчатые функции вида SRs = TMAX(TRm,TRn) ;

Как показано на рисунках 51 и 52, операция TMAX выполняется над операндами в регистрах TRm и TRn.

Эта команда может выполняться в параллель с командами ALU, сдвигового устройства, умножителя и командами загрузки регистров CLU. Она не может быть выполнена в параллель с командами CLU того же вычислительного блока.

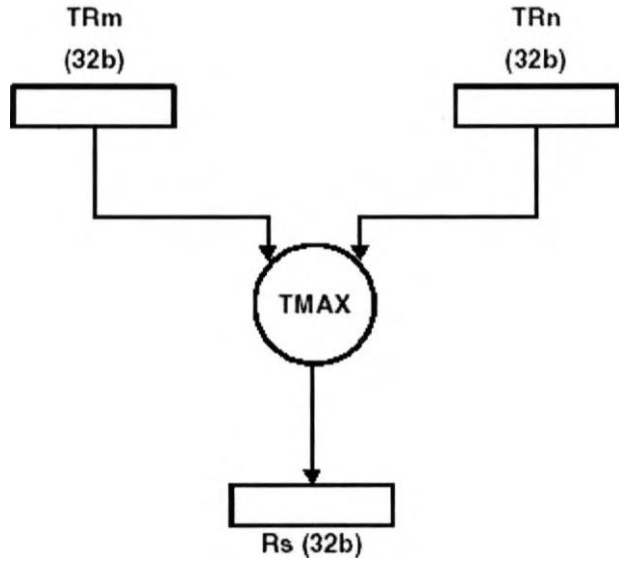


Рисунок 51 – Работа команды Rs = TMAX(TRm, TRn)

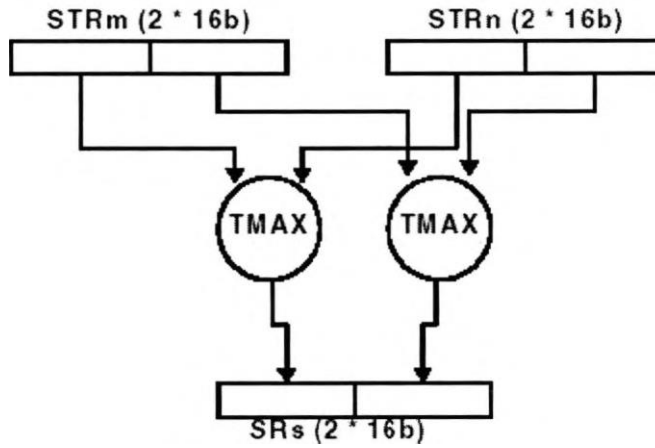


Рисунок 52 – SRs = TMAX (TRm, TRn)

4.1.2.4 Решётчатые функции вида
STRsd = MAX(TRmd+Rmq_h,TRnd+Rmq_l);

Как показано на рисунках 53 и 54, старшая часть регистра Rmq прибавляется к значению в TRmd, младшая часть регистра Rmq прибавляется к значению в TRnd, а затем операция MAX вычисляет максимум из результатов обеих операций сложения.

В данной команде поддерживается режим насыщения.

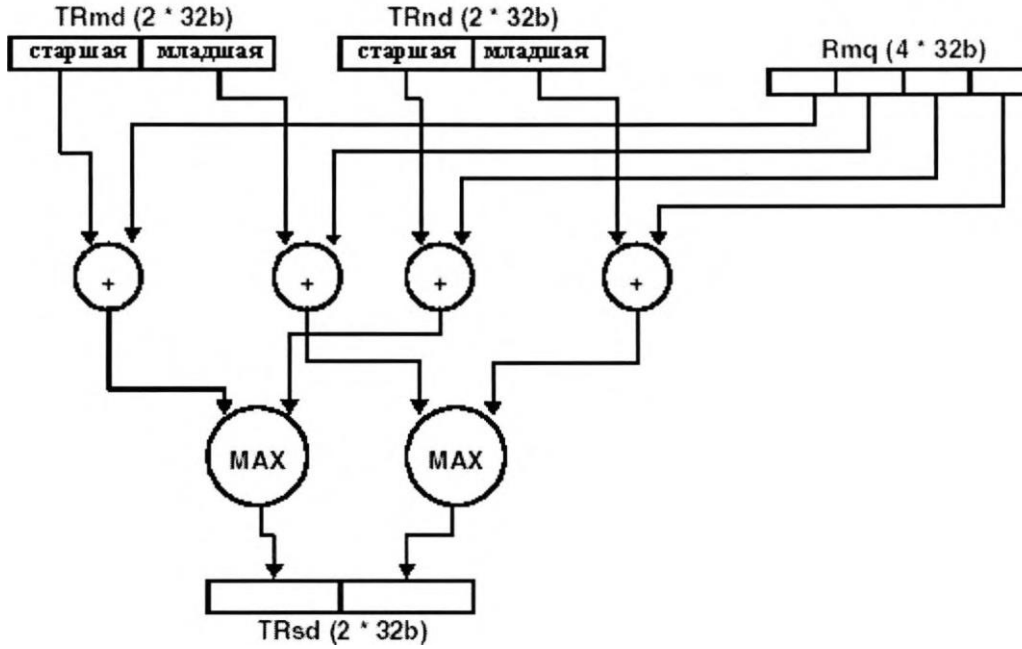


Рисунок 53 – TRsd = MAX(TRmd + Rmq_h, TRnd + Rmq_l)

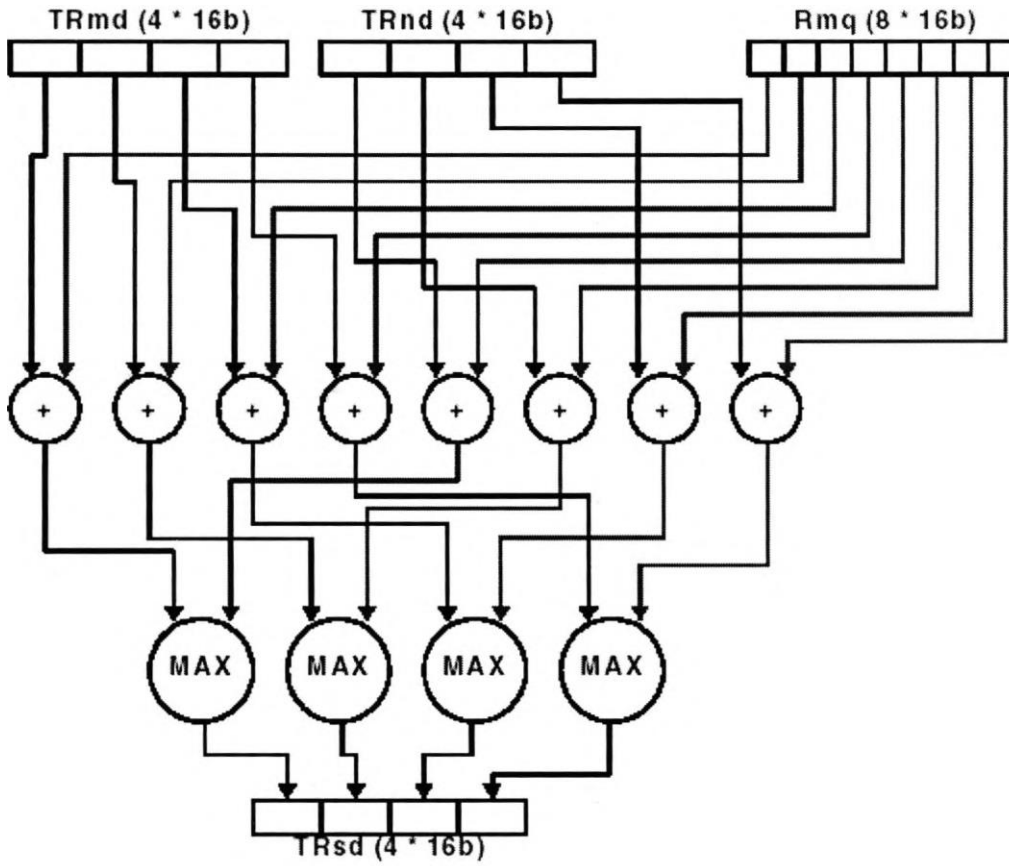


Рисунок 54 – STRsd = MAX(TRmd + Rmq_h, TRnd + Rmq_l)

4.1.2.5 Решётчатые функции вида
STRsd = MAX(TRmd-Rmq_h,TRnd-Rmq_l);

Как показано на рисунках 55 и 56, старшая часть регистра Rmq вычитается из значения в TRmd, младшая часть регистра Rmq вычитается из значения в TRnd, а операция MAX вычисляет максимум из результатов обеих операций вычитания.

В данной команде поддерживается режим насыщения.

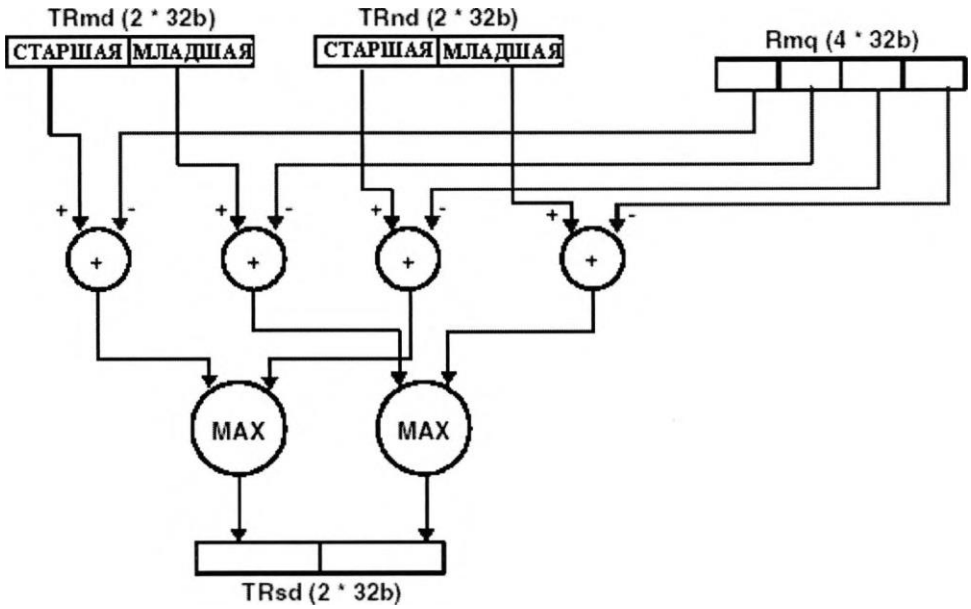


Рисунок 55 – TRsd = MAX(TRmd - Rmq_h, TRnd - Rmq_l)

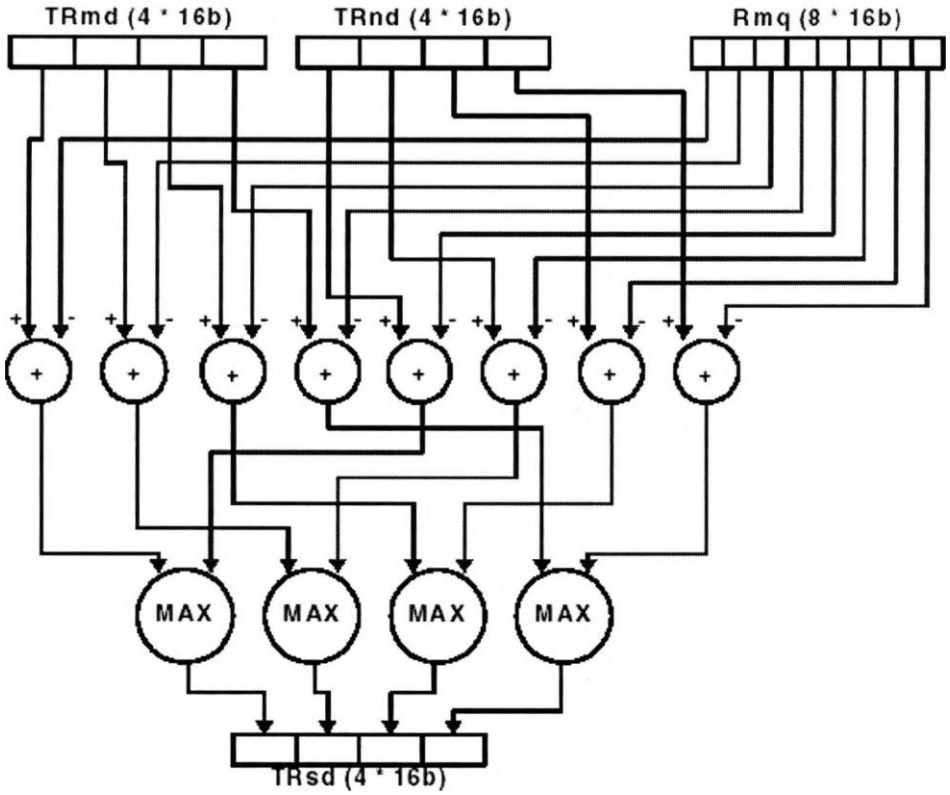


Рисунок 56 – STRsd = MAX(TRmd - Rmq_h, TRnd - Rmq_l)

4.1.3 *Функции сжатия частотной полосы сигнала*

Команда DESPREAD реализует высокопараллельную операцию комплексного умножения с накоплением, оптимизированную для использования в системах CDMA. Процедура сжатия частотной полосы сигнала включает в себя вычисление нескольких наборов корреляций между комплексными входными данными и предвычисленной комплексной рассеивающей/скремблирующей кодовой последовательностью.

Входные данные состоят из отсчётов с 8-битными действительной и мнимой частями. С другой стороны, отсчёты из кодовой последовательности всегда являются элементами множества {1+j, -1+j, -1-j, 1-j} и, следовательно, могут быть заданы 1-битными действительной и мнимой частями. Команда DESPREAD, пользуясь этим свойством, позволяет параллельно вычислять восемь результатов операций умножения с накоплением в каждом вычислительном блоке за один такт.

Команда DESPREAD поддерживает суммирование по последовательностям отсчётов, имеющим длину (фактор рассеивания), равную четырем, восьми или кратную восьми.

Входной регистр Rmq для команды DESPREAD состоит из восьми комплексных коротких слов, с D7 по D0, в которых каждое комплексное число состоит из 2 байт. Старший байт представляет мнимую часть, а младший байт – действительную часть. Как показано на рисунке 57, Dn состоит из DnI и DnQ, где I обозначает действительную часть, а Q – мнимую часть.

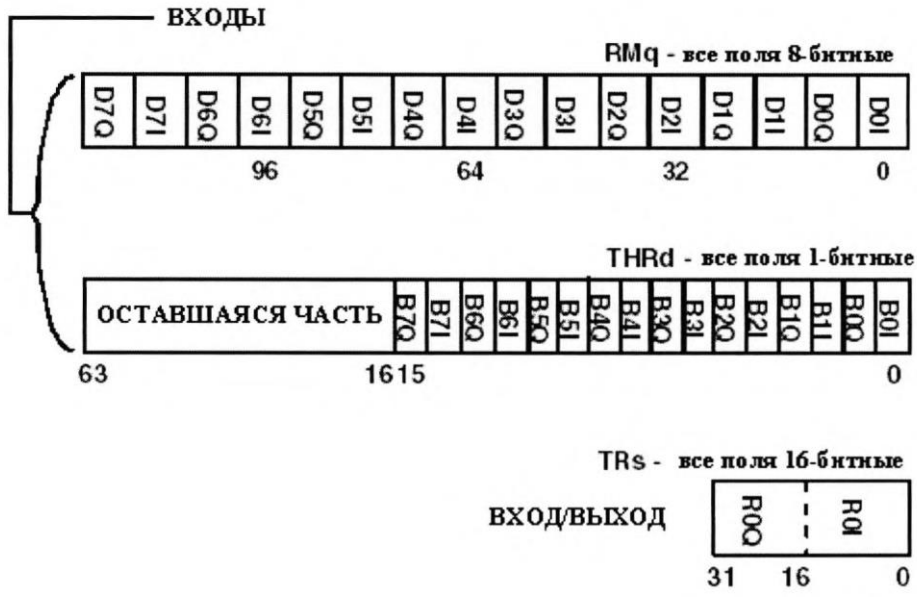


Рисунок 57 – Битовые поля для TRs += DESPREAD (Rmq, THRd);

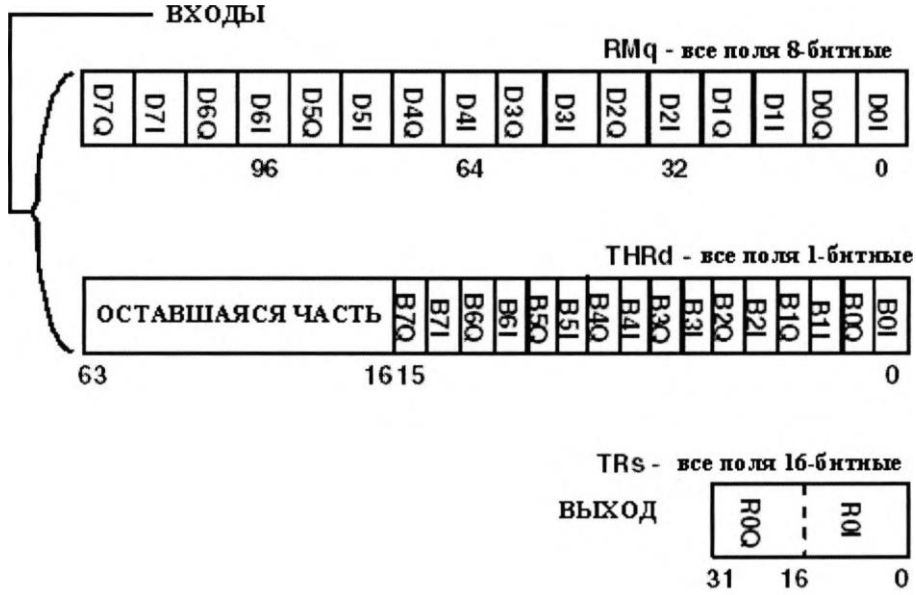


Рисунок 58 – Rs = TRs, TRs = DESPREAD (Rmq, THRd);

Регистр THRd состоит из восьми комплексных чисел {B7...B0} и оставшихся 48 бит. Каждое комплексное число состоит из одного бита действительной части (младший) и одного бита мнимой части. Каждый их этих битов представляет значение +1 (если сброшен) и -1 (если установлен). По завершению операции регистр THRd сдвигается вправо на 16 бит, так что младшие 16 бит оставшейся части могут быть использованы в команде сжатия частотной полосы сигнала на следующем такте. В данной команде режим насыщения всегда включён.

При выполнении этой команды в параллель с командой загрузки регистров THR последняя имеет приоритет над сдвигом регистра THR в данной команде.

4.1.3.1 Функции сжатия частотной полосы сигнала вида TRs += DESPREAD(Rmq, THRd) ;

В регистре TRs содержится комплексное слово, составленное из двух коротких: действительной (младшее короткое) и мнимой (старшее короткое) частей.

Функция (показанная на рисунке 59) может быть определена следующим образом:

RsI = (Сумма (n = 0 по 7) (BnI * DnI - BnQ * DnQ)) RsQ = (Сумма (n = 0 по 7) (BnI * DnQ + BnQ * DnI))

Все умножения выполняются на целое число (±1), результат выровнен по правой границе, и при суммировании происходит распространение знака.

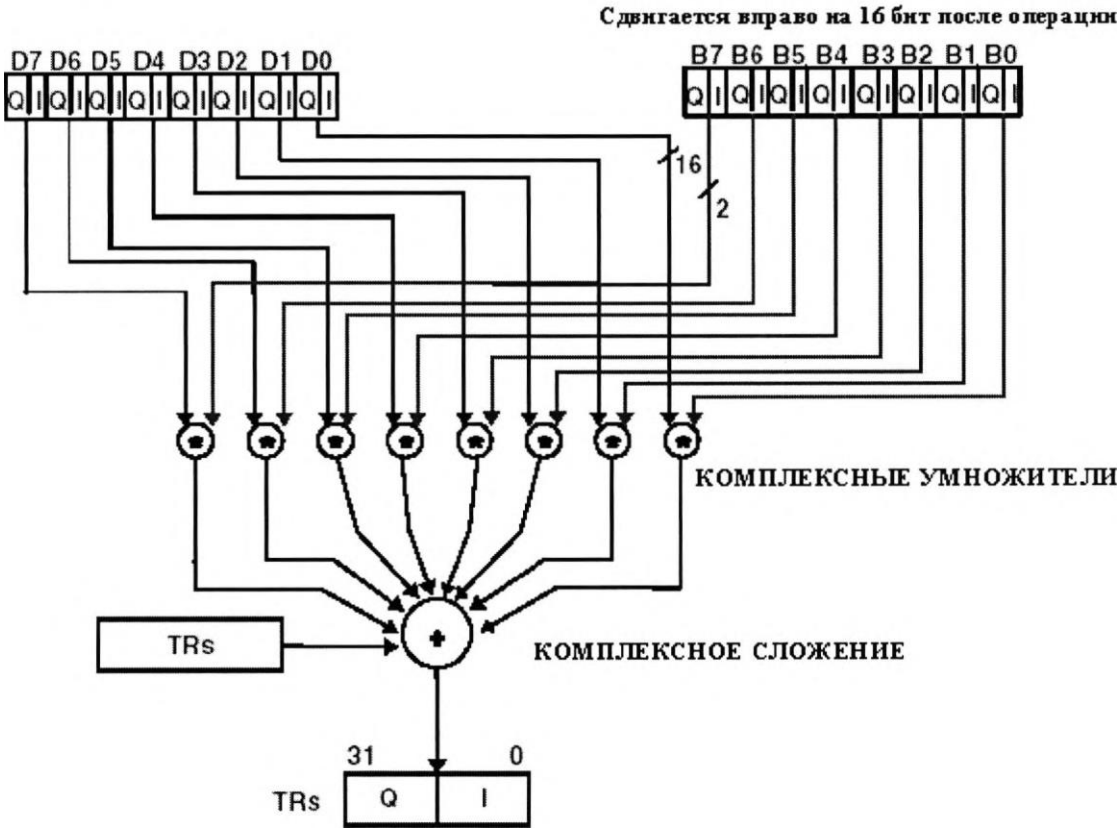


Рисунок 59 – TRs += DESPREAD (Rmq, THRd) ;

4.1.3.2 Функции сжатия частотной полосы сигнала вида Rs = TRs, TRs = DESPREAD (Rmq, THRd) ;

В регистре TRs содержится комплексное слово, составленное из двух коротких: действительной (младшее короткое) и мнимой (старшее короткое) частей.

Функция, показанная на рисунке 60, может быть определена следующим образом:

RsI = (Сумма (n = 0 по 7) (BnI * DnI - BnQ * DnQ)) RsQ = (Сумма (n = 0 по 7) (BnI * DnQ + BnQ * DnI))

Значение в регистре TRs до операции сохраняется в регистре Rs и сдвигается вправо на 16 бит после операции.

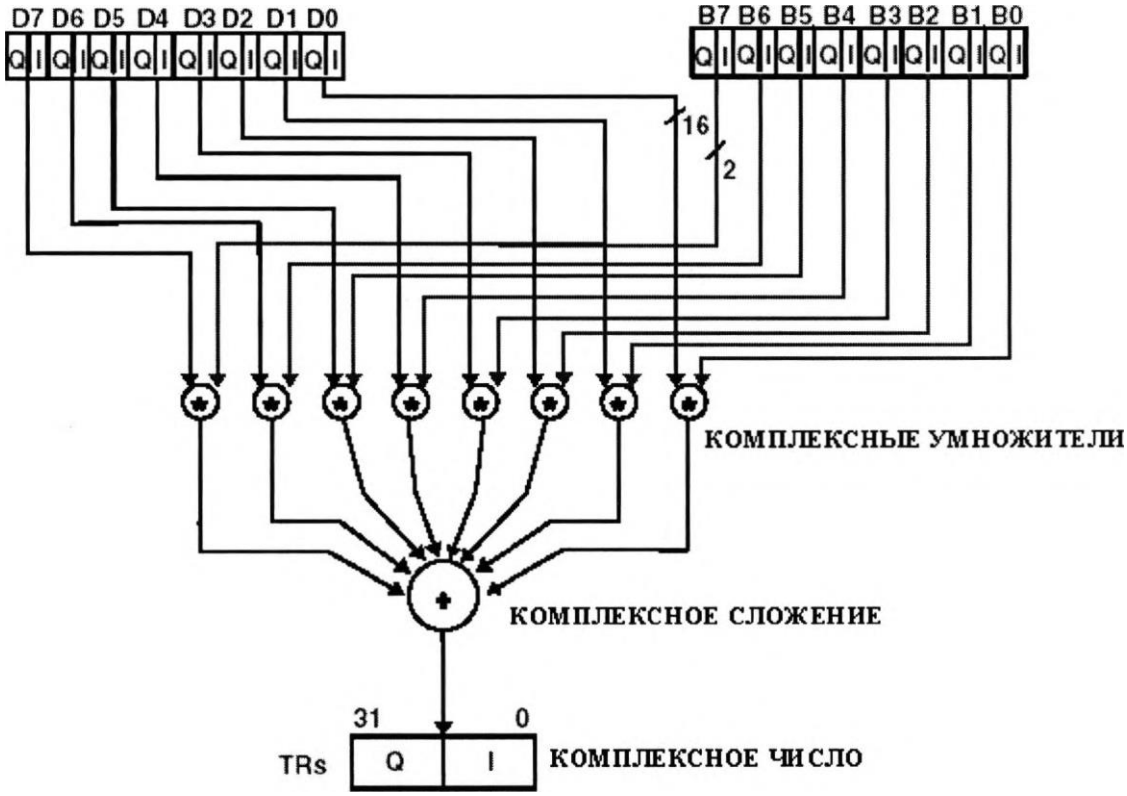


Рисунок 60 – Rs = TRs, TRs = DESPREAD (Rmq, THRd);

4.1.3.3 Функции сжатия частотной полосы сигнала вида Rsd = TRsd, TRsd = DESPREAD (Rmq, THRd);

Функция, показанная на рисунках 61 и 62, может быть определена следующим образом:

R0I = (Сумма (n = 0 no 3) (BnI * DnI - BnQ * DnQ)) R0Q = (Сумма (n = 0 no 3) (BnI * DnQ + BnQ * DnI)) R1I = (Сумма (n = 4 no 7) (BnI * DnI - BnQ * DnQ)) R1Q = (Сумма (n = 4 no 7) (BnI * DnQ + BnQ * DnI))

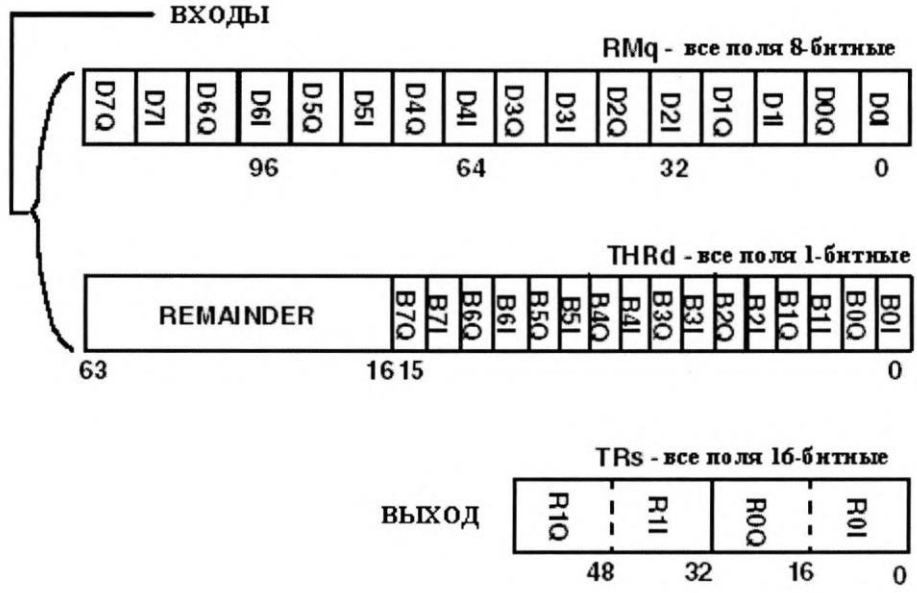


Рисунок 61 – Битовые поля для Rsd = TRsd, TRsd = DESPREAD (Rmq, THRd); Обратите внимание на то, что на выходе получаются два слова

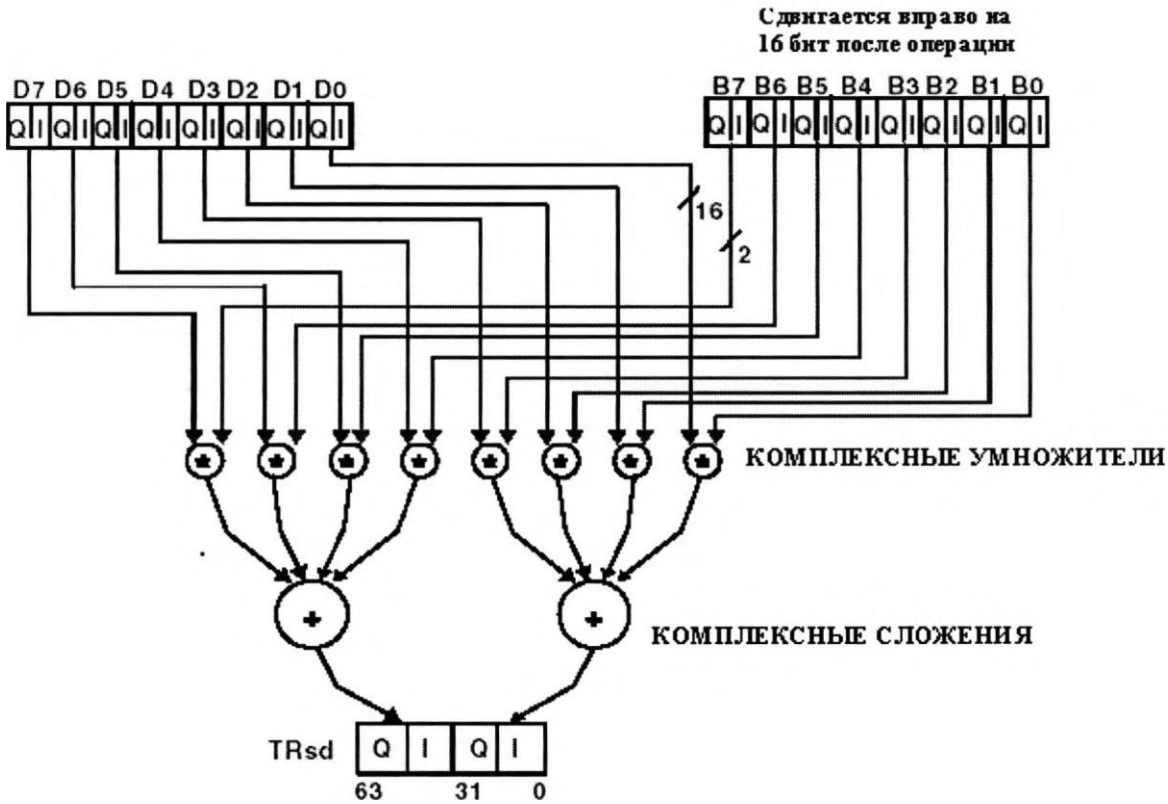


Рисунок 62 – Rsd = TRsd, TRsd = DESPREAD (Rmq, THRd);

4.1.4 *Функция вычисления кросс-корреляций*

Команда XCORRS вычисляет корреляцию длинной входной последовательности (такой как 2048 входных комплексных чисел с 8-битным пилот-сигналом) и известной опорной последовательности для нескольких значений задержки. Удобно представлять команду XCORRS как выполнение 16 параллельных

команд DSPREAD за один такт. Среди важных особенностей команды XCORRS следующие:

- опция Сброс (CLR), предоставляющую механизм для очистки регистров TR перед началом вычисления новой кросс-корреляции;
- опция Усечения входа (CUT), предоставляющую механизм для отбрасывания некоторых входных данных (для начального и конечного циклов; верхний и нижний треугольники);
- опция Повышенной точности (EXT) для поддержки 32-битных комплексных данных на входе и 64-битных комплексных результатов накопления (вместо принятых по умолчанию 16-битных входных данных и 32-битных комплексных результатов накопления);

Для каждой задержки определяется величина корреляции. Уравнения, определяющие XCORRS для вычислительных блоков X и Y

for (к = 0; к ≤15; к++)

{

TR[к] = TR[к] + $\sum_{m=0}^{7} D[m]\, C[m - к + 15]\, f_{CUT}[m - к + 15]$;

}

THR3:0 = THR3:0 >> 16;

где входные данные (в зависимости от значения CUT)[1]

C[т - к + 15] = ±1 ± i (2 бита),

D[m] = ($D_{действ}$[m] + i$D_{мним}$[m]) (16 бит).

Если CUT *положительно*, f_{CUT}[w] = 1 при w < CUT;
0 при w ≥ CUT.

Если CUT *отрицательно*, f_{CUT}[w] = 1 при w ≥ abs(CUT);
0 при w < abs (CUT).

На рисунках 63 и 64 показано расположение битов для команды XCORRS, использующей входные данные одинарной точности (режим по умолчанию). Заметьте, что рисунки демонстрируют 16 операций умножения с накоплением.

[1] При C[ ] = +1 в уравнении, Cx=0 в регистре, а при C[ ] = -1 в уравнении, Cx=1 в регистре (см. Рисунок 61).

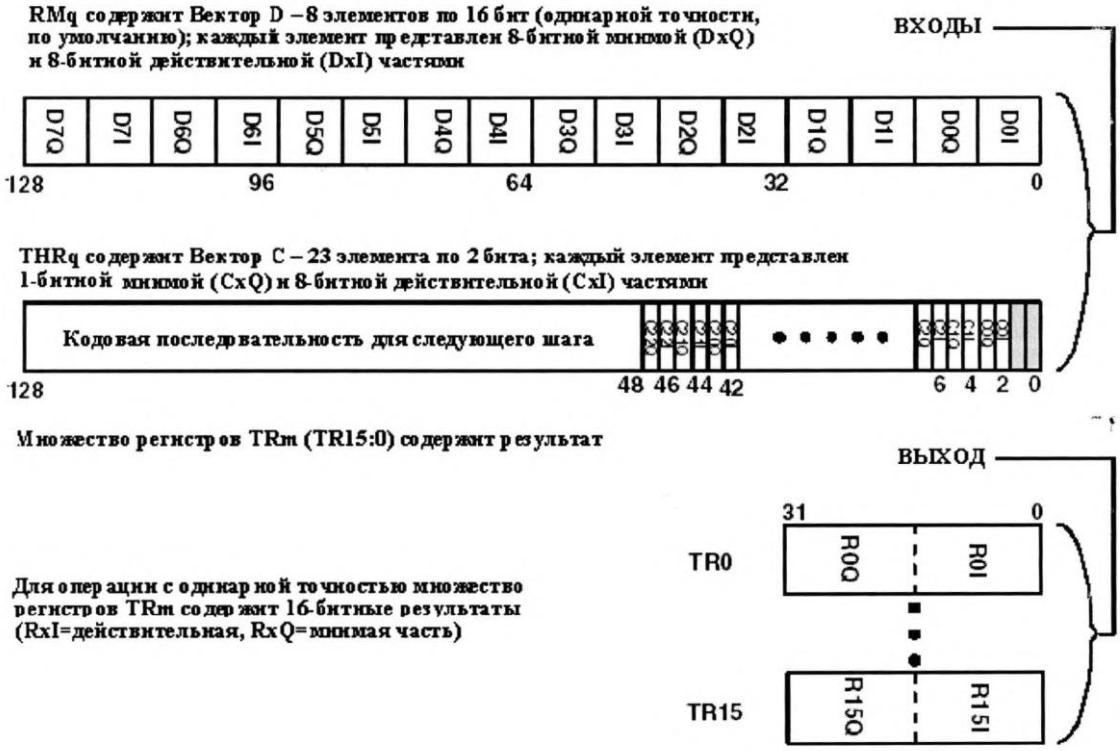


Рисунок 63 – Битовые поля для TRm = XCORRS(Rmq, THRd);

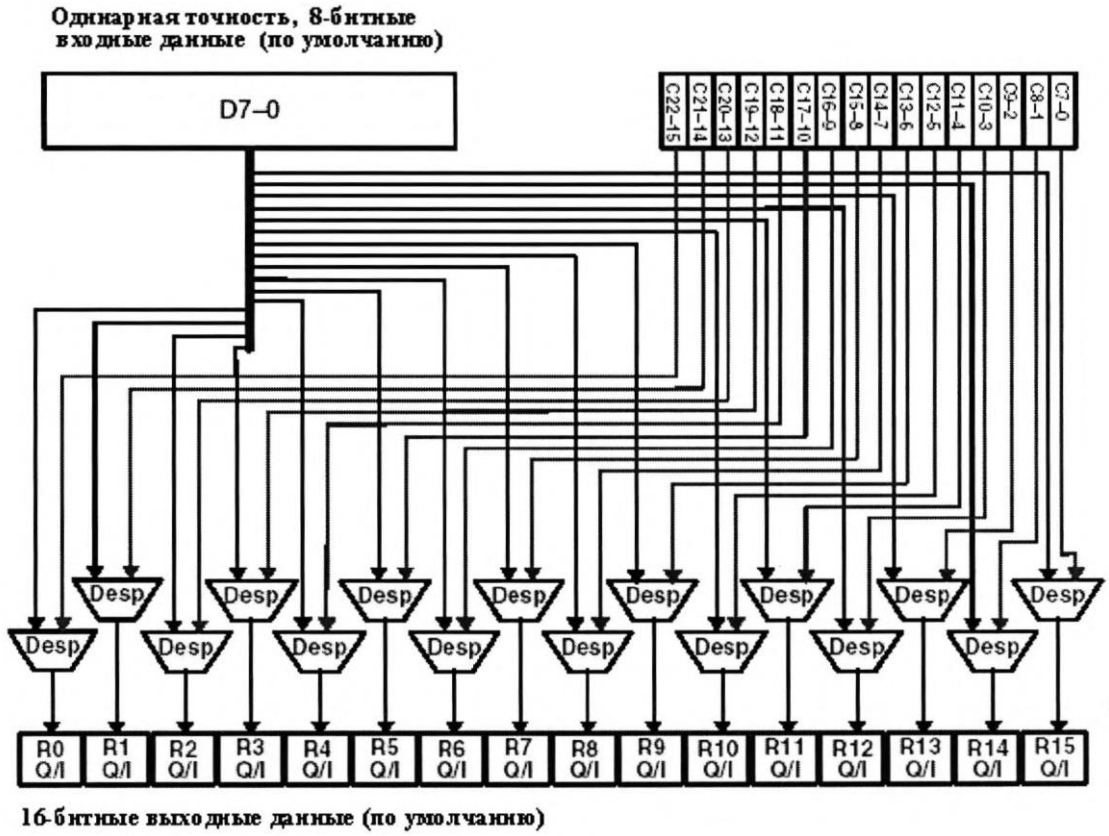


Рисунок 64 – TRm = XCORRS(Rmq, THRd);

На рисунках 65 и 66 показано расположение битов для команды XCORRS, использующей входные данные повышенной точности (включена опция EXT). Заметьте, что рисунки демонстрируют 8 операций умножения с накоплением.

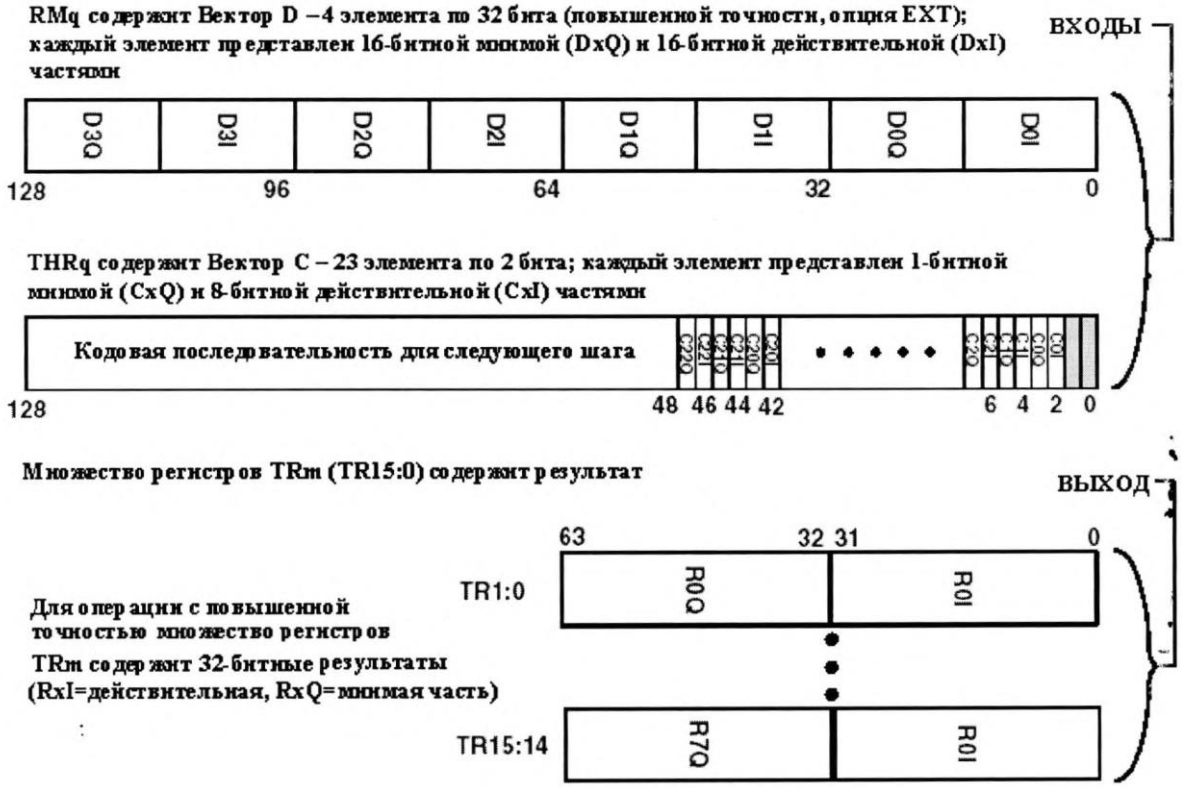


Рисунок 65 – Битовые поля для TRm = XCORRS(Rmq, THRd) (EXT);

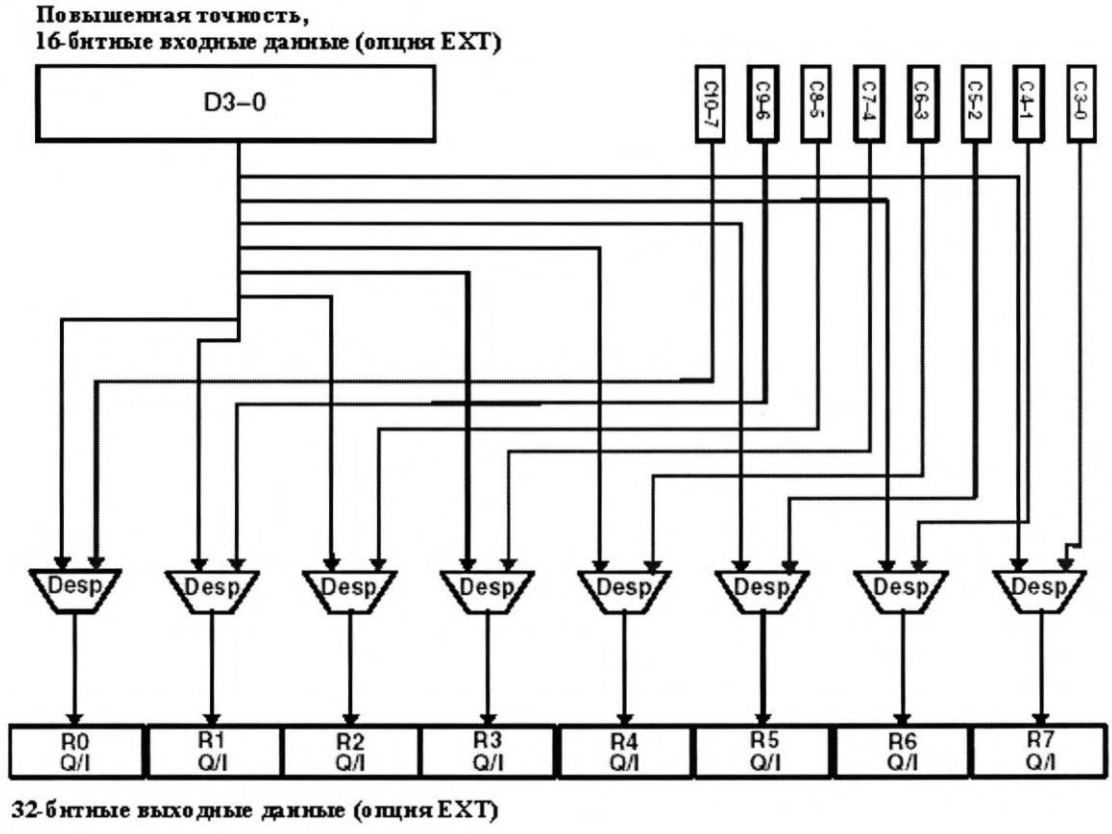


Рисунок 66 – TRm = XCORRS(Rmq, THRd) (EXT);

Примечание – При выполнении этой команды параллельно с командой загрузки регистров THR последняя имеет приоритет над сдвигом регистра THR в данной команде.

Когда опция CUT опущена, предполагается значение CUT=0; нумерация кодовой последовательности в этом случае изображена на рисунке 67. Обратите внимание, что каждый индекс для кодовой последовательности на рисунке 67 отображается на вектор “C” с рисунка 64.

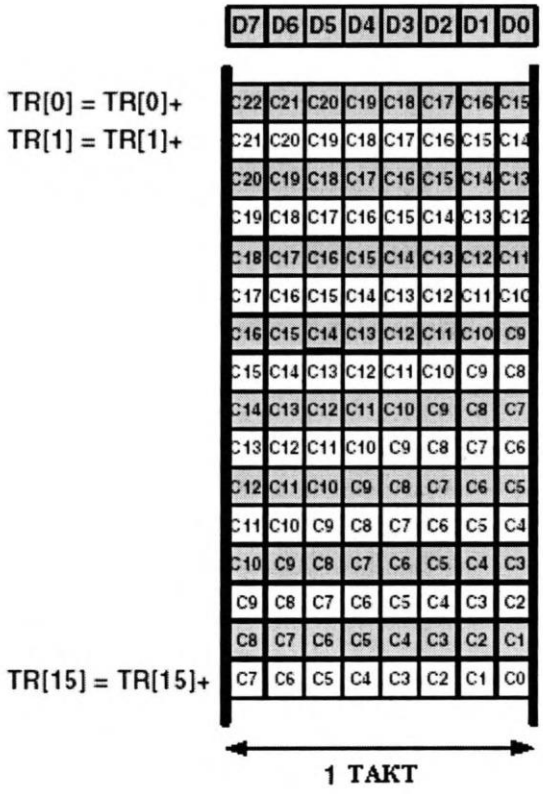


Рисунок 67 – Нумерация кодовой последовательности при CUT = 0

Когда значение в опции CUT отрицательно (например, CUT = -k), все операции Умножения с накоплением для всех элементов вектора “C” до элемента C[k] игнорируются.

На рисунке 68 показан пример при CUT = -7.

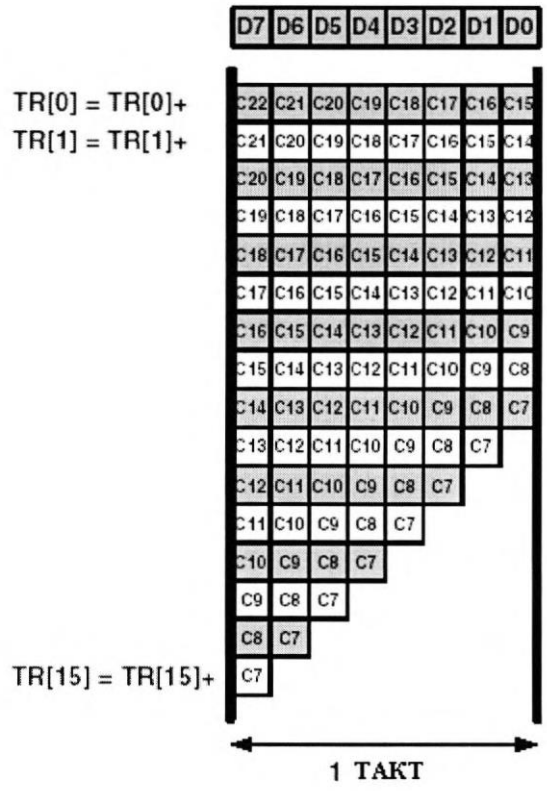


Рисунок 68 – Пример нумерации кодовой последовательности при отрицательном значении CUT

Когда значение в опции CUT положительно (например, CUT = + k), все операции умножения с накоплением для элемента C [k] и всех элементов вектора “C” после него игнорируются.

На рисунке 69 показан пример при CUT = +15.

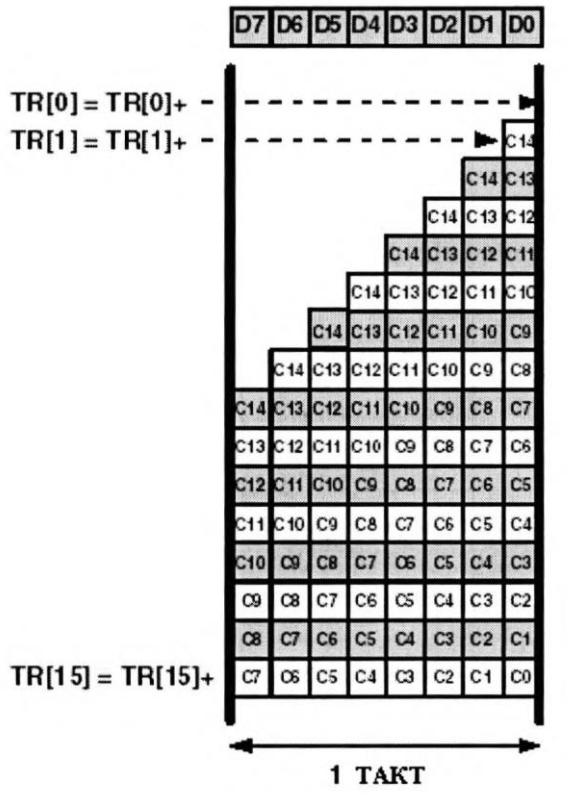


Рисунок 69 – Пример нумерации кодовой последовательности при положительном значении CUT

4.1.5 *Функция сложить/сравнить/выбрать*

При выполнении функции ACS в команде T*Rsq* = ACS (T*Rmd*, T*Rnd, Rm)* ; каждое *короткое слово* в *Rm* прибавляется и вычитается из соответствующих коротких слов в T*Rmd* и T*Rnd.* Затем, как показано на рисунке 70, *четыре* результата сложения и вычитания сравниваются по решётчатому принципу.

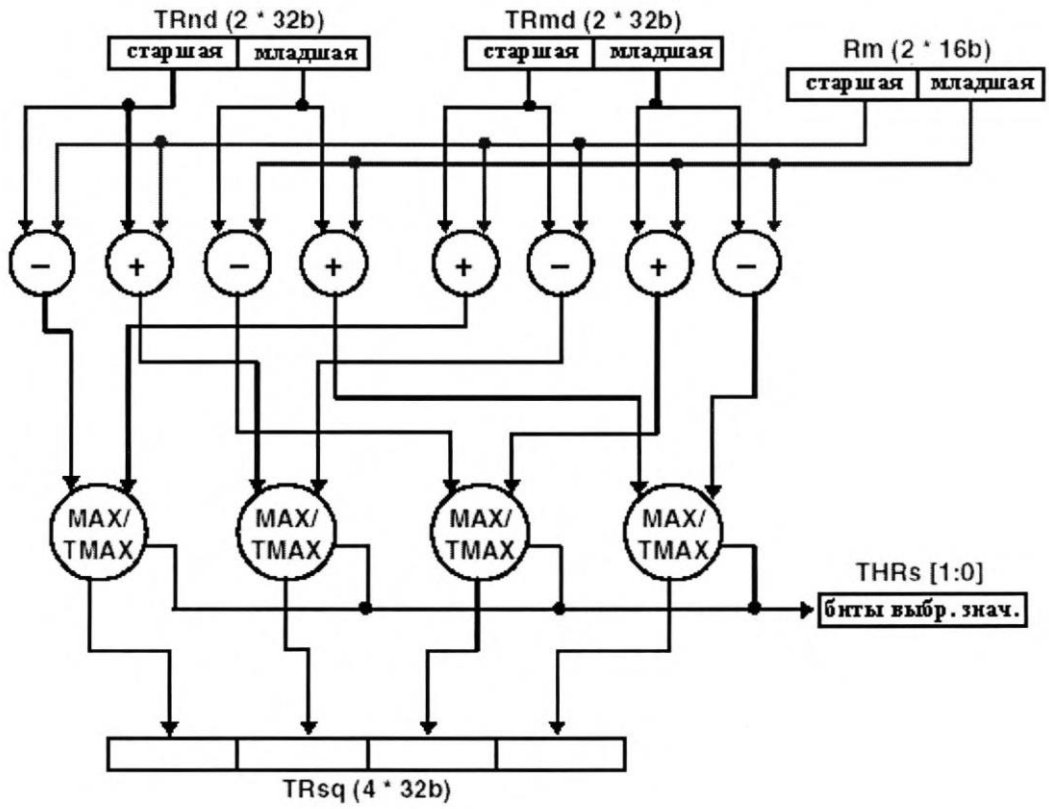


Рисунок 70 – TRsq = ACS (TRmd, TRnd, Rm);

В регистр истории решётчатых операций THR [1:0] заносится значение, определяющее выбор функций MAX/TMAX. После каждой операции ACS содержимое THR1: 0 сдвигается вправо на *4 бита,* и *четыре* бита выбранных значений загружаются в биты 31:28 регистра THR1. Бит выбранного значения указывает, какой из входов был выбран функцией MAX или TMAX. При выборе T*Rm+/-Rm* бит выбранного значения равен 1, а при выборе T*Rn-/+Rm* бит выбранного значения равен 0.

При выполнении команды ST*Rsq* = ACS (T*Rmd,* T*Rnd*, *Rm)* ; каждый *байт* в *Rm* прибавляется и вычитается из соответствующих байтов в T*Rmd* и T*Rnd*. Затем, как показано на рисунке 71, *восемь* результатов сложения и вычитания сравниваются по решётчатому принципу.

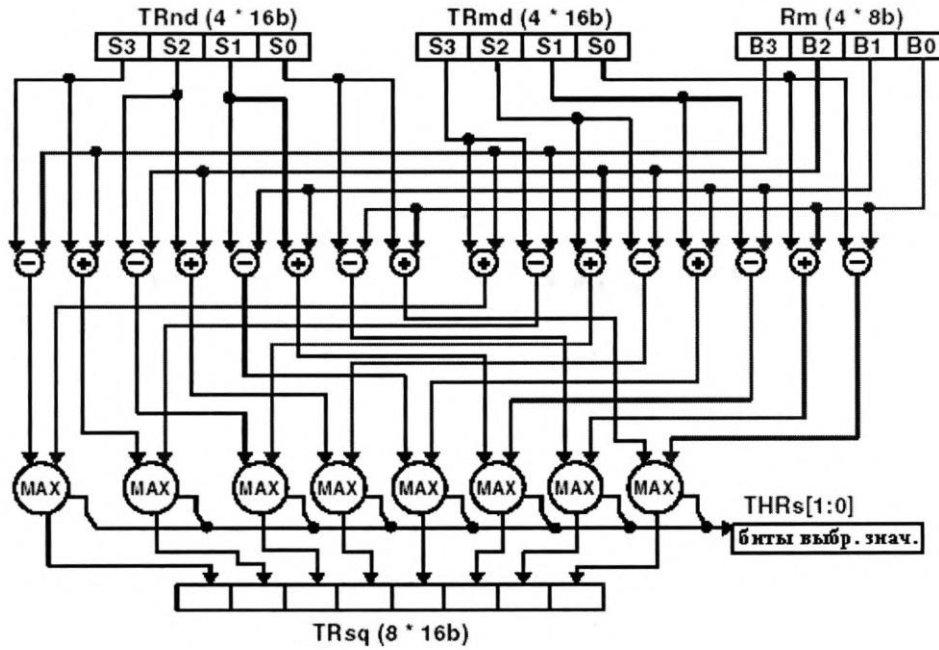


Рисунок 71 – STRsq = ACS (TRmd, TRnd, Rm);

В регистр истории решётчатых операций THR [1:0] заносится значение, определяющее выбор функций MAX/TMAX. После каждой операции ACS содержимое THR1: 0 сдвигается вправо на *8 бит,* и *восемь* битов выбранных значений загружаются в биты 31:24 регистра THR1. Бит выбранного значения указывает, какой из входов был выбран функцией MAX или TMAX. При выборе T*Rm+/-Rm* бит выбранного значения равен 1, а при выборе T*Rn-/+Rm* бит выбранного значения равен 0.

Эта команда допускает добавление пересылки регистров решётчатых операций (команда с двойной операцией), как например в:

Rsq = TRaq, {S}TRsq = ACS (TRmd, TRnd, Rm);

4.1.6 *Опции команд CLU*

Некоторым командам CLU сопоставлены опции, позволяющие гибко управлять режимом исполнения данной команды. Важно отметить, что эти опции влияют на то, как именно выполняются команды, а также тот факт, что существуют опции, применимые лишь с определёнными группами команд – не все опции используемы со всеми командами. Опции команд указываются в круглых скобках в конце поля команды. Список всех опций, используемых с конкретными командами CLU, можно найти в разделе «Сводка команд CLU». Опции команд CLU:

- (TMAX) указывает, что надо использовать операцию TMAX вместо операции MAX (только для команды ACS);
- (CUT *Imm)* определяет число данных, отбрасываемых от множества при вычислении кросс-корреляции (только для команды XCORRS);
- (CLR) сбрасывает указанные регистры решётчатых операций до следующего суммирования при вычислении кросс-корреляции;
- (EXT) указывает на входные данные и результаты повышенной точности при вычислении кросс-корреляции (только для команды XCORRS);
- (NF) не обновлять флаги.

4.1.7 *Статус выполнения CLU*

Операции CLU изменяют флаги состояния арифметических регистров состояния вычислительных блоков (XSTAT и YSTAT). Программы могут использовать флаги состояния для управления исполнением условных команд и порождения программных исключений.

В таблице 10 приведены флаги регистров XSTAT и YSTAT, отражающие статус завершения последней операции CLU.

Таблица 10 – Флаги состояния CLU

Флаг	Определение	Изменяется...
TROV	Переполнение в CLU	Всеми операциями CLU

Операции CLU также изменяют залипающие флаги состояния арифметических регистров состояния вычислительных блоков (XSTAT и YSTAT). В таблице 11 приведены флаги регистров XSTAT и YSTAT, отражающие залипающий статус завершения последней операции CLU. Установленный однажды, залипающий флаг остаётся в этом положении до тех пор, пока не будет сброшен явно.

Таблица 11 – Залипающие флаги состояния CLU

Флаг	Определение	Изменяется...
TRSOV	Переполнение в CLU, залипающий	Всеми операциями CLU

Обновление флагов происходит в конце выполнения каждой операции, делая флаги доступными уже в следующем поле команды. Программа не может записать значение в арифметические регистры состояния явно в том же командном такте, в котором происходит обновление слова состояния.

Команды с множественными операндами (например, ST*Rsd* = TMAX (T*Rmd + Rmq h,* T*Rnd + Rmq_1)* ;) порождают несколько результатов. В этом случае процессор вычисляет флаг состояния как битовое ИЛИ соответствующих флагов для каждого результата.

Примечание – При использовании опции (NF) вычислительное устройство не обновляет значения флагов состояния в соответствии со статусом завершения операции. Разрешено использовать опцию (NF) со всеми командами CLU, за исключением команд пересылки регистров TRx, THRx и CMCTL.

4.2 Примеры использования CLU

Команды CLU были разработаны для облегчения программирования различных алгоритмов, используемых в телекоммуникационных приложениях. Эти команды были разработаны в основном для использования в следующих алгоритмах (хотя для них имеется множество других возможных приложений):

- декодирование по Витерби;
- декодирование турбо кодов;
- сжатие частотной полосы сигнала в системах CDMA.

В листинге 8 представлен пример команды DESPREAD, в нём отмечены вычисления, проводимые командой.

Листинг 8 – Пример команды DESPREAD (команда CLU) Для следующих значений в регистрах

```
xTR0 = 0x00010002 xTHR1 = 0x00001111 xTHR0 = 0x11110000 xR3 = 0x00010203
xR2 = 0x04050607
xR1 = 0x08090a0b
```

xR0 = 0x0c0d0e0f

Выполним следующую команду:

xTR0 += DESPREAD (R3:0, THR1:0) ;;

Получим следующие значения в регистрах:

xTR0=0x00780008
xTHR1=0x00000000
xTHR=0x11111111

Действительная часть результата

= (0x01 - 0x00)
+ (0x03 - 0x02)
+ (0x05 - 0x04)
+ (0x07 - 0x06)
+ (0x09 - 0x08)
+ (0x0b - 0x0a)
+ (0x0d - 0x0c)
+ (0x0f - 0x0e)
= 0x0008

Мнимая часть результата

= (0x01 + 0x00)
+ (0x03 + 0x02)
+ (0x05 + 0x04)
+ (0x07 + 0x06)
+ (0x09 + 0x08)
+ (0x0b + 0x0a)
+ (0x0d + 0x0c)
+ (0x0f + 0x0e)
= 0x78

Заметьте, что значения в THR изменились и сдвинулись вправо на 16 бит и что результат в TR0 также изменился. Выполним эту команду ещё раз:

xTR0 += DESPREAD (R3:0, THR1:0) ;;

До накопления результаты таковы:

Действительная часть результата

= (-0x01 + 0x00)
+ (-0x03 + 0x02)
+ (-0x05 + 0x04)
+ (-0x07 + 0x06)
+ (-0x09 + 0x08)
+ (-0x0b + 0x0a)
+ (-0x0d + 0x0c)
+ (-0x0f + 0x0e)
= 0xFFF8

Мнимая часть результата

= (-0x01- 0x00)
+ (-0x03 - 0x02)
+ (-0x05 - 0x04)
+ (-0x07 - 0x06)
+ (-0x09 - 0x08)
+ (-0x0b - 0x0a)
+ (-0x0d - 0x0c)
+ (-0x0f - 0x0e)
= 0xFF88

Проведение накопления в регистре TR0 в последней команде даст окончательный результат:

xTR0=0x00000000
xTHR1=0x00000000
xTHR0=0x00001111

Так как входные данные (R3:0) для двух команд DESPREAD были одинаковы, а кодовая последовательность изменилась с 1+j на-1-j, накопления полностью сократились после выполнения второй команды.

4.3 Сводка команд CLU

Листинг 9 «Команды CLU» демонстрирует синтаксис команд CLU. Соглашения по обозначениям имён регистров, необязательных элементов или вариантов, используемые в этих листингах, подробно освещены в разделе «Регистры регистровых файлов». Вкратце эти соглашения таковы:

- { } - необязательные элементы указаны в фигурных скобках; фигурные скобки не являются частью синтаксиса команды;
- | - вертикальная черта разделяет варианты значений элементов; вертикальные черты не являются частью синтаксиса команды;
- Rmd - имена регистров, выделенные курсивом, представляют собой подставляемые шаблоны для имён одиночных (Rs, Rm или Rn), двойных (Rsd, Rmd или Rnd) или квадро- (Rsq, Rmq или Rnq) регистров.

Листинг 9 – Команды CLU

```
{X|Y|XY}{S}T Rsd = TMAX ( T Rmd + Rmq_h, T Rnd + Rmq_1) {(NF)} ;
{X|Y|XY}{ S }T Rsd = TMAX ( T Rmd - Rmq_h, T Rnd - Rmq_1) {(NF)} ;
{X|Y|XY}{S} Rs = TMAX(T Rm, T Rn) {(NF)} ;
{X|Y|XY}{S}T Rsd = MAX ( T Rmd + Rmq_h, T Rnd + Rmq_1) { (NF) } ;
{X|Y|XY}{S}T Rsd = MAX ( T Rmd - Rmq_h, T Rnd - Rmq_1) {(NF)} ;
/* здесь Rmq_h должен быть старшей частью квадрорегистра, а Rmq_l должен быть
младшей частью ТОГО ЖЕ квадрорегистра */
{X|Y|XY} Rs = T Rm ;
{X|Y|XY} Rsd = T Rmd ;
{X|Y|XY} Rsq = T Rmq ;
{X|Y|XY} Rs = TH Rm ;
{X|Y|XY} Rsd = TH Rmd ;
{X|Y|XY} Rsq = TH Rmq ;
{X|Y|XY} Rs = CMCTL ;
/* Синтаксис команд загрузки регистров TR, THR и CMCTL можно найти в разделе
«Краткий справочник по сдвиговому устройству» * /
{X|Y|XY}T Rs += DESPREAD (Rmq, TH Rd ) ;
{X|Y|XY} Rs = T Rs, T Rs = DESPREAD (Rmq, TH Rd ) {(NF)} ; (двойная операция)
{X|Y|XY}R sd = T Rsd, T Rsd = DESPREAD (Rmq, TH Rd ) {(NF)} ; (двойная операци)
{X|Y|XY}T Rsa = XCORRS (Rmq, TH Rnq ) { (CUT< Imm> | R)} { (CLR) ) { (EXT) } { (NF) }
;
{X|Y|XY} Rsq = TRbq, T Rsa = XCORRS (Rmq, THRnq)
{(CUT <Imm>|R)}{(CLR)}{ (EXT)}{(NF)} ; (двойная операция)
/* здесь T Rsa = TR15:0 или TR31:16 */
{X|Y|XY}(S)T Rsq = ACS (T Rmd, T Rnd, Rm) {(TMAX)} { (NF)} ;
{X|Y|XY} Rsq = T Raq, (S) T Rsq= ACS (T Rmd, TRnd, Rm) {(TMAX)}{(NF)} ; (двойная
операция)
```

5 Умножитель

Ядро процессора содержит два вычислительных устройства, известных как вычислительные блоки. Каждый вычислительный блок содержит регистровый файл и четыре независимых вычислительных устройства - ALU, CLU, умножитель и сдвиговое устройство. Умножитель выделен на рисунке 72. Умножитель берёт входные данные из регистрового файла и сохраняет выходные данные в регистровом файле или регистрах накопления MR.

Данная глава содержит разделы:

- «Операции умножителя»;
- «Примеры умножителя»;
- «Сводка команд умножителя».

Умножитель выполняет в процессоре все *операции умножения* над данными в форматах с фиксированной и плавающей точкой и все *операции умножения с накоплением* над данными в формате с фиксированной точкой. Это устройство выполняет также все *операции комплексного умножения* над данными в формате с фиксированной точкой. И наконец, умножитель выполняет *операции сжатия данных* над накопленными результатами при пересылке данных в регистровый файл в форматах с фиксированной точкой.

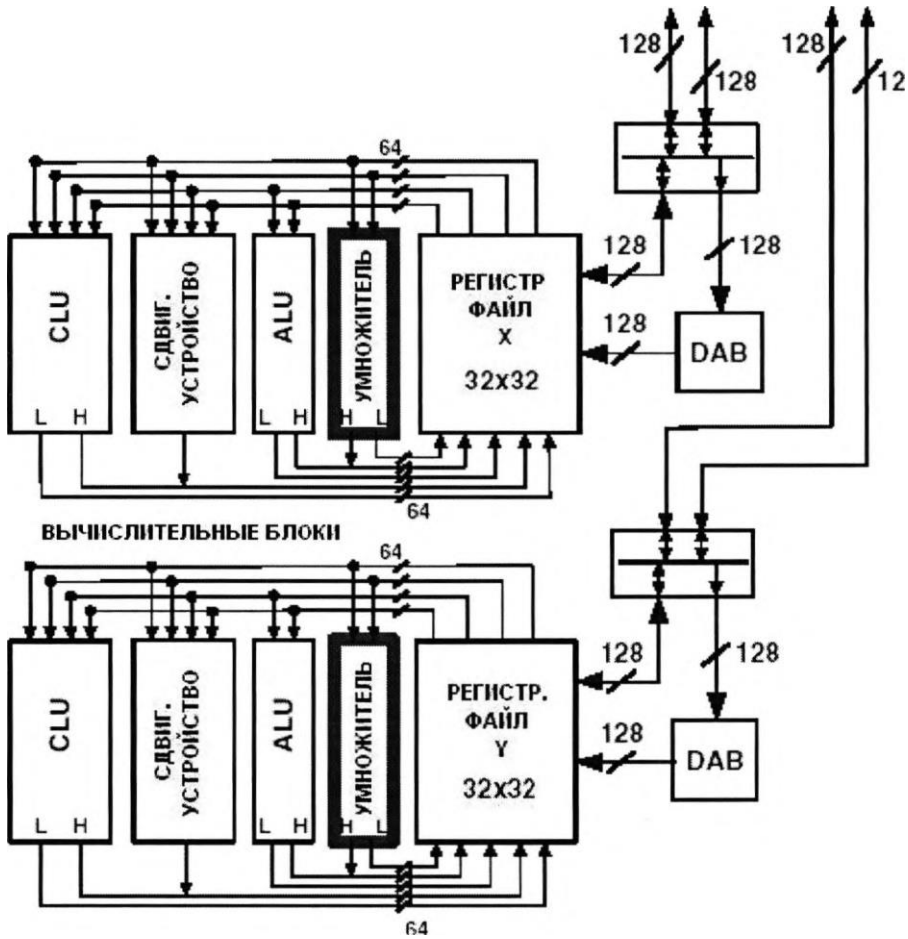


Рисунок 72 – Умножители вычислительных блоков X и Y

Просматривая возможные операнды каждой команды, можно увидеть, какие классы операций умножителя какие типы данных поддерживают.

Операции умножения и умножения с накоплением с дробными и целыми аргументами в формате с фиксированной точкой поддерживают:

- восемь 16-битных (коротких) входных операндов при 16-, 20-, 32- или 40-битном результате;
- два 32-битных (обычных) входных операнда при 32-, 40-, 64- или 80-битном результате.

Операции комплексного умножения с накоплением с дробными и целыми аргументами в формате с фиксированной точкой поддерживают:

- два 32-битных (16-битная действительная часть, 16-битная мнимая часть) входных операнда при 40-битном (20-битная действительная часть, 20-битная мнимая часть) результате или 80-битном (40-битная действительная часть, 40-битная мнимая часть) результате.

Операции умножения с аргументами в формате с плавающей точкой поддерживают:

- два 32-битных (одинарная точность) входных операнда (в формате IEEE) при 32-битном результате;
- два 40-битных (расширенная точность) входных операнда при 40-битном результате;
- два 64-битных (двойная точность) входных операнда (в формате IEEE) при 64-битном результате.

Операции сжатия данных с аргументами в формате с фиксированной точкой поддерживают:

- 20-битные (короткие) входные операнды о 40-битные (обычные) входные операнды о 80-битные (длинные) входные операнды о 16- или 32-битные результаты на выходе.

Имеются следующие особенности работы умножителя с данными в формате с фиксированной точкой:

- умножитель может использовать два 32-битных обычных слова для получения 64- или 32-битного произведения, либо использовать восемь 16-битных коротких слов для получения четырёх 32- или 16-битных произведений. Умножитель не поддерживает однобайтные слова;
- результат операции умножителя (за исключением команды сжатия) всегда имеет размер больший или равный размеру операндов;
- умножение обычных слов даёт результат в виде обычного или длинного слова;
- квадроумножение коротких слов всегда даёт результат в виде четырёх коротких или обычных слов.

Команды умножения могут использовать константу в случае команды с двумя операндами. В этом случае второй операнд (регистр Rn) не используется и его номер (биты 4:1) принимает участие в формировании константы.

Процессор поддерживает операции комплексного умножения с накоплением. Комплексные числа представляются в виде пары коротких слов в 32-битном регистре. Младшая часть в регистрах входных операндов (*Rm_L*, *Rn_L)* определяет действительную часть, а старшая часть в регистрах входных операндов (*Rm_H*, *Rn_H)* определяет мнимую часть. Результат комплексного умножения всегда сохраняется в паре регистров MR. Операция комплексного умножения с накоплением (обозначается оператором **) определяется так:

$$\text{Действ. часть результата} = (\text{Действ.}_{Rт_L} \times \text{Действ.}_{Rn_L}) - (\text{Мним.}_{Rm_H} \times \text{Мним.}_{Rn_H});$$

$$\text{Мним. часть результата} = (\text{Действ.}_{Rm_L} \times \text{Мним.}_{Rn_H}) + (\text{Мним.}_{Rт_H} \times \text{Действ.}_{Rn_L}).$$

Операции комплексного умножения с накоплением имеют опцию, предписывающую умножать значение первого комплексного операнда *(Rm)* на комплексно сопряжённое значение второго *(Rn).* Эта операция с комплексным сопряжением определяется так:

Действ. часть результата = (Действ.$_{Rт_L}$ × Действ.$_{Rn_L}$) + (Мним. $_{Rm_H}$ × Мним. $_{Rn_H}$);

Мним. часть результата = - (Действ. $_{Rm_L}$ × Мним. $_{Rn_H}$) + (Мним. $_{Rт_H}$ × Действ. $_{Rn_L}$).

Опция комплексного сопряжения включается при указании (J) после команды (см. раздел «Опция Комплексное сопряжение»).

Процессор поддерживает 32-битный и 64-битный форматы данных с плавающей точкой одинарной и двойной точности в стандарте IEEE с небольшими отличиями.

Правила именования регистров в команде определяют тип данных и размер входных операндов и результата на выходе. Информацию об указании типа данных в командах умножителя можно найти в разделе «Регистры регистровых файлов». Информацию об указании размера данных в командах умножителя см. в примерах в разделе «Операции умножителя».

Примечание – Соглашения, принятые для указания размера входных операндов и результата в командах умножителя, немного отличаются от соглашений, используемых в командах ALU и сдвигового устройства.

Часть этой главы посвящена описанию команд умножителя, их опций и результатов с использованием синтаксиса команд. Объяснение соглашений по синтаксису команд умножителя и других устройств, можно найти в разделе «Синтаксис и структура командной строки». Список команд умножителя и их синтаксис приведён в разделе «Сводка команд умножителя».

5.1 Операции умножителя

Умножитель выполняет операцию умножения операндов с фиксированной или плавающей точкой, а также операции умножения с накоплением с фиксированной точкой. Умножитель поддерживает несколько типов данных в форматах с фиксированной и плавающей точкой. Поддерживаются форматы чисел с плавающей точкой одинарной точности, расширенной и двойной точности. Входные операнды большинства операций берутся из регистрового файла вычислительного блока, туда же помещается результат операции.

В умножителе имеется один пятисловный регистр специального назначения – регистр MR, хранящий результат накопления. Процессор использует регистр MR для сохранения результатов операций умножения с накоплением с фиксированной точкой. Также умножитель может также переслать содержимое регистра MR в регистровый файл до выполнения накопления. Старшие 32 бита регистра MR (MR4) хранят значение переполнения в результате операций умножения с накоплением. Более подробную информацию о регистре MR см. в разделе «Регистр переполнения результата умножителя (MR4)».

Рисунки с 73 по 75 иллюстрируют выполнение операций умножителя. Ниже приведены команды умножителя, демонстрирующие операции умножения и умножения с накоплением.

```
XR2 = R1 * R0 ;;

/* Это перемножение двух знаковых 32-битных дробных операндов с
фиксированной точкой в регистрах XR1 и XR0; округлённый 32-битный результат
помещается в XR2. */

YR1:0 = R3 * R2 ;;
```

/* Это перемножение двух знаковых 32-битных дробных операндов с фиксированной точкой в регистрах YR3 и YR2; 64-битный результат помещается в YR1:0. */

Примечание – Для операций умножения с фиксированной точкой одиночные имена регистров входных операндов (Rm, Rn) указывают на 32-битные операнды. Одиночные или двойные имена регистров выходных операндов указывают на 32- или 64-битные результаты соответственно.

XR3:2 = R5:4 * R1:0 ;;

/* Это четыре [параллельных] операции умножения (квадроумножение) 16-битных знаковых дробных операндов:

XR5_старшая_часть *XR1_старшая_часть -> XP3_старшая_часть,

XR5_младшая_часть * XR1_младшая_часть -> XR3_младшая_часть,

XR4_старшая_часть * XR0_старшая_часть -> XR2_старшая_часть,

XR4_младшая_часть * XR0_младшая_часть -> XR2_младшая_часть.

YR3:0 = R7:6 * R5:4 ;;

/* Этот пример аналогичен предыдущему примеру 16-битного квадроумножения, но указание квадро-регистра в качестве приёмника приводит к 32-битным результатам; результаты помещаются в регистры YR3, YR2, YR1 и YR0.*/

Примечание – Для операций умножения с фиксированной точкой двойные имена регистров входных операндов (Rmd, Rnd) указывают на 16-битные операнды. Двойные или квадроимена регистров выходных операндов указывают на 16- или 32-битные результаты соответственно.

XFR2 = R1 * R0 ;;

Это операция перемножения двух входных операндов в формате с плавающей точкой одинарной точности в регистрах XR1 и XR0 (формат IEEE); результат одинарной точности помещается в регистр XR2. */

YFR1:0 = R5:4 * R3:2 ::

/* Это операция перемножения двух входных операндов в формате с плавающей точкой расширенной точности в регистрах YR5:4 и YR3:2; 40-битный результат помещается в YR1:0. */

Примечание – Для операций умножения с плавающей точкой одиночные имена регистров входных операндов (Rm, Rn) указывают на операнды и результат одинарной точности (32-битный). Для операций умножения с плавающей точкой двойные имена регистров (Rmd, Rnd, Rsd) входных и выходных операндов указывают на операнды и результат расширенной точности (40-битный) либо двойной точности (выбор типа в регистре управления SQCTL).

Стандартная операция умножения для чисел плавающей запятой одинарной точности выполняет операцию в одном вычислительном модуле только над одной парой чисел. В процессоре добавлена возможность выполнения операции умножения в одном вычислительном модуле над двумя парами чисел ПЗ одинарной точности.

Общая форма записи векторной команды умножения использует два варианта синтаксиса:

Вариант 1 – устаревший, имеет следующую форму записи

{X|Y|XY}FRsd = Rm{H|L} * {-}Rn{H|L}, Rm{L|H} * {-}Rn{L|H} (NF); //

Вариант 2 – является предпочтительным, имеет следующую форму записи

{X|Y|XY}*SFRsd* = *Rmd* * *Rnd* (mX)(nX) (T) (NF); //

Опция mX меняет местами регистры в паре Rmd, а опция nX – в паре Rnd.
При векторном умножении возможны следующие случаи:

- Бит “S” кода команды равен нулю.
  В этом случае две пары операндов перемножаются и формируют два результата. Если значение регистра в поле Rm или Rn нечетное (нечетное значение задается опцией), то это вызывает перестановку операндов в регистровой паре.
- Бит “S” кода команды равен единице.
  В этом случае операнд Rn для обоих умножений будет один и тот же. Его значение определяется кодом Rn и может быть как четным, так и нечетным регистром. В зависимости от того, четное значение регистра Rm или нет, возможна инверсия знака одного из множителей.

Rm	Rn	функция
четное	четное	Умножение каждого операнда регистровой пары Rmd на значение регистра Rn
четное	нечетное	Умножение каждого операнда регистровой пары Rmd на значение регистра Rn
нечетное	четное	Умножение старшего(левого) операнда регистровой пары Rmd на значение регистра -Rn. Умножение младшего (правого) операнда регистровой пары Rmd на значение регистра Rn
нечетное	нечетное	Умножение старшего(левого) операнда регистровой пары Rmd на значение регистра Rn. Умножение младшего (правого) операнда регистровой пары Rmd на значение регистра -Rn

Обращаем внимание, что если Rm нечётное, то операнды меняются местами, а потом уже нужно смотреть в таблицу.

Из таблицы следует, что в случае четного значения регистра Rm, мы имеем возможность умножить оба операнда на общую переменную. Если же значение Rm нечетное, мы получаем специальный вариант умножения с модификацией знака результата. Данный специальный случай позволяет реализовать процедуры комплексного умножения в зависимости от того в каких регистрах расположены действительная и мнимая части комплексного числа.

Второй вариант синтаксиса векторного умножения (см. Вариант 2) является предпочтительным по той причине, что в нём есть возможность использовать константу. Константа пишется вместо регистра Rnd.

Рисунок 73 – 32-битные операции умножителя с фиксированной точкой

Рисунок 74 – 32-битные операции умножителя с плавающей точкой

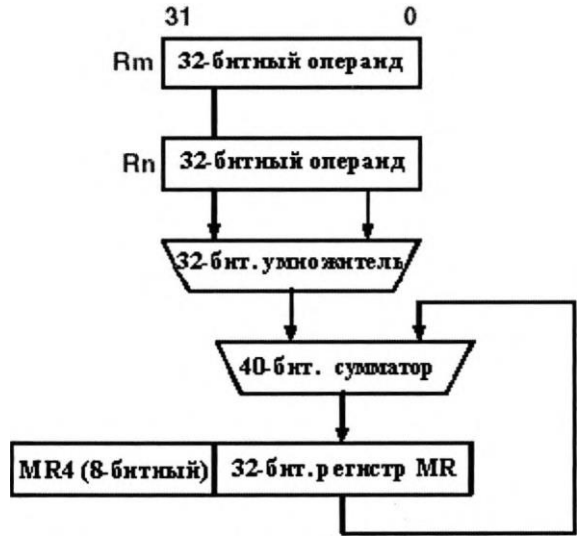


Рисунок 75 – 32-битные операции умножения с накоплением

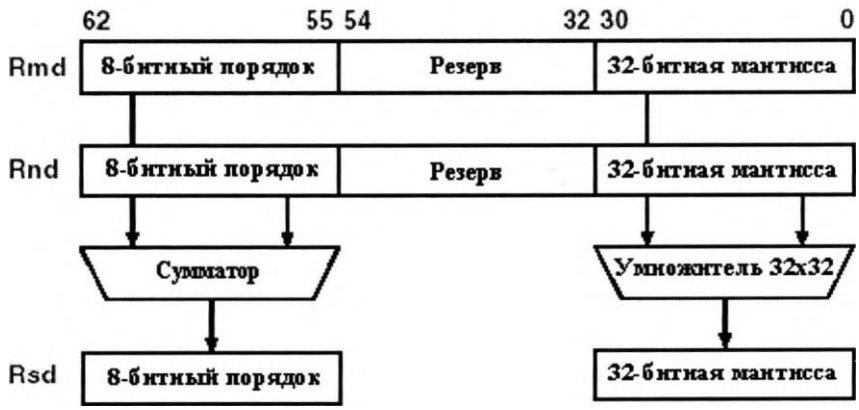


Рисунок 76 – 40-битные операции умножителя

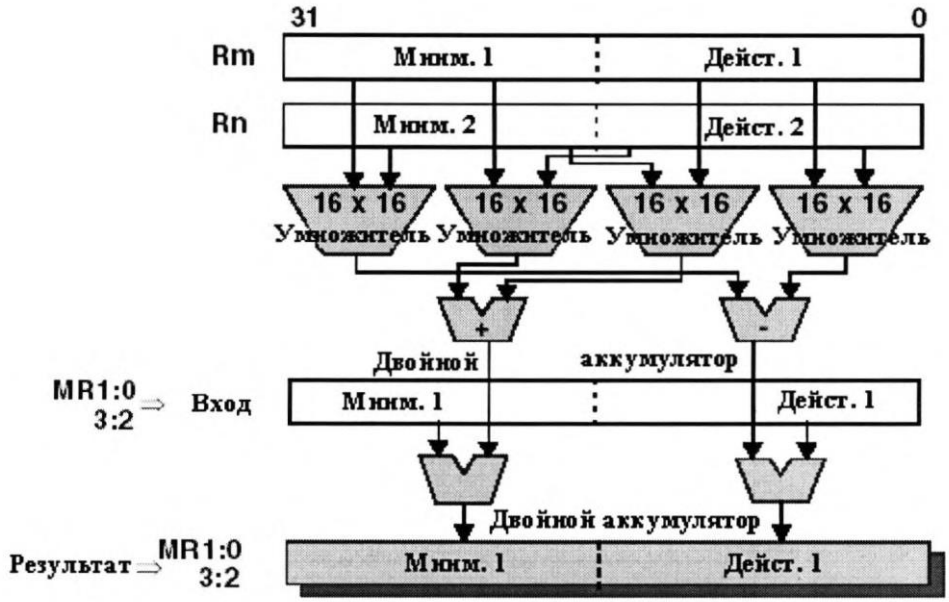


Рисунок 77 – 16-битные операции умножения с комплексными операндами

Рисунок 78 – 16-битные операции умножителя

5.1.1 *Опции команд умножителя*

Большинству команд умножителя сопоставлены опции, позволяющие гибко управлять режимом исполнения данной команды. Важно отметить, что эти опции влияют на то, как именно выполняются команды, а также тот факт, что существуют опции, применимые лишь с определёнными группами команд – не все опции используемы со всеми командами. Опции команд указываются в круглых скобках в конце поля команды.

Список всех опций, используемых с конкретными командами умножителя, см. в разделе «Сводка команд умножителя».

Опции команд умножителя:

- ( ) операция со знаковыми операндами, без насыщения[1], округление к ближайшему чётному[2], с дробными операндами[3];
- (U) операция с беззнаковыми операндами, без насыщения[1], округление к ближайшему чётному[2], с дробными операндами[3];
- (nU) операнд в Rm знаковый, операнд в Rn беззнаковый, без насыщения[1], округление к ближайшему чётному[2], с дробными операндами[3];
- (I) операция со знаковыми операндами, целочисленная[3];
- (S) операция со знаковыми операндами, с насыщением[1];
- (T) операция со знаковыми операндами, с усечением[4];
- (C) операция со сбросом;
- (CR) операция со сбросом/округлением;
- (J) операция с комплексно-сопряжённым;
- (NF) не обновлять флаги.

Следующие примеры представляют собой команды умножителя, демонстрирующие операции умножения и умножения с накоплением с применением опций.

```
XR2 = R1 * R0 (U) ;;
/* Это умножение двух беззнаковых дробных операндов с фиксированной точкой в регистрах XR1 и XR0; округлённый беззнаковый 32-битный результат помещается в XR2.*/
YR1:0 = R3 * R2 (I) ;;
/* Это умножение двух целых 32=битных операндов с фиксированной точкой в регистрах YR3 и YR2; 64-битный результат помещается в YR1:0.*/
XFR2 = R1 * R0 (T) ;;
/* Это умножение двух дробных 32-битных операндов с плавающей точкой в регистрах XR1 и XR0 (в формате IEEE); усечённый 32-битный результат помещается в XR2.*/
```

5.1.1.1 Опция Знаковое/Беззнаковое

Процессор всегда представляет 8-, 16-, 32- или 64-битные числа в формате с фиксированной точкой, используя до четырёх 32-битных регистров данных. Данные умножителя в формате с фиксированной точкой могут быть беззнаковыми или в дополнительном представлении. Данные умножителя в формате с плавающей точкой всегда знаковые величины. Информацию о поддерживаемых числовых форматах см. в разделе «Числовые форматы».

Все команды умножителя с операндами в формате с фиксированной точкой могут использовать как знаковые, так и беззнаковые типы данных. Для этого применяются следующие опции:

- ( ) оба входных операнда знаковые (по умолчанию) ;
- (U) оба входных операнда беззнаковые;
- (nU) операнд в Rm знаковый, в Rn беззнаковый; эта опция применима только к командам Rs = Rm * Rn и Rsd = Rm * Rn.

[1] Там, где насыщение применимо.

[2] Там, где возможно выполнение операции над данными в формате с плавающей точкой.

[3] Там, где возможно выполнение операции над данными в формате с фиксированной точкой.

[4] Там, где усечение применимо.

5.1.1.2 Опция Дробное/Целое

Процессор всегда представляет 8-, 16-, 32- или 64-битные числа в формате с фиксированной точкой, используя до четырёх 32-битных регистров данных. Команды умножения операндов с фиксированной точкой, умножения с накоплением и команда COMPACT умножителя могут быть применены к данным в дробном или целочисленном форматах. Информацию о поддерживаемых числовых форматах см. в разделе «Числовые форматы».

Опция выбора целых или дробных операндов определена для операций с фиксированной точкой:

- ( ) данные дробные (по умолчанию);
- (I) данные целые.

5.1.1.3 Опция Насыщение

Если опция (S) включена[1], то насыщение происходит во время операций с фиксированной точкой, когда результат переполняется - превышает максимальное значение, представимое в выходном формате. В таких случаях в качестве результата возвращается предельное значение, представимое в выходном формате данных операции в соответствии со знаком истинного результата. Например, если выходной формат операции

16-битное знаковое и полный результат операции равен -0x100000, то насыщенным результатом будет 0x8000. Если бы операция была беззнаковая, то результатом был бы 0x0, поскольку отрицательное значение не может получиться в результате беззнаковой операции. Насыщение может иметь место для трёх типов операций:

5.1.1.4 Операции умножения

Если при перемножении целых данных действительный результат операции выходит из диапазона значений, допустимых в выходном формате, то возвращается наибольшее число, представимое в выходном формате. При перемножении дробных данных особым случаем является умножение -1 на -1 (например, 0x80...0 на 0x80...0), в котором при включённом насыщении[1] всегда возвращается наибольшая представимая дробь (например, 0x7F...F).

Операции умножения с накоплением

Насыщение применимо и к целому, и к дробному типу данных. Значения накопления хранятся с 80-, 40- и 20-битной точностью в комбинации регистров MR3:0 и MR4 (см. раздел «Примеры умножителя»). Перед выполнением насыщения в операции умножения с накоплением результат умножителя (64, 32 или 16 битов) добавляется к текущему значению накопления. Если значение накопления не помещается в 80, 40 или 20 бит, оно заменяется на максимальное или минимальное допустимое значение. Отметим, что операции умножения с накоплением всегда производятся с включённым насыщением.

Пересылки регистров MR

См. раздел «Примеры использования умножителя».

Конечный результат с насыщением для 32-битных операндов во всех операциях равен:

- 0x7F...F – если операция знаковая и результат положительный;

[1] За исключением команд *Rsd = Rm * Rn, Rsq = Rmd * Rnd,* в которых насыщение не опционально и всегда подразумевается.

- 0x80...0 – если операция знаковая и результат отрицательный;
- 0xFF...F – если операция беззнаковая и результат положительный;
- 0x00...0 – если операция беззнаковая и результат отрицательный (только для знаковой команды MR - =Rm *Rn).

Опции насыщения имеются для всех команд умножения с фиксированной точкой, в которых может произойти переполнение. Для этого применяются следующие опции:

- ( ) без насыщения (по умолчанию);
- (S) насыщение включено.

5.1.1.5 Опция Усечение

В командах умножителя, поддерживающих опцию (T), с её помощью возможно выбирать режим округления результата. Процессор поддерживает два метода округления – округление к нулю и округление к ближайшему. Эти способы округления соответствуют стандарту IEEE 754 и определяются следующим образом:

- Округление к ближайшему чётному – без использования опции (T), в операциях с операндами в формате с плавающей точкой или дробными операндами в формате с фиксированной точкой. Если результат до округления не может быть точно представлен в требуемом формате, округлённый результат вычисляется как число, ближайшее к результату до округления. Если результат до округления в точности посередине между двумя числами, представимыми в требуемом формате (отличающимися в LSB), округлённый результат вычисляется как число с LSB, равным 0.
- Округление к нулю – с использованием опции (T), в операциях с операндами в формате с плавающей точкой. Если результат до округления не может быть точно представлен в требуемом формате, округлённый результат вычисляется как число, ближайшее к нулю. Это эквивалентно усечению непредставимой части.
- Округление к нулю – с использованием опции (T), в операциях с дробными операндами в формате с фиксированной точкой. Если результат до округления имеет значащие биты ниже LSB для выходного формата операции, то эти биты игнорируются (усекаются). При выполнении этого усечения результат округляется к плюс или минус бесконечности (для положительного результата к меньшему значению; для отрицательного результата к большему отрицательному значению).

Например, пусть умножение с фиксированной точкой даёт 16-битный дробный результат, равный 0x0.1FFFF до усечения. Тогда после усечения, этот положительный результат будет округлён до 0x0.1FFF, что меньше исходного значения. Для сравнения, пусть умножение с фиксированной точкой даёт 16-битный дробный результат, равный 0x. FFFFF (или -220) до усечения. Тогда после усечения этот отрицательный результат будет округлён до 0x. FFFF (или 216), что является большим отрицательным значением.

На практике округление вверх и вниз происходят с одинаковой частотой, поэтому такое округление не даёт систематического смещения. Так как максимальное значение числа в формате с плавающей точкой отличается только в LSB от представления бесконечности, результат операции, лежащий посередине между максимальным значением в формате с плавающей точкой и бесконечностью округляется к бесконечности.

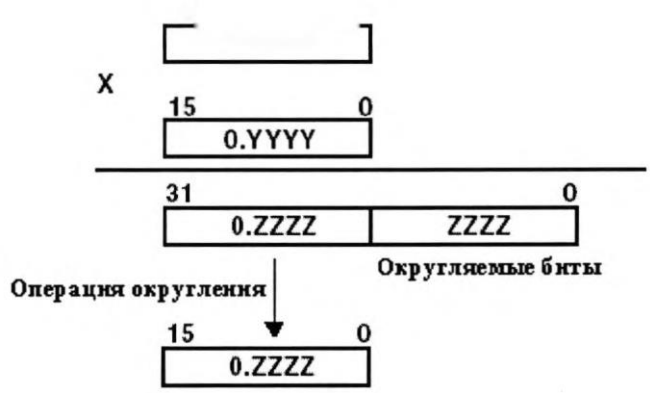


Рисунок 79 – Округление результатов умножителя

Биты 15-0 должны быть использованы для округления, после чего биты 32-16 должны быть возвращены в качестве результата. Округление происходит в зависимости от значений 15-го бита (бит округления), 16-го бита (LSB) и того, равны ли биты 14-0 (младшие биты) нулю или нет:

- если 15-ый бит равен нулю, то результат не инкрементируется;
- если 15-ый бит равен 1 и биты 14-0 ненулевые, LSB увеличивается на единицу;
- если 15-ый бит равен 1 и биты 14-0 равны нулю, то значение 16-го бита прибавляется к результату в битах 31-16.

Режим округления к ближайшему чётному не поддерживается командами умножения с накоплением и пересылкой текущего содержимого регистра MR в регистровый файл. Если имеется необходимость в округлении к ближайшему чётному, то следует переслать регистры MR в регистровый файл полностью и воспользоваться командой ALU COMPACT. В качестве альтернативы, если допустимо округление к ближайшему нечётному, такого эффекта можно достигнуть применением опции сброса с округлением в первой команде умножения с накоплением в серии. Более подробную информацию можно найти в подразделе «Опция Сброс/Округление».

Имеются следующие опции округления:

- ( ) Округление к ближайшему чётному (по умолчанию);
- (T) Усечение - округление к нулю для операндов в формате с плавающей точкой и округление к минус бесконечности для операндов в формате с фиксированной точкой.

5.1.1.6 Опция Сброс/Округление

Операции умножения и умножения с накоплением с пересылкой регистра MR поддерживают опцию (C) сброса MR. Использование этой опции заставляет умножитель обнулить (=0) соответствующую часть регистра MR перед проведением операции накопления.

Операции умножения с накоплением (без пересылки MR) также поддерживают опцию (CR) сброса и округления в качестве альтернативы опции сброса.

Использование опции CR заставляет умножитель обнулить MR и установить бит округления перед проведением операции накопления. Более подробную информацию об округлении и бите округления можно найти в разделе «Опция Усечение». Имеются следующие опции сброса и сброса/округления:

- ( ) никакие изменения не вносятся в содержимое регистра MR до операции умножения с накоплением (по умолчанию);
- (C) установить используемые регистры MR в ноль до операции умножения

с накоплением;
- (CR) установить используемые регистры MR в ноль и установить бит округления до операции умножения с накоплением. Эта опция CR может быть использована только в операциях дробной арифметики, например, когда опция (I) не указана.

Если эта опция включена, регистры MR устанавливаются равными 0x00000000 80000000 для 32-битных дробных операций умножения с накоплением и 0x00008000 в каждом регистре MR для 16-битной дробной операции квадроумножения с накоплением. После такой инициализации результат может быть округлён с избытком с помощью операции сохранения старшей части результата [накопления в регистровом файле] в конце серии команд умножения с накоплением.

В качестве примера для опции CR рассмотрим серию из трёх операций короткословных дробных квадроумножений с накоплением (с четырьмя короткими результатами), такую что результатами умножений будут:
- Результат 1= 0x0024 0048, 0x06294501
- Результат 2= 0x01280128, 0x24702885
- Результат 3= 0x1011fffe, 0x4A306d40
- Сумма = 0x115d016e, 0x74c9dac6

В этом примере младшие 16 бит не будут использоваться, так как ожидается короткий результат. Извлечение старших 16 битов даст усечённый результат, равный 0x115d для первого короткого слова и 0x74c9 для второго. Округлённый результат равен 0x115d для первого короткого слова (не изменился) и 0x74ca для второго (инкрементировался). Если регистры MR были инициализированы значениями 0x00008000, то сумма получилась бы:

Сумма = 0x115d 816e, 0x74ca 5ac6

Полученные два коротких слова в точности совпадают с нужным результатом. Заметим, что это округление к ближайшему, а не округление к ближайшему чётному.

Старшие короткие слова в точности равны ожидаемым значениям. Метод округления – к ближайшему, а не к ближайшему чётному. Информацию об округлении см. в разделе «Опция Усечение».

5.1.1.7 Опция Комплексное сопряжение

В операциях комплексного умножения с накоплением (оператор **) умножитель поддерживает опцию (J) комплексного сопряжения. Опция J предписывает умножать операнд в *Rm* на комплексно сопряжённое значение к значению операнда в *Rn,* у которого знак мнимой части инвертирован. Имеются следующие опции:
- ( ) без сопряжения;
- (J) сопряжение в комплексном умножении.

5.1.1.8 Опция Не-обновлять-флаги

Почти все вычислительные операции влияют на значение регистра состояния вычислительного блока в соответствии со своим статусом завершения. Зачастую в некоторых приложениях оказывается полезным защитить значение регистра состояния от изменения в результате выполнения операции. Существует множество способов сохранения слова состояния (например, сохранение содержимого регистра состояния или загрузка значений флагов состояния в регистр статических флагов (SFREG)). Опция Не-обновлять-флаги (NF) предоставляет другой способ работы со словом состояния. Вместо того, чтобы сохранять статус завершения операции для дальнейшего использования, программа может запретить переписывать слово

состояния, используя опцию (NF) со следующей командой, что предотвращает генерирование той статуса завершения. Существуют некоторые ограничения на то, каким образом опция (NF) может быть использована в условных командах. Более подробно см. раздел «Отработка условных команд».

Регистр переполнения результата умножителя (MR4)

Регистр MR4 содержит дополнительные биты (переполнение) в операциях умножения с накоплением. Поля регистра MR4 распределяются в соответствии с используемыми регистрами MR и размером результата. См. рисунки ниже:

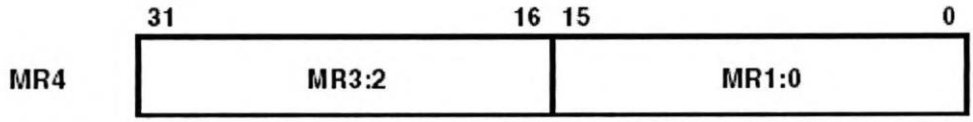


Рисунок 80 – MR4 для результата в виде длинного слова (80-битное значение накопления)

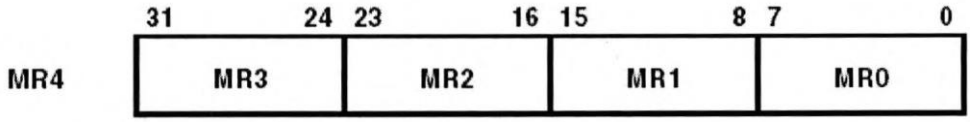


Рисунок 81 – MR4 для результата в виде обычного слова (40-битное значение накопления)

Рисунок 82 – MR4 для результата в виде короткого слова (20-битное значение накопления)

Эти биты также используются как входные данные на шаге накопления операции умножения с накоплением. Биты сбрасываются вместе со сбросом соответствующего регистра MR, а при сохранении учитываются при насыщении. Назначение этих битов – сделать возможным, чтобы промежуточные результаты серии операций умножения с накоплением могли выходить за диапазон значений, допустимых для конечного результата.

5.1.2 *Статус выполнения умножителя*

Операции умножителя изменяют флаги состояния арифметических регистров состояния вычислительных блоков (XSTAT и YSTAT). Программы могут использовать флаги состояния для управления исполнением условных команд и порождения программных исключений. Более подробно см. раздел «Предикаты умножителя». В таблице 12 приведены флаги регистров XSTAT и YSTAT, отражающие статус завершения последней операции умножителя.

Таблица 12 – Флаги состояния умножителя

Флаг	Определение	Изменяется...
MZ	Признак равенства нулю в операции умножителя с фиксированной точкой или потери значимости в операции с плавающей точкой	Всеми операциями умножителя с фиксированной и плавающей точкой, за исключением операций умножения с накоплением
MN	Признак отрицательного значения в операции умножителя	Всеми операциями умножителя с фиксированной и плавающей точкой, за исключением операций умножения с накоплением
MV	Признак переполнения[1] в операции умножителя	Всеми операциями умножителя с фиксированной и плавающей точкой, за исключением операций умножения с накоплением
MU	Признак потери точности[2] в операции умножителя	Всеми операциями умножителя с плавающей точкой; очищается операциями с фиксированной точкой; не изменяется операциями умножения с накоплением
MI	Признак некорректной операции[3] с плавающей точкой умножителя	Всеми операциями умножителя с плавающей точкой; очищается операциями с фиксированной точкой; не изменяется операциями умножения с накоплением

[1] Если бит OEN регистра X/YSTAT установлен, то флаг MV порождает программное исключение.
[2] Если бит UEN регистра X/YSTAT установлен, то флаг MU порождает программное исключение.
[3] Если бит IVEN регистра X/YSTAT установлен, то флаг MI порождает программное исключение.

Операции умножителя также изменяют залипающие флаги состояния арифметических регистров состояния вычислительных блоков (XSTAT и YSTAT). В таблице 13 приведены флаги регистров XSTAT и YSTAT, отражающие залипающий статус завершения последней операции умножителя. Установленный однажды, залипающий флаг остаётся в этом положении до тех пор, пока не будет сброшен явно.

Обновление флагов происходит в конце выполнения каждой операции, флаги доступны с задержкой в один такт; что делает их доступными уже во второй командной строке после командной строки, сгенерировавшей статус. Программа не может записать значение в регистр XSTAT или YSTAT явно в том же командном такте, в котором происходит обновление слова состояния.

Таблица 13 – Залипающие флаги состояния умножителя

Флаг	Определение	Изменяется...
MUS	Признак потери значимости в операции умножителя с плавающей точкой, залипающий[1]	Всеми операциями с плавающей точкой
MVS	Признак переполнения в операции умножителя с плавающей точкой, залипающий[2]	Всеми операциями с плавающей точкой
MOS	Признак переполнения в операции умножителя с фиксированной точкой, залипающий[2]	Всеми операциями с фиксированной точкой

Флаг	Определение	Изменяется...
MIS	Признак некорректной операции с плавающей точкой умножителя, залипающий[3]	Всеми операциями с плавающей точкой

[1] Если бит UEN регистра X/YSTAT установлен, то флаг MUS порождает программное исключение.
[2] Если бит OEN регистра X/YSTAT установлен, то флаги MVS или MOS порождают программное исключение.
[3] Если бит IVEN регистра X/YSTAT установлен, то флаг MIS порождает программное исключение.

Команды со множественными операндами (например, *Rsd = Rmd * Rnd*) порождают несколько результатов. В этом случае процессор вычисляет флаг состояния как битовое ИЛИ соответствующих флагов для каждого результата.

Примечание – При использовании опции (NF) вычислительное устройство не обновляет значения флагов состояния в соответствии со статусом завершения операции. Разрешено использовать опцию (NF) со всеми командами умножителя, за исключением команд пересылки регистра MR4.

5.1.3 *Предикаты умножителя*

В условных командах умножителя исполнение всей командной строки зависит от значения предиката, указанного в начале этой командной строки. Условные команды умножителя имеют вид:

IF cond; DO, instr; DO, instr; DO, instr. ;;

Такой синтаксис позволяет подчинить одному предикату до трёх условных команд. Пропуск DO перед командой делает её безусловной.

В таблице 14 приведён список предикатов умножителя. Более подробную информацию об условных командах см. в разделе «Отработка условных команд».

Таблица 14 – Предикаты умножителя

Предикат	Описание	Значения флагов
MEQ	Равно нулю в умножителе	MZ=1
MLT	Строго меньше нуля в умножителе	MN=1 и MZ=0
MLE	Меньше или равно нулю в умножителе	MN=1 или MZ=1
NMEQ	НЕ (Равно нулю в умножителе)	MZ=0
NMLT	НЕ (Строго меньше нуля в умножителе)	MN=0 или MZ=1
NMLE	НЕ (Меньше или равно нулю в умножителе)	MN=0 и MZ=0

Так как статус умножителя становится доступным с задержкой в один такт (становится доступным во второй командной строке после командной строки, сгенерировавшей статус), то использование условий умножителя в такте, следующем после генерации статуса, вызывает приостановку на один цикл.

5.1.4 *Статические флаги умножителя*

В устройстве управления регистр статических флагов (SFREG) может хранить значения флагов состояния для их дальнейшего использования в условных командах. В регистре SFREG каждому вычислительному блоку выделено два бита флагов: X/YSCF0 (предикаты XSF0, YSF0, XYSF0 или SF0) и X/YSCF1 (предикаты XSF1, YSF1, XYSF1 или SF1).

Следующий пример демонстрирует загрузку значения предиката вычислительного блока в регистр статических флагов. Более подробно о статических флагах приведено в разделе «Отработка условных команд».

```
XSF0 = XMEQ ;;
/* Загрузить значение предиката MEQ вычислительного блока X в бит XSCF0 регистра статических флагов (SFREG)*/
IF XSF0, DO XR5 = R4 * R3 ;;
/* предикат XSF0 истинен, если установлен статический флаг XSCF0 */
```

5.2 Примеры использования умножителя

В листинге 10 приведено несколько примеров команд умножения и умножения с накоплением. Ключевые особенности каждой команды, такие как формат операндов с фиксированной или плавающей точкой, размер операндов, а также использование регистров, перечислены в комментариях к команде.

Листинг 10 – Примеры команд умножителя

```
XYR4 = R6 * R8 ;;
/* Это команда 32-битного дробного умножения на обоих вычислительных блоках с 32-битным округлённым результатом. */
XYR5:4 = R6 * R8 ;;
/* Это команда 32-битного дробного умножения на обоих вычислительных блоках с 64-битным результатом. */
XR11:10 = R9:8 * R7:6 ;;
/* Это команда 16-битного квадроумножения; входные операнды перемножаются по схеме XR9_H * XR7_H, XR9_L * XR7_L, XR8_H * XR6_H и XR8_L * XR6_L (здесь _H обозначает старшую часть, а _L - младшую часть); 16- битные результаты сохраняются в XR11_H, XR11_L, XR10_H и XR10_L (соответственно).*/
XMR3:2 += R1 * R0 ;;
/* Это перемножение входных операндов из XR1 и XR0 с добавлением результата умножения к текущему содержимому указанных регистров XMR, при этом переполнение сохраняется в XMR4_H. */
YMR1:0 -= R3 * R2 ;;
/* Это перемножение входных операндов из YR3 и YR2 с вычитанием результата умножения из текущего содержимого указанных регистров YMR, при этом переполнение сохраняется в YMR4_L. */
XR7 = MR3:2, MR3:2 += R1 * R0 ;;
/* Эта команда выполняет операцию умножения с накоплением и пересылает предыдущее содержимое регистров MR в регистровый файл. */
YMR3:0 += R5:4 * R7:6 ;;
/* Эта команда представляет собой четыре умножения четырёх 16-битных коротких слов в регистровой паре YR5:4 и четырёх 16-битных коротких слов в регистровой паре YR7:6. Четыре результата накапливаются в MR3:0 в виде обычных слов. Биты переполнения записываются в MR4. */
XMR3:2 += R9:8 * R7:6 ;;
/* Это команда 16-битного квадроумножения с накоплением с 16-битными результатами; входные операнды перемножаются по схеме XR9_H * XR7_H, XR9_L * XR7_L, XR8_H * XR6_H и XR8_L * XR6_L (здесь _H обозначает старшую часть, а _L - младшую часть); 16-битные результаты накопления сохраняются в XMR3_H, XMR3_L, XMR2_H и XMR2_L (соответственно). */
MR3:0 += R9:8 * R7:6 ;;
/* Это команда 16-битного квадроумножения с накоплением с 32-битными результатами; входные операнды перемножаются по схеме XR9_H * XR7_H,
```

XR9_L * XR7_L, XR8_H * XR6_H и XR8_L * XR6_L (здесь _H обозначает старшую часть, а _L - младшую часть); 32-битные результаты накопления сохраняются в XMR3, XMR2, XMR1 и XMR0 (соответственно). */
XMR1:0 += R9 ** R7 ;;
/* Это команда перемножения комплексного значения из XR9 на комплексное значение из XR7. Результат накапливается в XMR1:0. */
XFR20 = R22 * R23 (T) ;;
/* Это команда 32-битного (с одинарной точностью) плавающего умножения с 32-битным результатом; использование одиночных регистров указывает на 32-битную операцию. */
YFR25:24 = R27:26 * R30:29 (T) ;;
/* Это команда 40-битного (с расширенной точностью) плавающего умножения с 40-битным результатом; использование двойных регистров указывает на 40-битную операцию. Аналогичная команда может указывать на операцию плавающего умножения операндов двойной точности */

5.3 Сводка команд умножителя

Следующие листинги демонстрируют синтаксис команд умножителя:

- Листинг 11 «Команды умножения 32-битных операндов с фиксированной точкой»;
- Листинг 12 «Команды квадроумножения 16-битных операндов с фиксированной точкой»;
- Листинг 13 «Команды комплексного умножения 16-битных операндов с фиксированной точкой»;
- Листинг 14 «Команды умножения операндов с плавающей точкой»;
- Листинг 15 «Команды загрузки регистров умножителя»;
- Листинг 16 «Команды векторного умножения чисел одинарной точности».

Соглашения по обозначениям имён регистров, необязательных элементов или вариантов, используемые в этих листингах, подробно освещены в разделе «Регистры регистровых файлов». Вкратце эти соглашения таковы:

- {} - необязательные элементы указаны в фигурных скобках; фигурные скобки не являются частью синтаксиса команды;
- | - вертикальная черта разделяет варианты значений элементов; вертикальные черты не являются частью синтаксиса команды;
- Rmd - имена регистров, выделенные курсивом, представляют собой подставляемые шаблоны для имён одиночных (Rs, Rm или Rn), двойных (Rsd, Rmd или Rnd) или квадро- (Rsq, Rmq или Rnq) регистров.

Регистры MR3:0 представляют собой четыре 32-битных регистра накопления. Для размещения значений переполнения операций накопления используется регистр MR4, в котором могут храниться два 16-битных значения переполнения в случае 32-битных множителей, четыре 8-битных значения переполнения в случае двух пар 16-битных множителей или восемь 4-битных значения переполнения в случае четырёх пар коротких 16-битных множителей.

Листинг 11 – Команды умножения 32-битных операндов с фиксированной точкой

{X|Y|XY}R s = Rm * Rn {({U|nU}{I|T}{S}{NF})}[1]
{X|Y|XY}Rsd = Rm * Rn {({U|nU}{I}{NF})} ;

[1] Допустимые опции: (): дробное, знаковое, без насыщения, (S): с насыщением, знаковое, (SU): с насыщением, беззнаковое.

```
{X|Y|XY}MRa += Rm * Rn {({U}{I}{C|C R}{NF})} ;[1]
{X|Y|XY}MRa -= Rm * Rn {({I}{C|CR}{NF})} ;
{X|Y|XY}Rs = MRa, MRa += Rm * Rn {({U}{I} {C|CR}{NF})} ; (двойная операция)
{X|Y|XY}Rsd = MRa, MRa += Rm * Rn { ({U}{I}{C}{NF})} ; (двойная операция)
/* здесь MRa либо MR1:0, либо MR3:2 */
```

Листинг 12 – Команды квадроумножения 16-битных операндов с фиксированной точкой

```
{X|Y|XY}Rsd = Rmd * Rnd {({U}{I|T}{ S}{NF})} ;
{X|Y|XY}Rsq = Rmd * Rnd {({U}{I}{NF})} ;
{X|Y|XY}MR3:0 += Rmd * Rnd {({U}{I}{C|CR}{NF})} ;
{X|Y|XY}MRb += Rmd * Rnd {({U}{I}{C}{ NF})} ;
{X|Y|XY} Rsd =MRb, MR3:0 += Rmd * Rnd { ( {I} {C | CR}{NF})},* (двойная операция)
{X|Y|XY}Rsd = MRb, MRb += Rmd * Rnd {({I}{C} {NF))}     ; (двойная операция)
/* здесь MRb либо MR1:0, либо MR3:2 */
```

Листинг 13 – Команды комплексного умножения 16-битных операндов с фиксированной точкой

```
{X|Y|XY}MRa += Rm ** Rn {({I}{ C|CR}{J}{NF} )} ;
{X|Y|XY}MRa -= Rm ** Rn {({I}{C|CR} {J}{NF})} ;
{X|Y|XY}Rs = MRa, MRa += Rm ** Rn { ({I} {C |CR} {J} {NF}) } ; (двойная операция)
{X|Y|XY}Rsd = MRa, MRa += Rm ** Rn { ({I} {C} {J} {NF} ) } ; (двойная операция)
/* здесь MRa либо MR1:0, либо MR3:2 */
```

Листинг 14 – Команды умножения операндов с плавающей точкой

```
{X|Y|XY}FRs = Rm * Rn {({T}(NF})} ;
{X|Y|XY}FRsd = Rmd * Rnd {({T}{ NF})} ;
```

Листинг 15 – Команды загрузки регистров умножителя

```
{X|Y|XY}{S|L}MRa = Rmd {({SE|ZE}{NF})} ;
{X|Y|XY}MR4 = Rm {(NF)} ;
{X|Y|XY}{S}Rsd = MRa {({U}{S}{NF})} ;
{X|Y|XY}Rsq = MR3:0 { ({U}{S}{NF}) } ;
{X|Y|XY}Rs = MR4 ;
/* здесь MRa либо MR1:0, либо MR3:2 */
{X|Y|XY}Rsd = SMRb {({U}{NF})} ; /* извлечь 2 коротких слова */
{X|Y|XY}LRsd = MRb {({U}{NF})} ; /* извлечь 1 обычное слово */
/* здесь MRb равен MR0, MR1, MR2 или MR3 */
{X|Y|XY}Rsq = SMRa {({U}{NF})} ; /* извлечь 4 коротких слова */
{X|Y|XY}LRsq = MRa {({U}{NF})} ; /* извлечь 2 обычных слова */
{X|Y|XY}QRsq = LMRa {({U}{NF})} ; /* извлечь 1 длинное слово из MRa*/
/* здесь MRa либо MR1:0, либо MR3:2 */
{X|Y|XY}Rs = COMPACT MRa {({U}{I}{S} {NF})} ;
{X|Y|XY}SRsd = COMPACT MR3:0 {({U}{I}{S} {NF})} ;
/* здесь MRa либо MR1:0, либо MR3:2 */
```

Листинг 16 – Команды векторного умножения чисел одинарной точности

```
{X|Y|XY}SFRsd = Rmd * Rnd (mX)(nX)(T) (NF); // (Вариант 2)
{X|Y|XY}FRsd = Rmd * Rnd {({T}{NF})} ; // (Вариант 1)
```

[1] Допустимые опции: (): знаковое, округление к ближайшему чётному, (T): знаковое, с усечением, (U): беззнаковое, округление к ближайшему чётному, (TU): беззнаковое, с усечением.

{X|Y|XY}FRsd = RmH * RnH, RmL * RnL {({T}{NF})} ; // (Вариант 1)
{X|Y|XY}FRsd = RmH * RnL, RmL * RnH {({T}{NF})} ; // (Вариант 1)
{X|Y|XY}FRsd = RmL * RnH, RmH * RnL {({T}{NF})} ; // (Вариант 1)
{X|Y|XY}FRsd = RmL * RnL, RmH * RnH {({T}{NF})} ; // (Вариант 1)
{X|Y|XY}FRsd = RmL * RnL, RmH * RnH {({T}{NF})} ; // (Вариант 1)
{X|Y|XY}FRsd = RmH * -Rn, RmL * Rn {({T}{NF})} ; // (Вариант 1)
{X|Y|XY}FRsd = RmH * Rn, RmL * -Rn {({T}{NF})} ; // (Вариант 1)

6 Сдвиговое устройство

Ядро процессора содержит два вычислительных устройства, известных как вычислительные блоки. Каждый вычислительный блок содержит регистровый файл и четыре независимых вычислительных устройства: ALU, CLU, умножитель и сдвиговое устройство. Сдвиговое устройство (shifter) выделено на рисунке 83. Сдвиговое устройство берёт входные данные из регистрового файла и сохраняет выходные данные также в регистровом файле.

В данной главе описаны:

- «Операции сдвигового устройства»;
- «Примеры использования сдвигового устройства»;
- «Сводка команд сдвигового устройства».

Сдвиговое устройство выполняет *побитовые операции* (арифметические и логические сдвиги) и *операции с битовыми полями* (извлечение и размещение полей) процессора. Операциями сдвигового устройства являются:

- сдвиг и циклический сдвиг битового поля влево и вправо на значения, которые могут превышать длину битового поля;
- битовые операции; установка, сброс, переключение и проверка бита;
- операции с битовыми полями; извлечение и размещение полей;
- определение масштабирующего множителя в 16-битной блочной плавающей арифметике;
- извлечение порядка;
- подсчёт числа ведущих единиц или нулей.

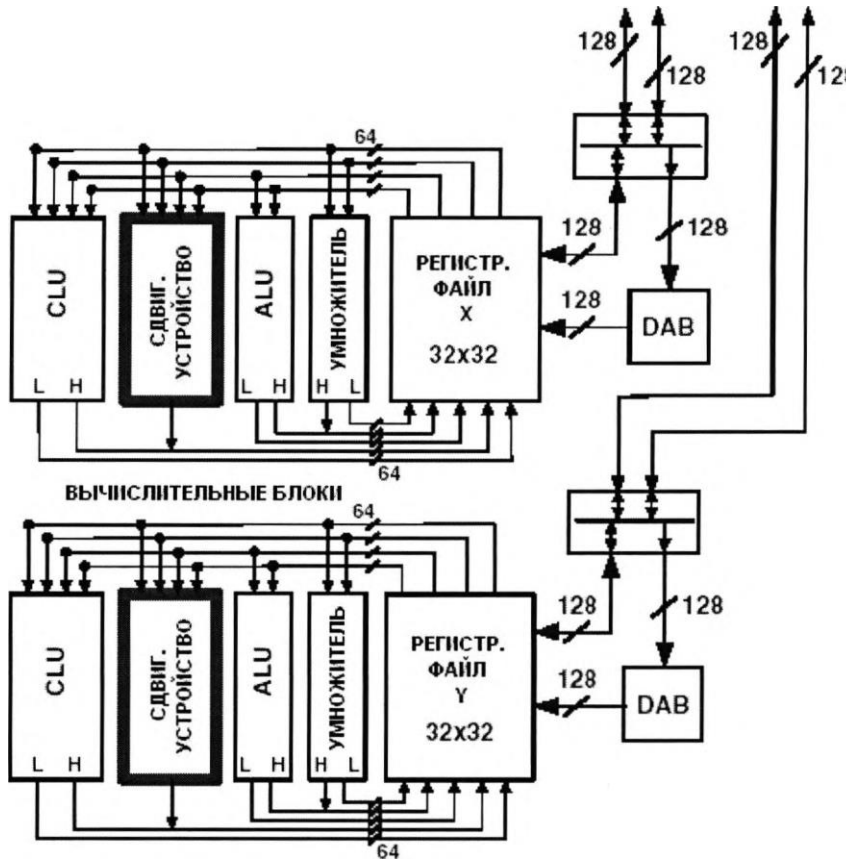


Рисунок 83 – Сдвиговые устройства вычислительных блоков X и Y

Сдвиговое устройство оперирует данными в формате с фиксированной точкой и принимает на входе следующие типы данных:

- один операнд в виде длинного слова (64 бита);
- один или два операнда в виде обычного слова (32 бита);
- два или четыре операнда в виде короткого слова (16 бит);
- четыре или восемь однобайтных (8 бит) операндов.

Как видно из рисунка 83, сдвиговое устройство имеет четыре входа (соответствуют трем операндам) и один 64-битный выход (в отличие от ALU и умножителя, которые имеют два входа и выхода). Способ подключения магистралей ввода-вывода к сдвиговому устройству внутри вычислительных блоков накладывает ряд ограничений на возможности запараллеливания команд.

Команды сдвигового устройства, использующие все входы, не могут быть выполнены в параллель ни с какой командой вычислительных блоков

В командах FDEP, MASK, GETBITS и PUTBITS на входы сдвигового устройства передаются три регистра.

Правила именования регистров определяют размер и тип данных входных операндов и результата в командах. Более подробную информацию о выборе размера и типа данных в командах сдвигового устройства можно найти в разделе «Регистры регистровых файлов».

Оставшаяся часть этой главы посвящена описанию команд сдвигового устройства, их опций и результатов с использованием синтаксиса команд. Объяснение соглашений по синтаксису команд сдвигового и других устройств, можно найти в разделе «Синтаксис и структура командной строки». Список команд сдвигового устройства и их синтаксис приведён в разделе «Сводка команд сдвигового устройства».

6.1 Операции сдвигового устройства

Сдвиговое устройство может использовать один 64-битный, один или два 32-битных, два или четыре 16-битных и четыре или восемь 8-битных операнда в формате с фиксированной точкой. Операциями сдвигового устройства являются:

- сдвиг и циклический сдвиг битового поля влево и вправо на значения, которые могут превышать длину битового поля;
- битовые операции, включая установку, сброс, переключение и проверку бита;
- операции с битовыми полями, включая извлечение и размещение полей с использованием регистра BFOTMP (внутреннего для сдвигового устройства);
- операции с однобитным FIFO для поддержки битовых потоков с полями переменной длины;
- поддержка операций преобразования форматов с фиксированной и плавающей точкой, (таких как извлечение порядка, подсчёт числа ведущих единиц или нулей).

Сдвиговое устройство использует данные из регистровых файлов вычислительных блоков и регистра BFOTMP, внутреннего регистра сдвигового устройства, используемого командой PUTBITS. В операциях сдвигового устройства входные данные для *Rm* (данные, над которыми производится операция) могут быть взяты из регистрового файла, входные данные для *Rn* (величины сдвига) могут быть взяты как из регистрового файла, так и из поля непосредственных данных в командной строке. В случаях, когда в операции участвует третий входной операнд, входные данные для *Rm* и *Rn* берутся из регистрового файла, а доступ к третьему операнду *Rs,* осуществляется по схеме чтение-модификация-запись (RMW).

В операциях с регистровым файлом, то есть когда величины сдвига берутся из *Rn,* значения этих величин хранятся в крайних правых битах *Rn.* Размер величины сдвига (количество бит) зависит от размера результата: для результата в виде длинного слова используются 8 бит *Rn,* для результата в виде обычного слова используются 7 бит, для результата в виде короткого слова используются 6 бит и для однобайтного результата используются 6 бит. Таким образом, доступны полномасштабные сдвиги вправо и влево. Биты *Rn* вне поля величины сдвига не используются.

В следующих разделах можно найти подробности об операциях сдвигового устройства:

- «Операция логического сдвига»;
- «Операция арифметического сдвига»;
- «Битовые операции»;
- «Операции с битовыми полями»;
- «Операции анализа битовых полей»;
- «Операции с битовыми потоками».

6.1.1.1 Операция логического сдвига

Следующая команда приведена в качестве примера операции логического сдвига (LSHIFT). Операция производит сдвиг содержимого регистра XR4 на величину сдвига (количество бит), указанную в регистре XR3. Результат помещается в регистр XR5. На рисунке 84 показано, как биты в регистре XR5 перемещаются в результате сдвигов на 4 (первый пример) и на -4 (второй пример).

XR5 = LSHIFT R4 BY R3;;

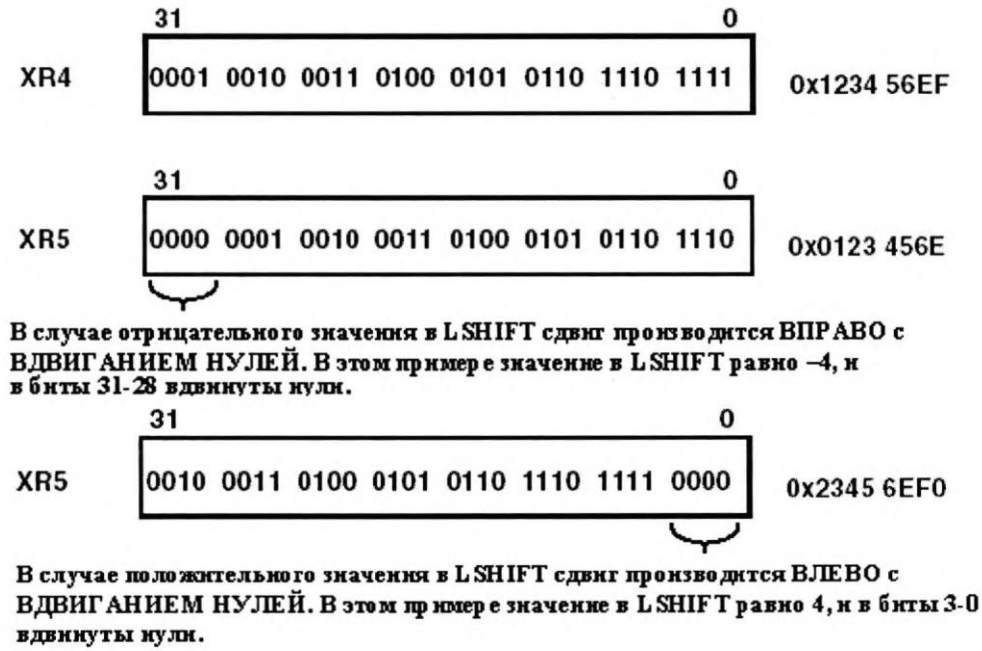


Рисунок 84 – Пример команды LSHIFT

6.1.1.2 Операция арифметического сдвига

Следующая команда приведена в качестве примера операции логического сдвига (ASHIFT). Операция производит сдвиг содержимого регистра XR4 на величину сдвига (количество бит), указанную в регистре XR3. Результат помещается в

регистр XR5. На рисунке 85 показано, как биты в регистре XR5 перемещаются в результате сдвигов на 8 (первый пример) и на минус 8 (второй пример).

XR5 = ASHIFT R4 BY R3 ;;

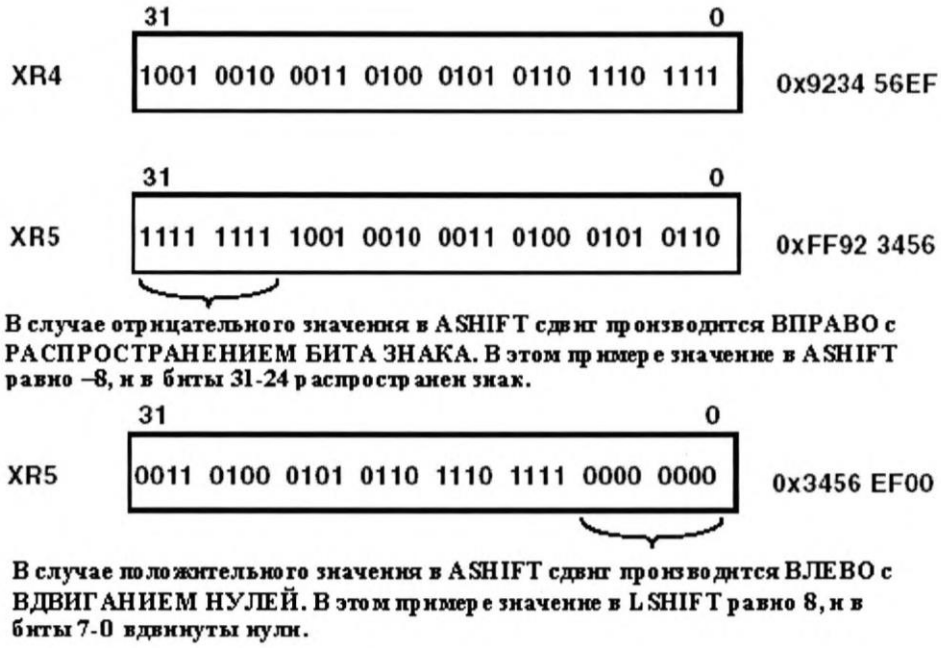


Рисунок 85 – Пример команды ASHIFT

6.1.1.3 Битовые операции

Сдвиговое устройство поддерживает битовые операции, включая сброс бита (BCLR), установку бита (BSET), переключение бита (BTGL) и проверку бита (BITEST). Допустимы операнды в виде простого или длинного слова. Например:

R5 = BCLR R3 By R2 ;; /* 32-битные операнды */

R5: 4 = BSET R3:2 By R6 ;; /* 64-битные операнды */

Следующая команда приведена в качестве примера битовой операции (BCLR). Операция сбрасывает бит в регистре XR4 с номером, указанным в регистре XR3. Результат помещается в регистр XR5. На рисунке 86 показано, как биты в регистре XR5 изменяются в результате операции над 8-ым битом.

XR5 = BCLR R4 By R3 ;;

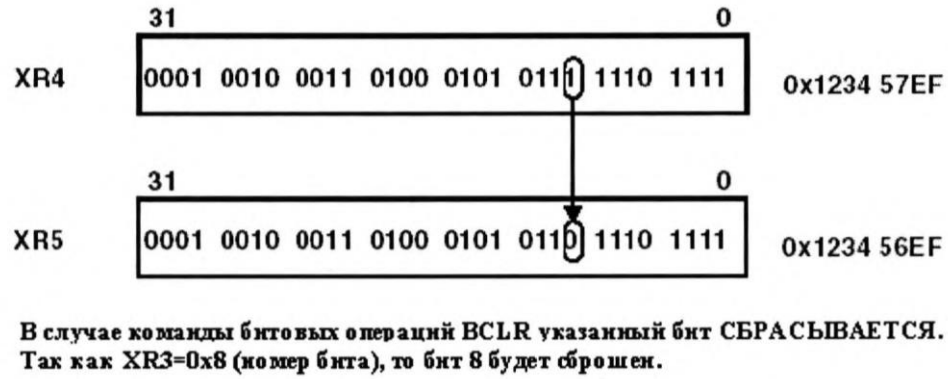


Рисунок 86 – Пример команды BCLR

6.1.1.4 Операции с битовыми полями

Сдвиговое устройство поддерживает операции с битовыми полями, включая:

- FEXT – извлечение поля из регистра, определённого своей длиной и позицией, указанными в другом регистре;
- FDEP – размещение в регистре выровненного по правому краю поля, определённого своей новой длиной и позицией, указанными в другом регистре;
- MASK – копирование 32- или 64-битных полей, определённых маской;
- XSTAT/YSTAT – загрузка или сохранение всех или только 14-ти LSB регистра XSTAT или YSTAT.

В командах извлечения и размещения полей операнд *Rm* содержит управляющую информацию в виде двух полей: Len7 и Pos8. В этих полях указывается количество извлекаемых битов (Len7) и начальная позиция поля в *Rm* (Pos8). Как показано на рисунке 87, расположение упомянутых полей в регистре зависит от того, какой указан регистр *Rn:* одиночный или двойной.

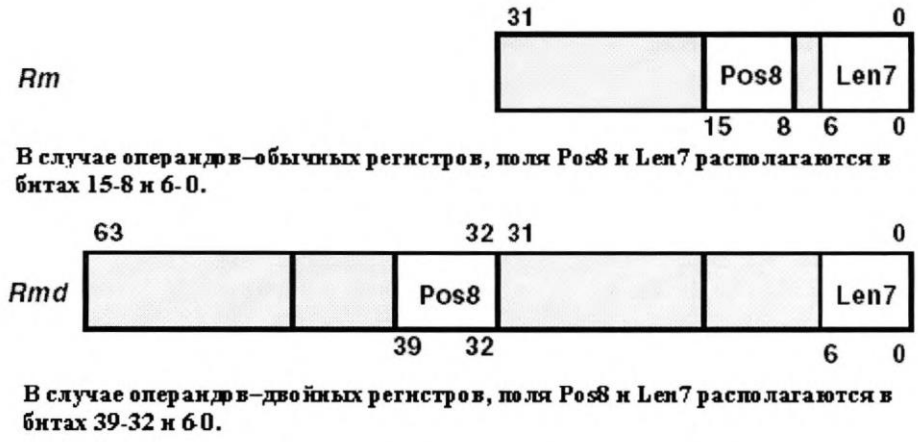
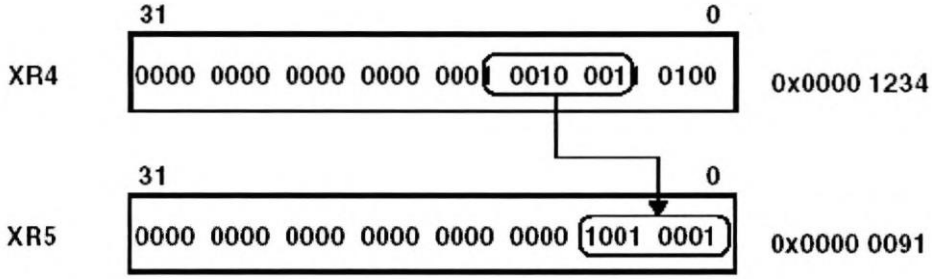


Рисунок 87 – Поля Len7 и Pos8 в командах FEXT и FDEP

Имеется две разновидности команд FEXT и FDEP. Первая из них получает управляющую информацию из регистровой пары. Вторая получает управляющую информацию из одиночного регистра. Команда FEXT извлекает битовые данные, начиная с указанной позиции в регистре источника, и помещает их, выровняв по правой границе, в регистр приёмника *(Rs).* Команда FDEP извлекает битовые данные, выровненные по правой границе, из регистра источника, и помещает их, начиная с указанной позиции, в регистр приёмника *(Rs).*

Следующая команда приведена в качестве примера операции извлечения битовых полей (FEXT). Сдвиговое устройство извлекает битовое поле из регистра XR4, определённое значениями позиции (Pos8) и длины (Len7) в регистре XR3. Результат, выровненный по правой границе, помещается в XR5. По умолчанию операция заполняет нулями неиспользуемые биты регистра приёмника (в примере - XR5). Если в команде FEXT указана опция распространения знака (SE), то значение старшего бита (MSB) извлекаемого поля распространяется на все неиспользуемые биты регистра приёмника. На рисунке 88 показано, как биты в регистре XR5 изменяются в результате операции над полем с позицией Pos8=5 и длиной Len7=8.

```
XR5 = FEXT R4 BY R3 ;; /* Pos8=5, Len7=8, XR3=0x0000 0508 */
```

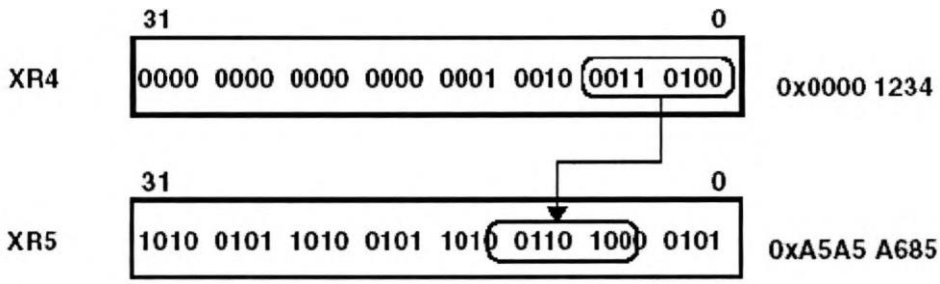


В команде извлечения битового поля FEXT неиспользуемые биты в регистре приемника будут СБРОШЕНЫ, если не указана опция SE. В этом примере биты 31-8 будут сброшены.

Рисунок 88 – Пример команды FEXT

Следующая команда приведена в качестве примера операции размещения битовых полей (FDEP). Сдвиговое устройство извлекает выровненное по правой границе битовое поле из регистра XR4 длины (Len7), определённой в регистре XR3. Результат помещается в XR5, начиная с указанной позиции (Pos8). По умолчанию операция не меняет неиспользуемые биты регистра приёмника (в примере – XR5). Если в команде FDEP указана опция распространения знака (SE), то значение старшего бита (MSB) извлекаемого поля распространяется на все неиспользуемые старшие биты регистра приёмника. Если в команде FDEP указана опция заполнения нулями (ZF), все неиспользуемые старшие биты регистра приёмника устанавливаются равными 0. На рисунке 89 показано, как биты в регистре XR5 изменяются в результате операции над полем с позицией Pos8=5 и длиной Len7=8.

```
XR3 = 0x00000508;;
XR4 = 0x00001234;;
XR5 = 0xA5A5A5A5;;  /* до команды fdep */
XR5 += FDEP R4 BY R3;;
/* После команды fdep XR5 = 0xA5A5A685 */
```



В команде размещения битового поля FDEP неиспользуемые биты в регистре приемника НЕ ИЗМЕНЯЮТСЯ, если не указана опция SE или ZF. В этом примере биты 31-13 и 4-0 останутся неизменными.

Рисунок 89 – Пример команды FDEP

Следующая команда приведена в качестве примера операции маскирования битовых полей (MASK). Сдвиговое устройство извлекает биты из регистра XR4 в соответствии с маской в регистре XR3 и размещает их в регистре XR5. Биты XR5, не указанные в маске, остаются неизменными.

```
XR3 = 0x00007B00;; XR4 = 0x50325032;;
XR5= 0x85FFFFFF;; /* до команды mask */
XR5 += MASK R4 BY R3
/* После команды mask XR5 = 0x85FFD4FF */
```

6.1.1.5 Операции анализа битовых полей

Сдвиговое устройство поддерживает операции преобразования данных в форматах с фиксированной и плавающей точкой, включая:

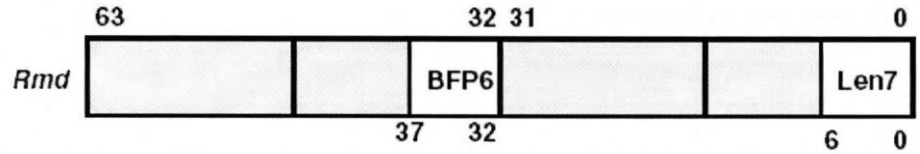
- BKFPT – определение масштабирующего множителя в 16-битной блочной плавающей арифметике;
- EXP – извлечение порядка;
- LDx – извлечение ведущих нулей (x=0) или единиц (x=1).

6.1.1.6 Операции с битовыми потоками

Операции с битовыми потоками в совокупности с командой BFOINC ALU позволяют реализовать однобитное FIFO, используемое для изменения битов в непрерывном битовом потоке. Сдвиговое устройство поддерживает операции с битовыми потоками, включая:

- GETBITS – извлечение битов из битового потока;
- PUTBITS – размещение битов в битовом потоке;
- BFOTMP – для временного сохранения или получения битов переполнения при командах GETBITS и PUTBITS.

В командах извлечения из битового потока (GETBITS) и размещения в битовом потоке (PUTBITS) операнд в *Rnd* содержит управляющую информацию в двух полях: BFP6 и Len7. Расположение этих полей в двойном регистре, *Rnd,* представлено на рисунке 90.



Операнд в двойном регистре определяет значения полей BFP6 и Len7 (биты 37-32 и 5-0). Обратите внимание, что BFP должен инкрементироваться с использованием команды ALU BFOINC.

Рисунок 90 – Поля BFP6 и Len7 в командах GETBITS и PUTBITS

Команда GETBITS извлекает биты, начиная с позиции, определённой полем BFP6, в количестве, определённом полем Len7, и помещает их, выровняв по правой границе, в выходной регистр. Неиспользуемые биты сбрасываются, если не указана опция (SE). Если опция SE указана, MSB извлекаемого поля распространяется на старшие неиспользуемые биты выходного регистра.

Следующая команда приведена в качестве примера операции извлечения из битового потока (GETBITS). Сдвиговое устройство извлекает биты из битового потока, расположенного в квадро-регистре XR3: 0, в соответствии со значениями поля позиции в однобитном FIFO (BFP6) и поля длины (Len7), указанными в двойном регистре XR7:6.

Примечание – используйте команду ALU BFOINC для продвижения указателя однобитного FIFO. После выполнения команды GETBITS обычно необходимо обновить указатель однобитного FIFO. Команда ALU BFOINC складывает значения полей BFP6 и Len7, делит полученную сумму на 64 и помещает остаток от деления (результат по модулю 64) в поле BFP6. Если, например, в BFP6 было 0x30, а в Len7 было 0x18, то новое значение в BFP6 будет 0x08, и флаг AN регистра XSTAT будет установлен. Этот флаг может быть использован, чтобы отследить ситуацию, когда сумма значений полей BFP6 и Len7 превысила 64, и действовать в соответствии с ней.

В данном примере результат операции, выровненный по правой границе, помещается в двойной регистр XR5:4. По умолчанию операция заполняет нулями неиспользуемые биты регистра приёмника (в примере – XR5:4). Если в команде GETBITS указана опция распространения знака (SE), то значение старшего бита (MSB) извлекаемого поля распространяется на все неиспользуемые биты регистра приёмника. На рисунке 91 показано, как изменяются биты в регистре XR5:4 в результате операции над полем с позицией BFP6=16 и длиной Len7=24.

```
XR0 = 0x76543210;;
XR1 = 0xFEDCBA98;;
XR2 = 0x82ABCDEF;;
XR3 = 0x01234567;;
XR6 = 0x00000018;;
XR7 = 0x00000010;;
XR5:4 = GETBITS R3:0 BY R7:6 ;;
/* После команды GETBITS XR5 = 0x0, XR4=0x987654 */
```

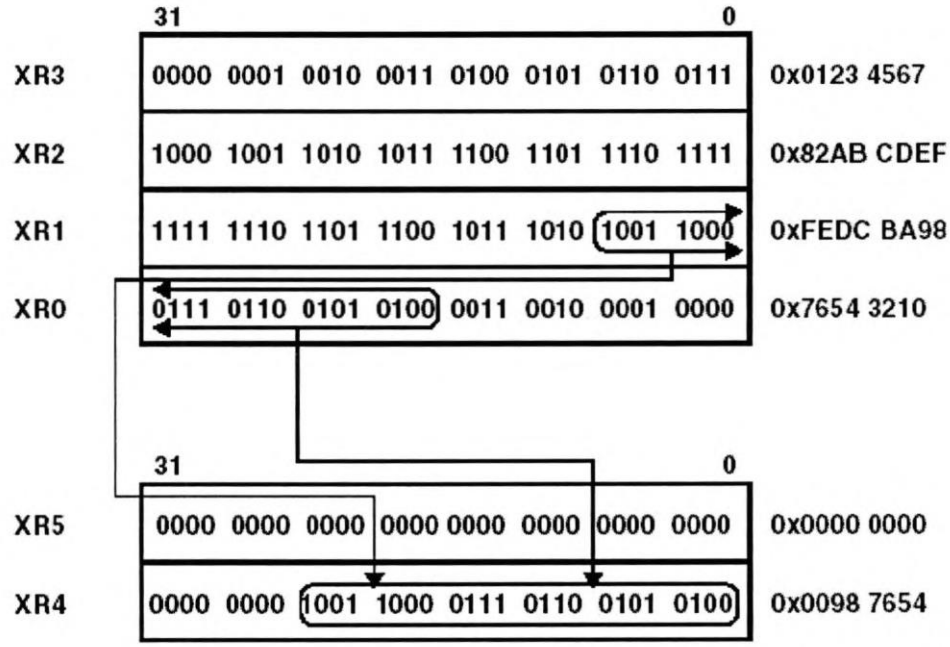
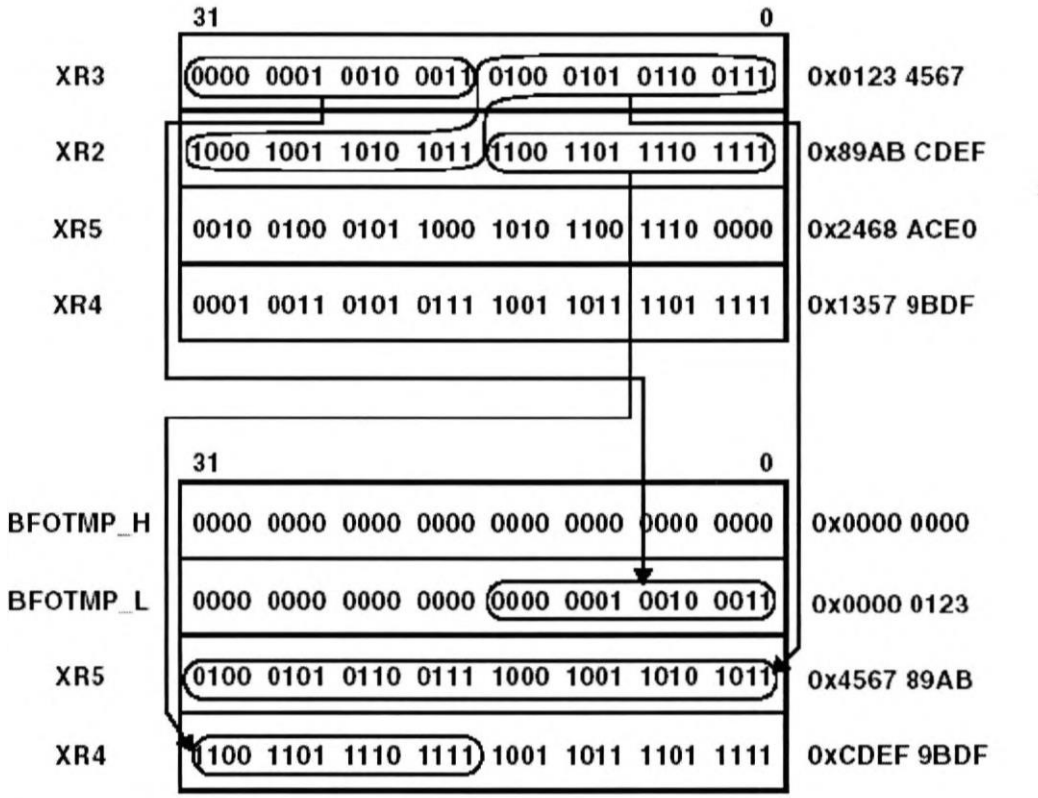


Рисунок 91 – Пример команды GETBITS

Команда PUTBITS помещает 64 бита из регистра Rmd в непрерывный битовый поток, расположенный в квадро-регистре, составленном из BFOTMP в старшей части и Rsd в младшей. В команде PUTBITS поле BFP6 определяет позицию, с которой начинается вставка в регистр Rsd, а поле Len7 не используется. Обновление значения в поле BFP6 должно быть произведено с помощью команды ALU BFOINC.

Следующая команда приведена в качестве примера операции размещения в битовом потоке (PUTBITS). Сдвиговое устройство помещает содержимое регистра XR3:2 в однобитное FIFO, составленное из содержимого регистров XR5: 4 и BFOTMP, начиная с 16-ого бита XR4 (в соответствии с указанным в поле BFP6 регистра XR7 значением). На рисунке 92 показано, как изменяются биты в регистре XR5:4 в результате операции над полем с позицией BFP6=16.

```
XR3 = 0x01234567;;
XR2 = 0x89abcdef;;
XR5 = 0x2468ace0;; /* до команды PUTBITS */
XR4 = 0x13579bdf;; /* до команды PUTBITS */
XR6 = 0x00000018;;
XR7 = 0x00000010;;
XR5:4 += PUTBITS R3:2 BY R7:6 ;;
/* После команды PUTBITS XR5 = 0x456789ab, XR4 = 0xcdef9bdf */
```



В команде размещения битового поля PUTBITS неиспользуемые биты в регистре приемника сохраняют свое предыдущее значение. В этом примере биты 15-0 регистра XR4 и биты 63-16 регистра BFOTMP сохранят свое предыдущее значение (предыдущее значение в BFOTMP предполагается равным 0x0000 0000).

Рисунок 92 – Пример команды PUTBITS

6.1.1.7 Опции команд сдвигового устройства

Некоторым командам сдвигового устройства сопоставлены опции, позволяющие гибко управлять режимом исполнения данной команды. Важно отметить, что эти опции влияют на то, как именно выполняются команды, а также тот факт, что существуют опции, применимые лишь с определёнными группами команд – не все опции используемы со всеми командами. Опции команд указываются в круглых скобках в конце поля команды.

Список всех опций, используемых с конкретными командами сдвигового устройства, можно найти в разделе «Сводка команд сдвигового устройства». Опции команд сдвигового устройства:

- ( ) заполнять нулями, выровнять по правой границе;
- (SE) распространять знак; применима к командам FEXT, FDEP и GETBITS;
- (ZF) заполнять нулями; применима к команде FDEP;
- (NF) не обновлять флаги.

Ниже представлены команды сдвигового устройства, демонстрирующие операции с битовыми полями при использовании опций.

```
XR5 = FEXT R4 By R3 (SE) ;; /* Опция SE в этой команде на рисунке 88 устанавливает биты 31-8 в 1. */
XR5 = FDEP R4 By R3 (ZF) ;; /* Опция ZF в этой команде на рисунке 89 сбрасывает биты 31-13 и 4-0. */
```

Опция Распространять знак

Опция распространения знака (SE) может быть использована в командах сдвигового устройства FEXT, FDEP и GETBITS. Указание этой опции предписывает команде распространить значение MSB используемого битового поля на все неиспользуемые биты выходного регистра.

Опция Заполнять нулями

Опция заполнения нулями (ZF) может быть использована только в команде сдвигового устройства FDEP. Указание этой опции предписывает команде сбросить все неиспользуемые биты, расположенные левее MSB используемого поля в выходном регистре.

Опция Не-обновлять-флаги

Почти все вычислительные операции влияют на значение регистра состояния вычислительного блока в соответствии со своим статусом завершения. Зачастую в некоторых приложениях оказывается полезным защитить значение регистра состояния от изменения в результате выполнения операции. Существует множество способов сохранения слова состояния (например, сохранение содержимого регистра состояния или загрузка значений флагов состояния в регистр статических флагов (SFREG)). Опция Не-обновлять-флаги (NF) предоставляет другой способ работы со словом состояния. Вместо того, чтобы сохранять статус завершения операции для дальнейшего использования, программа может запретить переписывать слово состояния, используя опцию (NF) со следующей командой для предотвращения генерирования той статуса завершения. Существуют некоторые ограничения на то, каким образом опция (NF) может быть использована в условных командах. Более подробно см. раздел «Отработка условных команд».

6.1.1.8 Статус выполнения сдвигового устройства

Операции сдвигового устройства изменяют флаги состояния арифметических регистров состояния вычислительных блоков (XSTAT и YSTAT). Программы могут использовать флаги состояния для управления исполнением условных команд и порождения программных исключений. Более подробно см. раздел «Предикаты сдвигового устройства».

В таблице 15 приведены флаги регистров XSTAT и YSTAT, отражающие статус завершения последней операции сдвигового устройства.

Таблица 15 – Флаги состояния сдвигового устройства

Флаг	Определение	Изменяется...
SZ	Признак равенства нулю в операции сдвигового устройства с фиксированной	Всеми операциями сдвигового устройства
SN	Признак отрицательного значения в операции сдвигового устройства	Всеми операциями сдвигового устройства

BF1-0	Признаки блочной плавающей арифметики	Только командой BKFPT

Обновление флагов происходит в конце выполнения каждой операции, делая флаги доступными уже в следующем поле команды. Программа не может записать значение в регистр XSTAT или YSTAT явно в том же командном такте, в котором происходит обновление слова состояния.

Команды с множественными операндами (например, BRs = ASHIFT Rm BY Rn) порождают несколько результатов. В этом случае процессор вычисляет флаг состояния как битовое ИЛИ соответствующих флагов для каждого результата.

Примечание – При использовании опции (NF) вычислительное устройство не обновляет значения флагов состояния в соответствии со статусом завершения операции. Разрешено использовать опцию (NF) со всеми командами сдвигового устройства, за исключением команд BITEXT, BKFPT и загрузкой в регистры X/YSTAT, TRx, THRx и CMCTL.

6.1.1.9 Предикаты сдвигового устройства

В условных командах сдвигового устройства исполнение всей командной строки зависит от значения предиката, указанного в начале командной этой строки. Условные команды сдвигового устройства имеют вид:

IF cond; DO, instr; DO, instr; DO, instruct. ;;

Такой синтаксис позволяет подчинить одному предикату до трёх условных команд. Пропуск DO перед командой делает её безусловной.

В таблице 16 приведён список предикатов сдвигового устройства. Более подробную информацию об условных командах см. в разделе «Отработка условных команд».

Таблица 16 – Предикаты сдвигового устройства

Предикат	Описание	Значения флагов
SEQ	Равно нулю в сдвиговом устройстве	SZ=1
SLT	Строго меньше нуля в сдвиговом устройстве	SN=1 и SZ=0
NSEQ	НЕ (Равно нулю в сдвиговом устройстве)	SZ=0
NSLT	НЕ (Строго меньше нуля в сдвиговом устройстве)	SN=0 или SZ=1

6.1.1.10 Статические флаги сдвигового устройства

В устройстве управления регистр статических флагов (SFREG) может хранить значения флагов состояния для их дальнейшего использования в условных командах. В регистре SFREG каждому вычислительному блоку выделено два бита флагов: X/YSCF0 (предикаты XSF0, YSF0, XYSF0 или SF0) и X/YSCF1 (предикаты XSF1, YSF1, XYSF1 или SF1).

Следующий пример демонстрирует загрузку значения предиката вычислительного блока в регистр статических флагов.

XSF0 = XSEQ;; /* Загрузить значение предиката SEQ вычислительного блока X в бит XSCF0 регистра статических флагов (SFREG) */

IF XSFO, DO XR5 = LSHIFT R4 BY R3 ;; /* предикат XSFO истинен, если установлен статический флаг XSCF0 */

Более подробно о статических флагах приведено в разделе «Отработка условных команд».

6.1.2 *Примеры использования сдвигового устройства*

В листинге 17 приведено несколько примеров команд сдвигового устройства. Ключевые особенности каждой команды, такие как формат операндов с фиксированной или плавающей точкой, размер операндов, а также использование регистров, перечислены в комментариях к команде.

Листинг 17 – Примеры команд сдвигового устройства

```
XR5 = LSHIFT R4 BY R3;;
/* Это логический сдвиг значения в регистре XR4 на количество бит, указанное в
регистре XR3. */
YR1 = ASHIFT R2 BY R0 ;;
/* Это арифметический сдвиг значения в регистре YR2 на количество бит, указанное
в регистре YR0. */
R1:0 = ROT R3:2 BY 8;;
/* Эта команда циклически сдвигает значение каждого слова в регистре R3:2 в
вычислительных устройствах X и Y на 8 бит и помещает результат в XR1:0 и YR1:0.*/
XBITEST R1:0 BY R7;;
/* Эта команда проверяет бит регистра XR1:0, указанный в регистре XR7, на
равенство 0 и устанавливает флаги XSZ и XSN в регистре XSTAT в соответствии с
результатом. */
R9:8 = BTGL R11:10 BY R13;;
/* Эта команда инвертирует бит регистра XR11:10 и YR11:10, указанный в регистрах
XR13 и YR13, и помещает результат в регистры XR9:8 и YR9:8.*/
XR15 = LD0 R17;;
/* Эта команда вычисляет количество ведущих нулей в регистре XR17 и помещает
результат в XR15. */
```

6.1.3 *Сводка команд сдвигового устройства*

В листинге 18 показан синтаксис команд сдвигового устройства. Соглашения по обозначениям имён регистров, необязательных элементов или вариантов, используемые в этих листингах, подробно описаны в разделе «Регистры регистровых файлов». Основные соглашения следующие:

- { } – необязательные элементы указаны в фигурных скобках; фигурные скобки не являются частью синтаксиса команды;
- | – вертикальная черта разделяет варианты значений элементов; вертикальные черты не являются частью синтаксиса команды;
- *Rmd* – имена регистров, выделенные курсивом, представляют собой подставляемые шаблоны для имён одиночных *(Rs, Rm* или *Rn),* двойных *(Rsd, Rmd* или *Rnd)* или квадро- *(Rsq, Rmq* или *Rnq)* регистров.

В командах сдвигового устройства имена выходных регистров связаны с размерами выходных операндов следующим образом:

- Префикс L перед именем выходного регистра указывает на выходной операнд в виде длинного слова (64 бита). Например, следующий синтаксис команды говорит об использовании одного выходного операнда в виде длинного слова:
  LRsd = ASHIFT Rmd BY Rnd
- Отсутствие префикса перед именем выходного регистра указывает на выходной операнд в виде обычного слова (32 бита). Например, следующий синтаксис команды говорит об использовании одного выходного операнда в виде обычного слова:

Rs = ASHIFT Rm BY Rn

- Двойной регистр, указанный в качестве выходного, говорит о двух выходных операндах в виде обычных слов (32 бита). Например, следующий синтаксис команды говорит об использовании двух выходных операндов в виде обычных слов:
  Rsd = ASHIFT Rmd BY Rnd
- Префикс S перед именем выходного регистра указывает на два или четыре выходных операнда в виде коротких слов (16 бит). Например, следующий синтаксис команды говорит об использовании двух или четырёх выходных операндов в виде коротких слов:
  S*Rs* = ASHIFT *Rm* BY *Rn* /* два выходных операнда */
  S*Rsd* = ASHIFT *Rmd* BY *Rnd* /* четыре выходных операнда */
- Префикс B перед именем выходного регистра указывает на четыре или восемь однобайтных выходных операнда (8 бит). Например, следующий синтаксис команды говорит об использовании четырёх или восьми однобайтных выходных операндов:
  B*Rs* = ASHIFT *Rm* BY *Rn* /* четыре выходных операнда */
  B*Rsd* = ASHIFT *Rmd* BY *Rnd* /* восемь выходных операндов */

Листинг 18 – Команды сдвигового устройства

```
{X|Y|XY}{B|S}Rs = LSHIFT|ASHIFT Rm BY Rn|<Imm> {(NF)} ;[1, 2]
{X|Y|XY}{B|S|L}Rsd = LSHIFT|ASHIFT Rmd BY Rn|<Imm> {(NF)} ; [1, 2]
{X|Y|XY}Rs = ROT Rm BY Rn|<Imm6> {(NF)}
{X|Y|XY} {L} Rsd = ROT Rmd BY Rn|<Imm> { (NF) } ; [1, 2]
{X|Y|XY}Rs = FEXT Rm BY Rn|Rnd {({SE}{ NF})} ; [3]
{X|Y|XY}LRsd = FEXT Rmd BY Rn|Rnd {({SE}{ NF})} ; [3]
{X|Y|XY}Rs += FDEP Rm BY Rn|Rnd {({SE|ZF}{NF})} ; [3]
{X|Y|XY}LRsd += FDEP Rmd BY Rn|Rnd {({SE|ZF}{NF})} ; [3]
{X|Y|XY}Rs += MASK Rm BY Rn {(NF)} ;
{X|Y|XY}LRsd += MASK Rmd BY Rnd {(NF)} ;
{X|Y|XY}Rsd = GETBITS Rmq BY Rnd {({SE}{NF})} ;
{X|Y|XY}Rsd += PUTBITS Rmd BY Rnd {(NF)} ;
{X|Y|XY}BITEST Rm BY Rn|<Imm5> ;
{X|Y|XY}BITEST Rmd BY Rn|<Imm6> ;
{X|Y|XY}Rs = BCLR|BSET|BTGL Rm BY Rn|<Imm5> { (NF)} ;
{X|Y|XY}Rsd = BCLR|BSET|BTGL Rmd BY Rn|<Imm6> {(NF)} ;
{X|Y|XY}Rs = LD0|LD1 Rm|Rmd {(NF)} ;
{X|Y|XY}Rs = EXP Rm|Rmd {(NF)} ;
{X|Y}STAT = Rm ;
{X|Y}STAT_L = Rm ;
{X|Y}Rs = {X|Y} STAT ;
{X|Y|XY}BKFPT Rmd, Rnd ;
{X|Y|XY}Rsd = BFOTMP {(NF)} ;
{X|Y|XY}BFOTMP = Rmd {(NF)} ;
{X|Y|XY}TRs = Rm ;
{X|Y|XY}TRsd = Rmd ;
{X|Y|XY}TRsq = Rmq ;
{X|Y|XY}THRs = Rm ;
```

[1] Размер данных в регистре *Rn,* определяющих величину сдвига, зависит от типа выходного операнда: для однобайтного - 5 бит, для короткого - 6, для обычного - 7, для длинного - 8.

[2] Размер данных *1mm,* определяющих величину сдвига, зависит от типа выходного операнда: для однобайтного - 4 бита, для короткого - 5, для обычного - 6, для длинного – 7.

[3] Расположение полей Pos8 и Len7 зависит от имени регистра - *Rn* или *Rnd.*

{X|Y|XY}THRsd = Rmd {(I)} ;
{X|Y|XY}THRsq = Rmq ;
{X|Y|XY}CMCTL = Rm ;
/ Синтаксис команд пересылки регистров TR, THR и CMCTL см. в разделе «Краткий справочник по CLU» */*

7 Целочисленное АЛУ (IALU)

Ядро процессора содержит два целочисленных арифметико-логических устройства, известных как IALU. Каждое IALU содержит регистровый файл и специальные регистры для адресации кольцевых буферов. Оба IALU могут управлять буферами выравнивания данных (DAB, Data Alignment Buffer), используемыми при операциях невыровненного доступа к памяти. На рисунке 93 блоки IALU и DAB отмечены выделением.

Два независимых целочисленных арифметико-логических устройства (IALU) процессора называются J-IALU и K-IALU. IALU может выполнять как обычные операции ALU, так и операции адресации данных. Целочисленными операциями ALU являются:

- сложение и вычитание с использованием или без использования бита переноса (займа);
- арифметический сдвиг вправо, логический сдвиг вправо и циклические сдвиги;
- логические операции: AND, AND NOT, NOT, OR и XOR;
- функции: абсолютное значение, минимум, максимум, сравнение.

IALU вычисляют адреса в памяти при пересылках данных между памятью и регистрами. Пара IALU позволяет одновременно вычислять адреса для операций доступа к памяти (чтение или запись). Операциями IALU типа *загрузка, сохранение и пересылка данных* являются:

- прямая и косвенная адресация памяти;
- адресация кольцевых буферов;
- бит-реверсивная адресация;
- пересылка и загрузка универсальных регистров *(Ureg);*
- генерация указателей на слова в памяти.

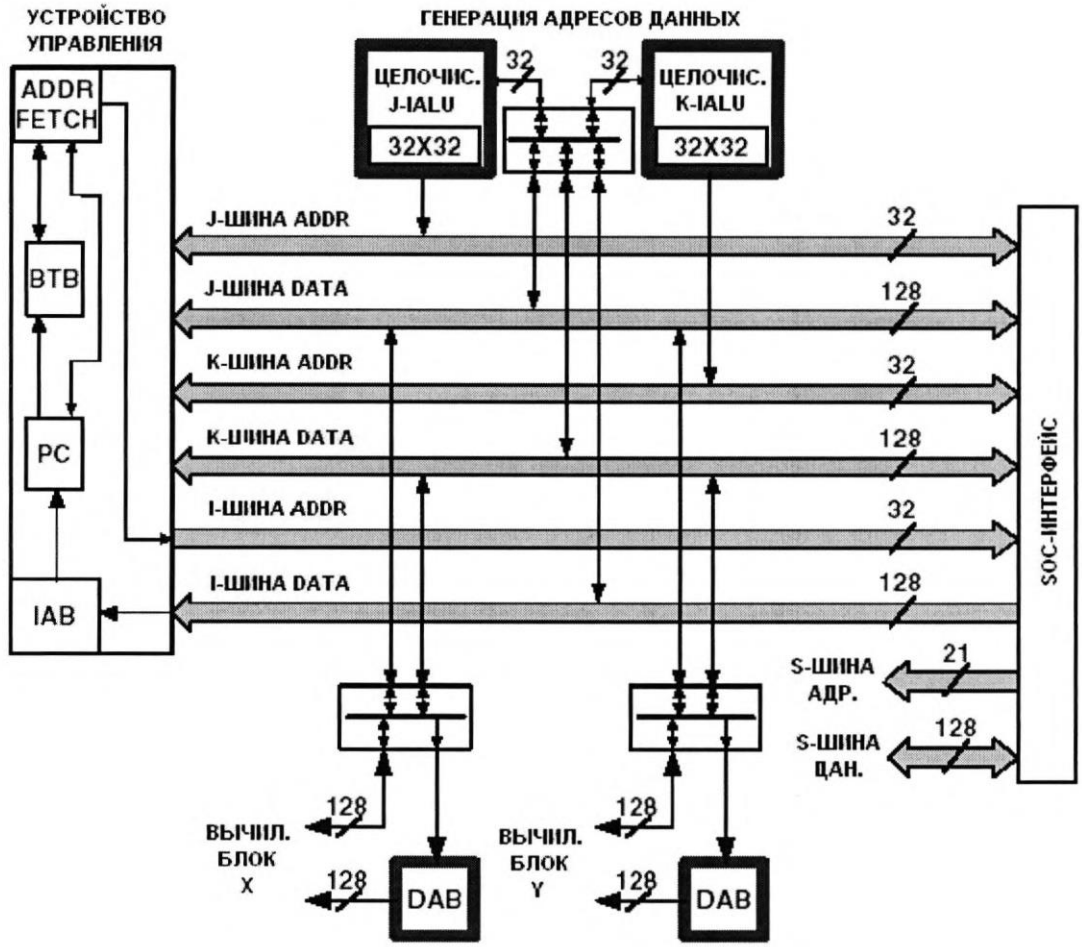


Рисунок 93 – IALU, DAB и шины данных

В каждой команде пересылки указывается, какой тип данных (обычное, длинное или квадрослово) используется при обращении к каждому блоку памяти. Так как в процессоре имеется два IALU, то на каждом такте может быть проведено два обращения к двум блокам памяти. Доступ к длинному слову может быть использован для получения двух выровненных обычных слов в одном вычислительном блоке, либо по одному выровненному обычному слову в каждом вычислительном блоке. Доступ к квадрослову может быть использован для получения четырёх выровненных обычных слов в одном вычислительном блоке, либо для получения двух пар обычных слов в обоих вычислительных блоках. Эта возможность оказывается полезной в приложениях, оперирующих действительными/мнимыми данными или параллельными множествами данных, выровненными в памяти, что нередко встречается в приложениях DSP. Эта возможность также используется для быстрого сохранения/восстановления контекста при вызовах функций и обработке прерываний.

IALU обеспечивает гибкость при пересылках данных, позволяя оперировать обычными, двойными или квадрословами. Любая такая команда может быть выполнена без задержек на каждом такте. Арифметические команды IALU выполняются с задержкой в один такт, в то время как арифметические команды вычислительных устройств с задержкой в два такта. Обычно задержки по зависимостям в командах IALU не имеют места, однако, если это не так, может произойти задержка на три или четыре такта. Более подробно см. подраздел «Роль зависимостей и ресурсов в работе конвейера».

Каждое IALU имеет регистровый файл данных из 32 регистров и восемь специальных регистров для адресации кольцевых буферов. Все регистры IALU являются 32-битными отображёнными в память универсальными регистрами. Перечислим несколько важных моментов, связанных с регистровыми файлами IALU:

- регистрами регистрового файла J-IALU являются регистры J30-J0, а регистрами регистрового файла K-IALU – регистры K30-K0. Все регистры, за исключением J31 и K31, являются регистрами общего назначения и могут содержать только целочисленные данные;
- для регистров J31 и K31, являющихся 32-битными статусными регистрами, используются также имена JSTAT и KSTAT.

Регистры JSTAT(J31) и KSTAT(K31) представлены на рисунке 94, для них определены специальные режимы работы:

- Если эти регистры используются в качестве входного операнда в арифметических, логических или функциональных командах IALU, то на них ссылаются как на J31 и K31, а их содержимое считается равным нулю. Если J31 [или K31] используется в качестве выходного операнда команды, то запись результата этой команды не производится, однако, все флаги устанавливаются адекватно этой операции.
- Если эти регистры используются в командах загрузки, сохранения или пересылки IALU (в качестве регистра-источника или приёмника, но не в качестве указателя), то на них ссылаются как на JSTAT и KSTAT, и их значение не очищается во время такой операции.

В операциях быстрого сохранения и восстановления команды загрузки и сохранения могут использовать J31: 28 в качестве квадро-регистра, сохраняя или восстанавливая тем самым J30: 28 и JSTAT одновременно.

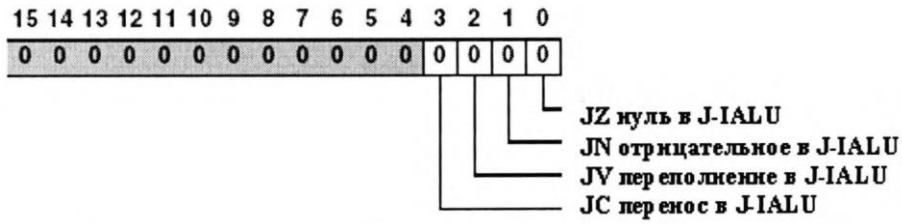


Рисунок 94 – Описание битов регистра JSTAT/KSTAT (младшая часть)

Специальные регистры для адресации кольцевых буферов в каждом IALU позволяют указать базовый адрес и длину кольцевого буфера. Эти регистры работают в связке с первыми четырьмя регистрами общего назначения в регистровом файле каждого из IALU, позволяя тем самым использовать до восьми кольцевых буферов. Перечислим несколько важных моментов, связанных со специальными регистрами IALU для адресации кольцевых буферов:

- Для хранения указателя кольцевого буфера (текущий адрес в нем) используется регистр общего назначения. Им может быть J3-J0 в J-IALU и K3-K0 в K-IALU.
- Для хранения базового адреса кольцевого буфера (его начальный адрес) используется специальный регистр. Им может быть JB3-JB0 в J-IALU и KB3-KB0 в K-IALU.
- Для хранения длины кольцевого буфера (количество ячеек памяти, занятых под буфер) используется специальный регистр. Им может быть JL3-JL0 в J-IALU и KL3-KL0 b K-IALU.
- *Модификатор* кольцевого буфера (приращение адреса в какой-либо

операции с ячейками памяти) может быть указан как регистр общего назначения IALU или как непосредственная константа. Значение модификатора не может превышать длину циклического буфера.

- Регистры указателя, базового адреса и длины при работе с кольцевыми буферами выступают как единое целое (J0 только с JB0 и JL0, J1 только с JB1 и JL1 и т.д.). В качестве модификатора может выступать любой регистр регистрового файла IALU, управляющего кольцевым буфером.

IALU могут задействовать команды сложения и вычитания при оперировании указателями в памяти как в режимах адресации кольцевых буферов или бит реверсивной адресации, так и в обычном режиме. Вычисленный адрес сохраняется в регистре данных IALU и при необходимости может быть записан в регистр вычисляемого перехода (CJMP) устройства управления.

7.1 Операции IALU

В следующих разделах описывается работа команд IALU всех типов. Все написанное применимо как к J-IALU, так и к K-IALU. Операциями IALU являются:

- целочисленные (арифметические и логические) операции IALU;
- операции IALU загрузки, сохранения и пересылки.

Во всех операциях IALU используются 32-битные целочисленные данные.

7.1.1 *Целочисленные (арифметические и логические) операции IALU*

IALU выполняет арифметические и логические операции над целочисленными данными в формате с фиксированной точкой. Процессор использует регистры регистрового файла IALU для получения входных операндов и записи результата операции IALU. Регистрами регистрового файла IALU являются J30-J0 и K30-K0. Каждое IALU имеет по одному регистру специального назначения, регистру JSTAT/KSTAT, для хранения слова состояния. Более подробную информацию о регистровых файлах и правилах именования регистров для указания типа и ширины данных см. в разделе «Регистры регистровых файлов». Ниже приведены примеры команд, демонстрирующих арифметические операции IALU.

```
J2 = J1 + J0 ;;
```

/* Это команда сложения двух 32-битных операндов в формате с фиксированной точкой в регистрах J1 и J0, результат помещается в регистр J2. */

```
K0 = ABS K2 ;; /* Процессор помещает в регистр K0 абсолютное значение 32-битного операнда в формате с фиксированной точкой из регистра K2. */
J2 = ( J1 + J0 ) / 2 ;;
```

/* Это команда сложения двух 32-битных операндов в формате с фиксированной точкой в регистрах J1 и J0 с последующим делением на 2, результат помещается в регистр J2. */

Все арифметические, логические и функциональные команды IALU генерируют значения флагов состояния по результатам операции. Например, если в предыдущей операции взятия полусуммы входные данные были -2 и 0 ((-2+0)/2), то будет установлен флаг JN (J- IALU, отрицательный результат в IALU), так как в результате операции получается отрицательное значение. Более подробно о статусном регистре IALU, см. раздел «Статус выполнения IALU».

7.1.1.1 Опции команд IALU

Большинству команд IALU сопоставлены опции, позволяющие гибко управлять режимом исполнения данной команды. Важно отметить, что эти опции влияют на то, как именно выполняются команды, а также тот факт, что существуют опции,

применимые лишь с определёнными группами команд - не все опции используемы со всеми командами. Опции команд указываются в круглых скобках в конце поля команды. Список всех опций, используемых с конкретными командами IALU, см. в разделе «Сводка команд IALU».

Опции команд ALU:

- ( ) операция со знаковыми операндами;
- (U) операция с беззнаковыми операндами;
- (CB) операция вычисления результата для адресации кольцевых буферов;
- (BR) операция вычисления результата для бит-реверсивной адресации;
- (CJMP) поместить результат в регистр-приёмник и регистр вычисляемого перехода (CJMP);
- (NF) не обновлять флаги.

Следующие примеры представляют собой команды IALU, демонстрирующие арифметические операции с применением опций.

```
J2 = J1 - J0 (CJMP) ;;
/* Это операция вычисления разности 32-битных операндов в формате с
фиксированной точкой, результат помещается в регистры J2 и CJMP. */
K1 = K3 + K4 (BR) ;;
/* Это операция сложения с обратным переносом 32-битных операндов в
формате с фиксированной точкой.*/
COMP(J1, J0) (U) ;;
/* Это операция сравнения беззнаковых 32-битных операндов. */
```

Опция Знаковое/Беззнаковое

В IALU для 32-битных данных в формате с фиксированной точкой используется дополнительное представление. Подробнее о поддерживаемых числовых форматах данных см. раздел «Числовые форматы».

Опция Кольцевой буфер

Команды сложения и вычитания IALU имеют опцию (CB) для адресации кольцевых буферов. Эти команды имеют вид:

```
Js = Jm +|- Jn|<Imm8>|<Imm32> {({CJMP|CB|BR})} ;
Ks= Km +|- Kn| <Imm8>|<Imm32> {({CJMP|CB|BR})} ;
```

Перед использованием опции CB в командах сложения и вычитания в IALU необходимо предварительно проинициализировать соответствующие регистры базового адреса и длины кольцевого буфера J-IALU или K-IALU. Команды сложения и вычитания в IALU с опцией CB вычисляют новый адрес, складывая значение указателя буфера и его модификатора или вычитая из значения указателя модификатор, а также, при необходимости, выполняя операцию перехода через границу кольцевого буфера. Результат операции IALU помещает в регистр Js или Ks (Jm или Km не изменяются).

Опция Обратный перенос

Команды сложения и вычитания IALU имеют опцию (BR) для бит-реверсивной адресации. Эти команды имеют вид:

```
Js = Jm +|- Jn|<Imm8>|<Imm32> {({CJMP|CB|BR})} ;
Ks= Km +|- Kn| <Imm8>| <Imm32> {({CJMP|CB|BR})} ;
```

Команды сложения и вычитания в IALU с опцией `BR` используют обратный перенос для вычисления результата сложения указателя буфера с его модификатором (в случае вычитания бит переноса не используется, и поэтому эта опция не оказывает

никакого влияния), а затем помещают полученное значение в регистр *Js* или *Ks (Jm* или *Km* не изменяются).

Опция Вычисляемый переход

Команды сложения и вычитания IALU имеют опцию (CJMP) для поддержки вычисляемых переходов. Эти команды имеют вид:

Js = Jm +|- Jn|<Imm8>|<Imm32> {({CJMP|CB|BR})} ;

Ks= Km +|- Kn|<Imm8>|<Imm32> {({CJMP|CB|BR})} ;

Опция вычисляемого перехода (CJMP) указывает IALU, что результат операции необходимо поместить как в регистр приемника *(Js* или *Ks),* так и в регистр CJMP устройства управления.

Опция Не-обновлять-флаги

Почти все вычислительные операции IALU влияют на значение регистра состояния IALU в соответствии со своим статусом завершения. Зачастую в некоторых приложениях оказывается полезным защитить значение регистра состояния от изменения в результате выполнения операции. Существует множество способов сохранения слова состояния (например, сохранение содержимого регистра состояния или загрузка значений флагов состояния в регистр статических флагов (SFREG)). Опция Не-обновлять-флаги (NF) предоставляет другой способ работы со словом состояния. Вместо того, чтобы сохранять статус завершения операции для дальнейшего использования, программа может запретить переписывать слово состояния, используя опцию (NF) в следующей команде, предотвращая генерирование той статуса завершения. Существуют некоторые ограничения на то, каким образом опция (NF) может быть использована в условных командах. Более подробно см. раздел «Отработка условных команд».

7.1.2 *Статус выполнения IALU*

Операции IALU изменяют значения флагов состояния в статусных регистрах IALU (регистры JSTAT и KSTAT). Программы могут использовать флаги состояния для управления исполнением условных команд и порождения программных исключений. Более подробно см. раздел «IALU-предикаты».

В таблице 17 приведены флаги регистров JSTAT и KSTAT, отражающие статус завершения последней операции IALU.

Таблица 17 – Флаги состояния IALU

Флаг	Определение	Изменяется...
JZ	Признак равенства нулю в операции J-IALU	Арифметическими, логическими и функциональными операциями J-IALU
JN	Признак отрицательного значения в операции J-ALU	Арифметическими, логическими и функциональными операциями J-IALU
JV	Признак переполнения в операции J-ALU	Всеми арифметическими операциями, сбрасывается логическими операциями
JC	Признак переноса в операции J-ALU	Устанавливается операциями сложения/вычитания, сбрасывается всеми остальными операциями
KZ	Признак равенства нулю в операции K-IALU	Арифметическими, логическими и функциональными операциями K-IALU
KN	Признак отрицательного значения в операции K-ALU	Арифметическими, логическими и функциональными операциями K-IALU

Флаг	Определение	Изменяется...
KV	Признак переполнения в операции K-ALU	Всеми арифметическими операциями, сбрасывается логическими операциями
КС	Признак переноса в операции K-ALU	Устанавливается операциями сложения /вычитания, сбрасывается всеми остальными операциями

Обновление флагов происходит в конце выполнения каждой операции, делая флаги доступными уже в следующей строке команд. Программа не может записать значение в статусный регистр IALU явно в том же командном такте, в котором происходит операция IALU.

Примечание – при использовании опции (NF) IALU не обновляет значения флагов состояния в соответствии со статусом завершения операции. Разрешено использовать опцию (NF) со всеми арифметическими, логическими и функциональными командами IALU, за исключением команды COMP.

7.1.2.1 JZ/KZ - Признак равенства нулю в IALU

Флаг JZ или KZ устанавливается, когда в результате операции J-IALU или K-IALU получился ноль. Флаг сбрасывается, если в результате получился не ноль.

7.1.2.2 JN/KN - Признак отрицательного в IALU

Флаг JN или KN устанавливается, если результат операции IALU отрицателен. Обычно флаг JN или KN устанавливается равным MSB результата. По иным правилам флаг JN или KN устанавливается только в случае команд, приведённых ниже:

- Js = ABS Jm ; здесь JN равен знаку Jm (входных данных)
- Ks = ABS Km ; здесь KN равен знаку Km (входных данных)

Знак результата команд, приведённых выше, не указан, поскольку их результат всегда положителен.

7.1.2.3 JV/KV - Признак переполнения в IALU

Флаг JV или KV отражает состояние переполнения. Во всех командах J-IALU или K-IALU этот бит устанавливается, когда результат операции настолько велик, что не может быть представлен в требуемом формате. Проверка переполнения производится всегда в предположении, что результат знаковый.

В следующем примере значения в регистрах J5 и J6 равны 0x70...0 (большие положительные числа), результат команды сложения больше, чем позволяет формат данных для знаковых операций.

J10 = J5 + J6 ;;

7.1.2.4 JC/KC - Признак переноса в IALU

Флаг JC или КС используется как бит переноса в командах сложения или вычитания, которые могут быть сцеплены. Он также может быть использован как признак беззнакового переполнения в этих операциях (в случае знакового переполнения устанавливается флаг JV или KV). Операции с обратным переносом не могут вызвать переполнения и не устанавливают флаги JC и КС.

7.1.3 *IALU-предикаты*

В условных командах IALU исполнение всей командной строки зависит от выполнения предиката, указанного в начале этой командной строки. Условные команды IALU имеют вид:

IF cond; DO, instr; DO, instr; DO, instr ;;

Такой синтаксис позволяет подчинить одному предикату до трёх условных команд. Пропуск DO перед командой делает её безусловной.

В таблице 18 приведён список IALU-предикатов. Более подробную информацию об условных командах можно найти в разделе «Отработка условных команд».

Таблица 18 – IALU-предикаты

Предикат	Описание	Значения флагов
JEQ	Равно нулю в J-IALU	JZ=1
JLT	Строго меньше нуля в J-IALU	JN=1 и JZ=0
JLE	Меньше или равно нулю в J-IALU	JN=1 или JZ=1
NJEQ	НЕ (Равно нулю в J-IALU)	JZ=0
NJLT	НЕ (Строго меньше нуля в J-IALU)	JN=0 или JZ=1
NJLE	НЕ (Меньше или равно нулю в J-IALU)	JN=0 и JZ=0
KEQ	Равно нулю в K-IALU	KZ=1
KLT	Строго меньше нуля в K-IALU	KN=1 и KZ=0
KLE	Меньше или равно нулю в K-IALU	KN=1 или KZ=1
NKEQ	НЕ (Равно нулю в K-IALU)	KZ=0
NKLT	НЕ (Строго меньше нуля в K-IALU)	KN=0 или KZ=1
NKLE	НЕ (Меньше или равно нулю в K-IALU)	KN=0 и KZ=0

Статические флаги IALU

В устройстве управления регистр статических флагов (SFREG) может хранить значения флагов состояния для их дальнейшего использования в условных командах. В регистре SFREG на все IALU выделено два бита флагов: ISF0 (предикат ISF0) и ISF1 (предикат ISF1). Следующий пример демонстрирует загрузку значения предиката IALU в регистр статических флагов.

ISF0 = JEQ ;; /* Загрузить значение предиката J-IALU JEQ в бит ISF0 регистра статических флагов (SFREG)*/

IF ISF0, DO J5 = J4 + J3 ;; /* предикат ISF0 истинен, если установлен статический флаг ISF0 */

Помимо предикатов IALU (JEQ, JLT, JLE, KEQ, KLT, KLE или их отрицаний) глобальные статические флаги IALU могут хранить также значения предикатов вычислительных устройств (предикатов ALU, умножителя или сдвигового устройства) и предикаты устройства управления (LCxE, ISFx, TRUE, BM и FLAGx_IN). Более подробно о статических флагах приведено в разделе «Отработка условных команд».

Операции IALU загрузки, сохранения и пересылки

Команды IALU адресации данных обеспечивают поддержку чтения и записи памяти для загрузки и сохранения регистров. Для чтения и записи памяти IALU поддерживает два типа адресации: прямая и косвенная адресации. При косвенной адресации используются указатель и модификатор адреса.

Указатель является адресом в памяти, а модификатор – значением, прибавляемым к нему до (предмодификация) или после (постмодификация) выполнения операции обращения по адресу в памяти. Синтаксис команды адресации IALU использует квадратные скобки ([ ]) для визуального отделения части команды, вычисляющей адрес, от других частей команды.

7.1.3.1 Прямая и косвенная адресация

Прямая адресация использует нулевой указатель адреса (регистр J31) и непосредственное значение в качестве модификатора. При этом указатель адреса пре- модифицируется, и полученное значение используется в качестве адреса при доступе в память. Следующая команда проводит загрузку регистра (чтение памяти) в режиме прямой адресации:

YR1 = [J31 + 0X00015F00] ;;

/* Эта команда считывает 32-битное слово, расположенное в памяти по адресу 0x00015f00, и загружает его в регистр YR1. Обратите внимание на то, что значение регистра J31 всегда 0 при использовании его в качестве входного операнда. */

Косвенная адресация использует любой регистр регистрового файла IALU в качестве указателя, а в качестве модификатора адреса – регистр или непосредственное значение. Как видно из рисунка 94, существует два вида косвенной адресации.

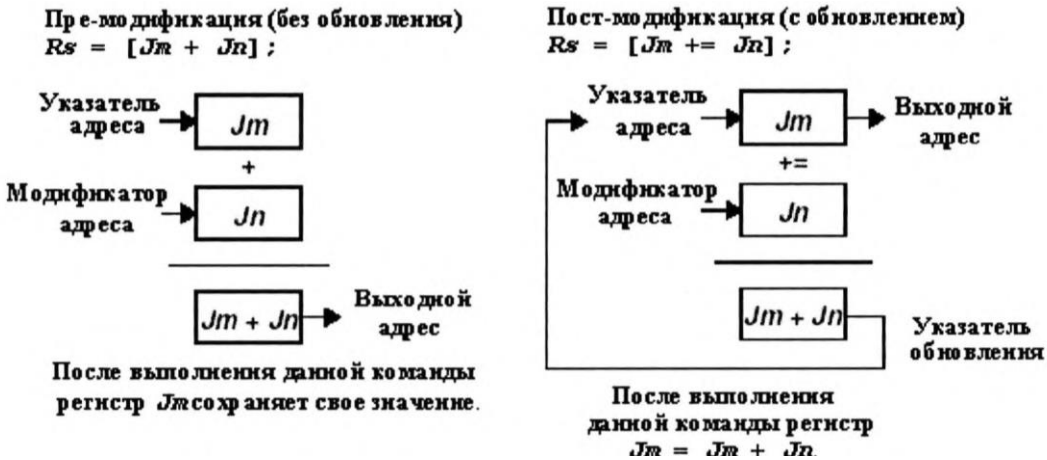


Рисунок 95 – Пре- и постмодификация в режиме косвенной адресации

Первый вид косвенной адресации – это предмодификация без обновления (используется оператор [+]). Команды этого вида обеспечивают доступ к памяти по адресу указатель + модификатор без изменения содержимого регистра указателя. Такие команды могут загружать значение в регистр-приемник или сохранять значение регистра-источника. Например:

XR0 = [J0+ J1] ;;

/* Эта команда считывает 32-битное слово, расположенное в памяти по адресу J0+J1 (указатель + модификатор), и загружает его в регистр XR0. При этом вычисленное значение адреса не заносится в регистр J0 (указатель). */

Второй вид косвенной адресации - это *постмодификация с обновлением* (используется оператор [+=]). Команды этого вида обеспечивают доступ к памяти по адресу в указателе. Такие команды могут загружать значение в регистр-приемник или сохранять значение регистра-источника. После доступа к памяти регистр указателя изменяется на значение модификатора. Например:

XR0 = [J0 += J1] ;;

/* Эта команда считывает 32-битное слово, расположенное в памяти по адресу J0 (указатель), и загружает его в регистр XR0. После этого в регистр J0 заносится новое значение J0+J1 (указатель + модификатор). */

Косвенная адресация с постмодификацией обязательна при использовании режимов адресации кольцевых буферов, DAB и бит-реверсивной адресации. Более подробно см. разделы «Адресация кольцевых буферов», «Доступ к памяти с

использованием буфера выравнивания данных (DAB)» и «Бит-реверсивная адресация».

Обычный, совмещённый и широковещательный доступ к памяти

IALU использует прямую или косвенную адресации для выполнения чтения или записи памяти при загрузке или сохранении данных регистров. Имеется три класса доступов к памяти: обычный доступ для чтения/записи, совмещённый доступ для чтения/записи и широковещательный доступ для загрузки. Эти классы различаются следующим:

Обычный доступ для чтения/записи - используется для загрузки или сохранения данных универсальных регистров. При обычном доступе для чтения или записи используется одно, два или четыре 32- битных слова в памяти, необходимые для загрузки регистра приёмника или сохранения регистра источника, в зависимости от конкретной команды. Обычный доступ к памяти имеет место, если размер регистра источника или приёмника удовлетворяет префиксу оператора IALU доступа к памяти. Примеры обычного доступа к памяти (приёмник <=> источник):

Одиночный регистр (Rs) <=> оператор не указан
Двойной регистр (Rsd) <=> оператор L (длинное)
Квадро-регистр (Rsq) <=> оператор Q (квадро-).

Широковещательный доступ для загрузки - используется для чтения данных из памяти и загрузки их в регистры данных вычислительных блоков. При широковещательном доступе используется одно, два или четыре 32 битных слова в памяти, которые загружаются в регистры приёмника в обоих вычислительных блоках. Широковещательный доступ к памяти имеет место, если размер регистра приёмника соответствует оператору IALU, а в имени регистра используется (или отсутствует) префикс XY, обозначающий оба вычислительных блока сразу. Примеры широковещательного доступа к памяти (приёмник <= источник):

Одиночный регистр *(Rs)* <= оператор не указан
Двойной регистр *(Rsd)* <= оператор L (длинное)
Квадро-регистр *(Rsq)* <= оператор Q (квадро-)

Совмещённый доступ для чтения/записи используется для загрузки или сохранения данных регистров вычислительных блоков. Пересылки совмещённых данных могут выполняться только длинными или квадрословами.

При этом пересылка осуществляется между памятью и регистровыми файлами обоих вычислительных блоков, а данные разделяются между вычислительными блоками X и Y. Так при чтении длинного слова из памяти одна его часть (одно слово) загружается в X, а вторая (одно слово) – в Y. При чтении квадрослова из памяти одна его часть (два слова) загружается в X, а вторая (два слова) – в Y. Совмещённый доступ к памяти может быть осуществлен в обоих направлениях (загрузка и сохранение), а порядок доступа может быть определён при помощи префиксов XY и YX. Если указан префикс XY, то старшая часть пересылаемых данных (старшее слово при пересылке длинного слова или два старших слова при пересылке квадрословами) будет загружена в регистры вычислительного блока X, а младшая часть – в регистры Y. Если указан префикс YX, то старшая часть пересылаемых данных будет загружена в регистры вычислительного блока Y, а младшая часть – в регистры X. Примеры совмещённого доступа к памяти (приёмник <=> источник):

Одиночный регистр (Rs) <=> оператор L (длинное)

Двойной регистр (Rsd) <=> оператор Q (квадро-)

На рисунках 96 – 109 представлены лишь существенные примеры, демонстрирующие классы доступов к памяти. На этих рисунках показан доступ к памяти только для случая регистров данных, а не всех универсальных регистров.

Также на них показан доступ к памяти только при косвенной адресации с постмодификацией. Полный синтаксис всех команд IALU доступа к памяти можно найти в разделе «Сводка команд IALU».

Глядя на рисунок 96 (содержимое памяти) и другие рисунки (примеры доступа к памяти), важно обратить внимание на взаимосвязь размера данных и их выравнивания. При доступе к памяти для загрузки или сохранения одиночного регистра выравнивание данных в памяти не является проблемой, в этом случае значение указателя адреса может быть любым.

При доступе к памяти для загрузки или сохранения двойных или квадро-регистров выравнивание данных в памяти очень важно. При загрузке или сохранении двойных регистров необходимо, чтобы значение указателя адреса делилось на два (выравнивание на длинное слово). При загрузке или сохранении квадро-регистров необходимо, чтобы значение указателя адреса делилось на четыре (выравнивание на квадрослово). Выровнять данные в памяти можно с помощью директив ассемблера или используя синтаксис файла описания компоновки (. LDF).

Доступ к памяти с использованием DAB и SDAB не имеет ограничений на выровненность данных. Более подробно о доступе к памяти с использованием DAB и SDAB см. раздел «Доступ к памяти с использованием буфера выравнивания данных (DAB)».

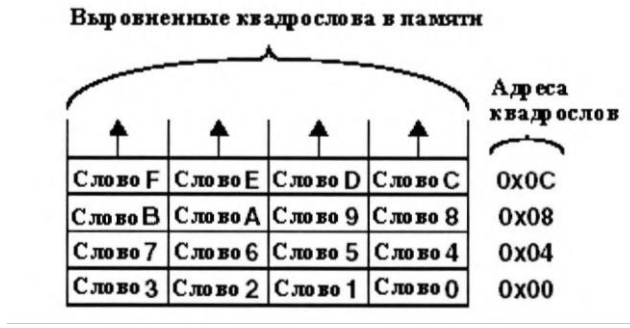
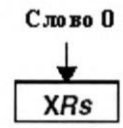


Рисунок 96 – Содержимое памяти для примеров обычного, широковещательного и совмещённого доступов к памяти

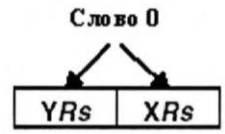
```
XRs = [ Jm += Jn] ; ; /* Jm = адресу слова 0. */
```



Содержимое памяти представлено на рисунке 96

Рисунок 97 – Обычный доступ на чтение для случая одиночного регистра

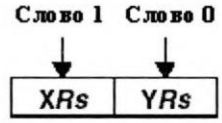
```
XYRs = [Jm += Jn] ; ; /* Jm = адресу слова 0. */
```



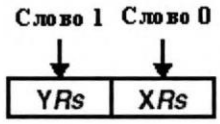
Содержимое памяти представлено на рисунке 96

Рисунок 98 – Широковещательный доступ для загрузки одиночного регистра

XY*Rs* = L [*Jm* += *Jn*] ;; /* *Jm* = адресу слова 0. */



YX*Rs* = L [*Jm* += *Jn*] ;; /* *Jm* = адресу слова 0. */

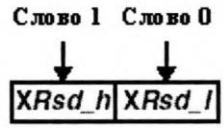


Содержимое памяти представлено на рисунке 96

Отметьте, что использование синтаксиса с префиксами XY и YX приводит к *разным* результатам.

Рисунок 99 – Совмещённый доступ на чтение для случая одиночного регистра

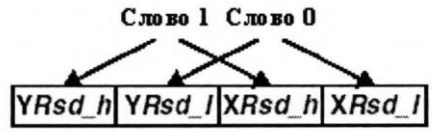
XR*sd* = L [*Jm* += *Jn*] ;; /* *Jm* = адресу слова 0. */



Содержимое памяти представлено на рисунке 96

Рисунок 100 – Обычный доступ на чтение для случая двойного регистра

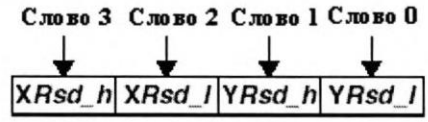
XY*Rsd* = L [*Jm* += *Jn*] ;; /* *Jm* = адресу слова 0. */



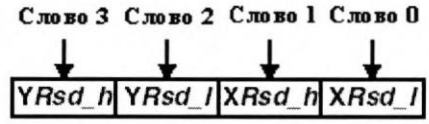
Содержимое памяти представлено на рисунке 96

Рисунок 101 – Широковещательный доступ для загрузки двойного регистра

```
XYRsd = Q [Jm += Jn] ;; /* Jm = адресу слова 0. */
```




```
YXRsd = Q [Jm += Jn] ;; /* Jm = адресу слова 0. */
```

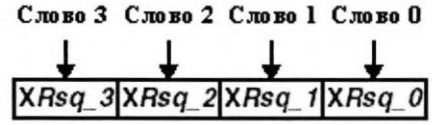


Содержимое памяти представлено на рисунке 96

Отметьте, что использование синтаксиса с префиксами XY и YX приводит к *разным* результатам.

Рисунок 102 – Совмещённый доступ на чтение для случая двойного регистра

```
XRsq = Q [Jm += Jn] ;; /* Jm = адресу слова 0. */
```



Содержимое памяти представлено на рисунке 96

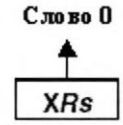
Рисунок 103 – Обычный доступ на чтение для случая квадро-регистра

```
XYRsq = Q [Jm += Jn] ;; /* Jm = адресу слова 0. */
```

Содержимое памяти представлено на рисунке 96

Рисунок 104 – Широковещательный доступ для загрузки квадро-регистра

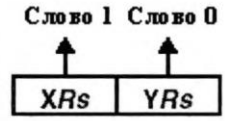
```
[Jm += Jn] = XRs ;; /* Jm = адресу слова 0. */
```



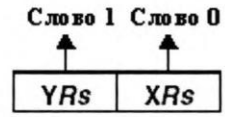
Содержимое памяти представлено на рисунке 96

Рисунок 105 – Обычный доступ на запись для случая одиночного регистра

```
L  [Jm += Jn]  = XYRs;;  /* Jm = адресу слова 0. */
```




```
L  [Jm += Jn]  = YXRs  ;;  /* Jm = адресу слова 0. */
```

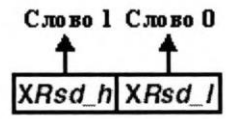


Содержимое памяти представлено на рисунке 96

Отметьте, что использование синтаксиса с префиксами XY и YX приводит к *разным* результатам.

Рисунок 106 – Совмещённый доступ на запись для случая одиночного регистра

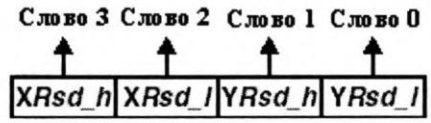
```
L  [Jm += Jn]  = XRsd  ;;  /* Jm = адресу слова 0. */
```



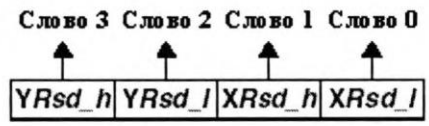
Содержимое памяти представлено на рисунке 96

Рисунок 107 – Обычный доступ на запись для случая двойного регистра

```
Q  [Jm += Jn]  = XYRsd;;  /* Jm = адресу слова 0. */
```




```
Q  [Jm += Jn]  = YXRsd;;  /* Jm = адресу слова 0. */
```

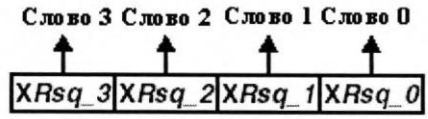


Содержимое памяти представлено на рисунке 96

Отметьте, что использование синтаксиса с префиксами XY и YX приводит к *разным* результатам.

Рисунок 108 – Совмещённый доступ на запись для случая двойного регистра

```
Q [Jm += Jn] = XRsq; ; /* Jm = адресу слова 0. */
```



Содержимое памяти представлено на рисунке 96

Рисунок 109 – Обычный доступ на запись для случая квадро-регистра

7.1.3.2 Доступ к памяти с использованием буфера выравнивания данных (DAB)

Каждый вычислительный блок имеет собственный буфер выравнивания данных (X-DAB и Y-DAB) для доступа к невыровненным данным в памяти. Пользуясь DAB, программа может прочитать из памяти невыровненное квадрослово данных и загрузить его в квадрорегистр данных (*Rsq*).

При загрузке двойных и квадро-регистров без использования оператора DAB или SDAB данные в памяти должны быть выровнены. Более подробно о выравнивании данных см. в разделе «Обычный, совмещённый и широковещательный доступ к памяти».

DAB представляет собой 128-разрядный буферный регистр. На вход DAB поступают выровненные квадрослова из памяти, а на выходе формируются квадрослова, которые, возможно, располагаются в памяти как невыровненные. Это достигается за счет того, что в буфере DAB хранится выровненное квадрослово, а требуемое для загрузки регистров невыровненное квадрослово будет формироваться частично из слов, хранящихся в DAB, а частично из того (выровненного) квадрослова, которое получено при текущем считывании.

Для понимания работы DAB можно рассмотреть выровненный и невыровненный доступы к памяти и сравнить, какие операции выполняет DAB при каждом из них. На рисунке 110 показано, как работает DAB, когда команда IALU читает выровненные данные из памяти. Сравните с доступом к невыровненным данным с использованием DAB, показанном на рисунке 111.

Рисунок 110 демонстрирует несколько важных особенностей доступа к памяти с использованием DAB. Доступ к памяти с использованием DAB предназначен для использования в случае серии обращений к памяти, используя режим адресации кольцевых буферов или линейный режим адресации. Перед тем как DAB будет готов к использованию в серии обращений к памяти, необходимо провести первое чтение для инициализации DAB - очистки предыдущих данных и загрузки первых данных серии. При этом DAB автоматически определяет границу квадрослова, ближайшую к указателю адреса, и считывает из памяти корректное квадрослово для загрузки его в DAB.

Так как при доступе к памяти с использованием DAB автоматически выполняются вычисления адреса в режиме адресации кольцевых буферов, то регистры режима адресации кольцевых буферов – указателя, базового адреса, длины и модификатора – должны быть установлены в соответствующие значения до того, как будет использован доступ к памяти с использованием DAB. По этой же причине команды доступа с использованием DAB должны использовать только регистры IALU, предназначенные для режима адресации кольцевых буферов (J3-J0, K3-K0). Более подробно о режиме адресации кольцевых буферов см. раздел «Адресация кольцевых буферов». Для корректной работы DAB в режиме адресации кольцевых буферов значение регистра модификатора адреса (Jn или Kn) должно быть равно четырём.

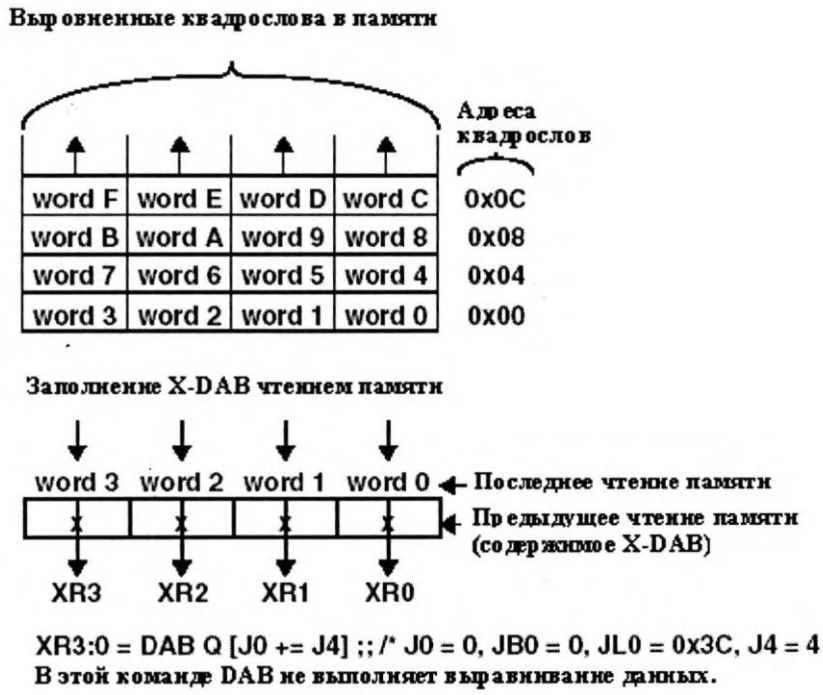


Рисунок 110 – Работа DAB при доступе к выровненным данным

Если при доступе к памяти с использованием DAB необходимо использовать линейный режим адресации, установите регистр длины кольцевого буфера (соответствующий регистр JL/KL) в ноль.

Рисунок 111 – Работа DAB при доступе к невыровненным данным

Как показано на рисунке 112, механизм DAB также поддерживает доступ к невыровненным короткословным данным в памяти. При доступе к памяти с использованием буфера выравнивания коротких данных (SDAB) все сказанное о настройке и работе DAB, за исключением двух моментов, также имеет место. Но в этом случае, во-первых, для корректной работы в режиме адресации кольцевых буферов значение регистра модификатора адреса *(Jn* или *Kn)* должно быть равно восьми. Во-вторых, значение указателя адреса в командах SDAB либо четно (в случае коротких слов в памяти, выровненных на границу обычного слова), либо нечетно (в случае коротких слов в памяти, не выровненных на границу обычного слова).

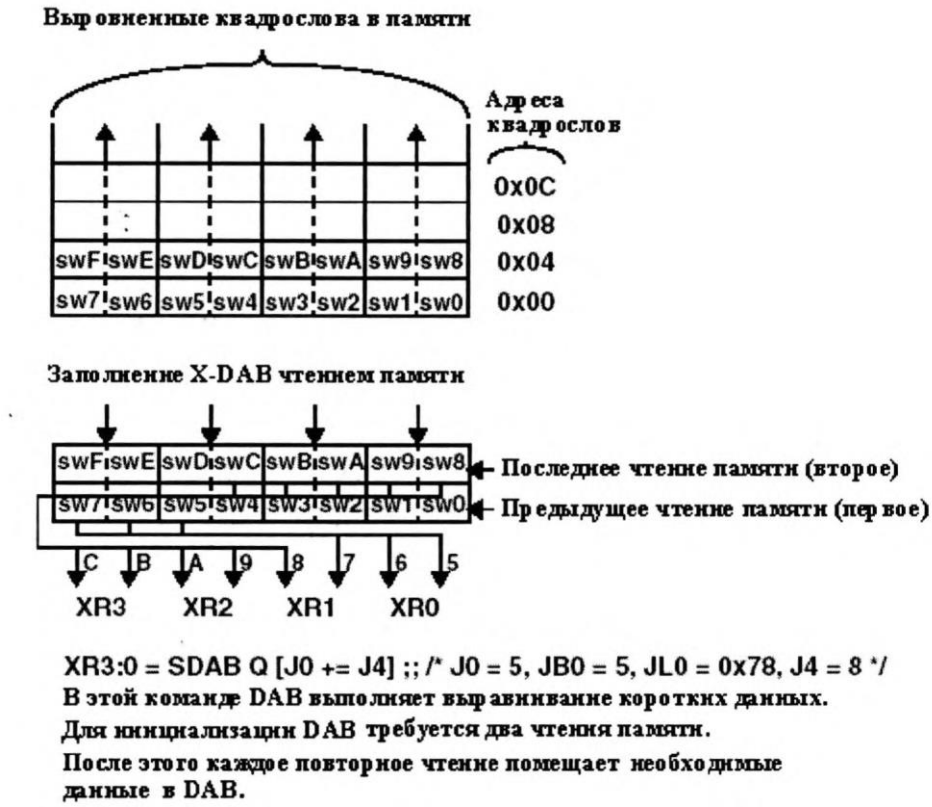


Рисунок 112 – Работа SDAB при доступе к невыровненным данным

Сравнение случаев с такими значениями указателей адреса в командах DAB и SDAB представлено на рисунке 113.

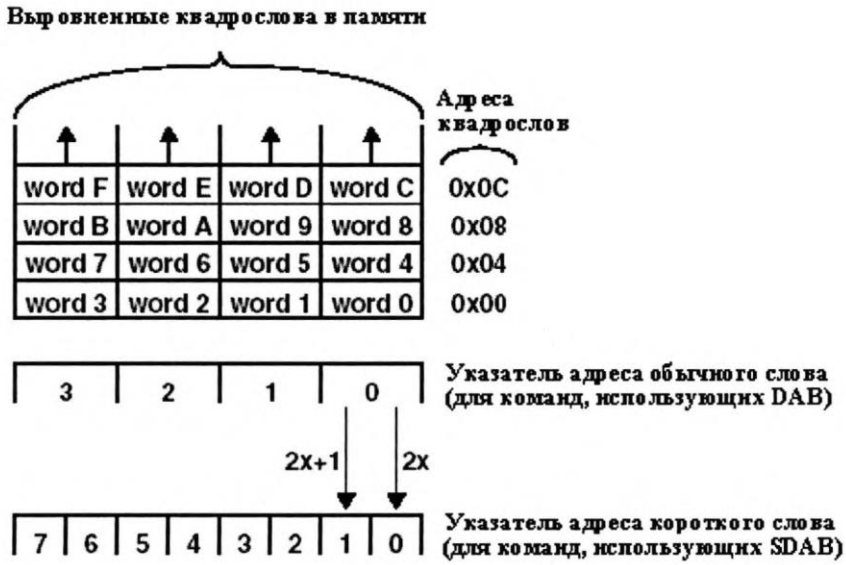


Рисунок 113 – Значения указателей адреса при использовании DAB и SDAB

Доступ к памяти с использованием DAB со смещением

Загрузка регистров вычислительных блоков может быть выполнена с использованием DAB со специальным смещением. В отличие от простого режима работы с DAB, помимо обычного сдвига DAB, в этой операции выполняется фиксированный сдвиг на 1 для одного вычислительного блока и на 0 для другого. Сдвиг может быть выполнен на одно короткое (16-битное) слово или одно обычное (32-битное) слово. Синтаксис таких команд показан в листинге 19. При помощи такого специального смещения можно использовать перемежающиеся данные, эта возможность дает важный способ оптимизации некоторых алгоритмов на процессорах с вычислительными блоками, поддерживающими SIMD.

Листинг 19 – Синтаксис команд доступа к памяти с использованием DAB со смещением

```
{XY|YX}Rsq = XDAB Q [Jm += Jn |<Imm8> | <Imm32>] ;
/* в DAB X сдвинуться на одно 32-битное слово от смещения, необходимого для
получения невыровненных данных; в DAB Y сдвинуться ровно на смещение,
необходимое для получения невыровненных данных */
{XY|YX}Rsq = XSDAB Q [Jm += Jn | <Imm8> | <Imm32>] ;
/* в DAB X сдвинуться на одно 16-битное слово от смещения, необходимого для
получения невыровненных данных; в DAB Y сдвинуться ровно на смещение,
необходимое для получения невыровненных данных */
{XY|YX}Rsq = YDAB Q [Jm += Jn | <Imm8> | <Imm32>] ;
/* в DAB Y сдвинуться на одно 32-битное слово от смещения, необходимого для
получения невыровненных данных; в DAB X сдвинуться ровно на смещение,
необходимое для получения невыровненных данных */
(XY|YX}Rsq = YSDAB Q [Jm += Jn | <Imm8> | <Imm32>] ;
/* в DAB Y сдвинуться на одно 16-битное слово от смещения, необходимого для
получения невыровненных данных; в DAB X сдвинуться ровно на смещение,
необходимое для получения невыровненных данных */
{XY|YX}Rsq = XDAB Q [Km += Kn | <Imm8> | <Imm32>] ;
/* в DAB X сдвинуться на одно 32-битное слово от смещения, необходимого для
получения невыровненных данных; в DAB Y сдвинуться ровно на смещение,
```

необходимое для получения невыровненных данных */
{XY|YX}Rsq = XSDAB Q [Km += Kn | <Imm8> | <Imm32>] ;
/* в DAB X сдвинуться на одно 16-битное слово от смещения, необходимого для получения невыровненных данных; в DAB Y сдвинуться ровно на смещение, необходимое для получения невыровненных данных */
(XY|YX}Rsq = YDAB Q [Km += Kn | <Imm8> | <Imm32>] ;
/* в DAB Y сдвинуться на одно 16-битное слово от смещения, необходимого для получения невыровненных данных; в DAB X сдвинуться ровно на смещение, необходимое для получения невыровненных данных */
{XY|YX}Rsq = YSDAB Q [Km += Kn | <Imm8> | <Imm32>] ;
/* в DAB Y сдвинуться на одно 16-битное слово от смещения, необходимого для получения невыровненных данных; в DAB X сдвинуться ровно на смещение, необходимое для получения невыровненных данных */
/* m должно быть равно 0, 1, 2 или 3 при использовании обратного переноса или кольцевых буферов */

Тип доступа указывается битами группы доступа в команде доступа к данным. Более подробно о декодировании таких команд см. таблицу В-20.

Выравнивания данных в памяти при доступе к памяти с использованием DAB со смещением представлены в таблице 19.

Таблица 19 – Выравнивание данных в памяти при доступе к памяти с использованием DAB со смещением[1]

Адрес2	SDAB	SDAB+смещение	DAB	DAB+смещение
000	Старые значения, поз. [7:0]	Новое значение, поз. [0], старые значения, поз. [7:1]	Старые значения, поз. [3:0]	Новое значение, поз. [0], старые значения, поз. [3:1]
001	Новое значение, поз. [0], старые значения, поз. [7:1]	Новые значения, поз. [1:0], старые значения, поз. [7:2]	Новое значение, поз. [0], старые значения, поз. [3:1]	Новые значения, поз. [1:0], старые значения, поз. [3:2]
010	Новые значения, поз. [1:0], старые значения, поз. [7:2]	Новые значения, поз. [2:0], старые значения, поз. [7:3]	Новые значения, поз. [1:0], старые значения, поз. [3:2]	Новые значения, поз. [2:0], старое значение, поз. [3]
011	Новые значения, поз. [2:0], старые значения, поз. [7:3]	Новые значения, поз. [3:0], старые значения, поз. [7:4]	Новые значения, поз. [2:0], старое значение, поз. [3]	Новые значения, поз. [3:0]
100	Новые значения, поз. [3:0], старые значения, поз. [7:4]	Новые значения, поз. [4:0], старые значения, поз. [7:5]		
101	Новые значения, поз. [4:0], старые значения, поз. [7:5]	Новые значения, поз. [5:0], старые значения, поз. [7:6]		
110	Новые значения, поз. [5:0], старые значения, поз. [7:6]	Новые значения, поз. [6:0], старое значение, поз. [7]		
111	Новые значения, поз. [6:0], старое значение, поз. [7]	Новые значения, поз. [7:0]		

7.1.3.3 Адресация кольцевых буферов

IALU поддерживают адресацию *кольцевых буферов* – диапазона адресов, содержащего данные, неоднократно в цикле пробегаемые и обрабатываемые IALU таким образом, что при выходе из диапазона происходит переход на его начало и дальнейшая обработка данных. Для указания режима адресации кольцевых буферов в командах чтения и записи в память используется оператор CB.

При адресации кольцевого буфера IALU двигает указатель адреса по буферу, постмодифицируя его на положительное или отрицательное значение модификатора при каждой операции доступа к памяти. Если при этом указатель адреса выходит за границы буфера, IALU вычитает из него или прибавляет к нему длину буфера, возвращая тем самым указатель во внутрь буфера.

Для адресации кольцевых буферов IALU задействует регистровый файл и специальные регистры кольцевых буферов. В режиме кольцевой адресации эти регистры имеют следующее назначение:

- В регистре указателя адреса хранится положительное значение (адрес), который выводится IALU на шину адреса. В сводках по синтаксису команд регистр указателя адреса обозначен как *Jm* или *Km.* Этими регистрами могут быть регистры J3-J0 в J-IALU или K3-K0 в K-IALU.
- Модификатор адреса – это значение (положительное или отрицательное), которое IALU прибавляет к регистру указателя адреса в конце каждой операции доступа к памяти. Значение модификатора может храниться в любом регистре регистрового файла того же IALU, в котором находится регистр указателя адреса. Также, вместо регистра, значение модификатора адреса может быть взято из поля непосредственной константы. Абсолютное значение модификатора, расположен ли он в регистре или задан непосредственной константой, должно быть меньше длины кольцевого буфера.
- Регистр длины буфера должен соответствовать регистру указателя адреса, например, J0 используется с JL0, K0 используется с KL0 и т.д. Регистр длины буфера определяет размер кольцевого буфера и диапазон адресов, который пробегает указатель адреса IALU. Если значение в регистре длины буфера равно нулю, то режим адресации кольцевых буферов не используется.
- Регистр базового адреса должен соответствовать регистру указателя адреса, например, J0 используется с JB0, K0 используется с KB0 и т.д. Для определения необходимости перехода через границу буфера после каждой операции доступа к памяти IALU сравнивает значение модифицированного указателя адреса со значением в регистре базового адреса (начальный адрес буфера) и значением в регистре базового адреса плюс значение в регистре длины буфера. Начальное значение указателя адреса циклического буфера должно быть в пределах этого буфера (базовый адрес <= указатель адреса < базовый адрес + длина буфера).

Адресация кольцевых буферов может использовать только вид адресации с постмодификацией. IALU не поддерживает вид адресации с предмодификацией для кольцевых буферов, так как для кольцевых буферов необходимо вычислять новое значение указателя адреса при каждом доступе к памяти.

Если в регистре *JL/KL* ноль, режим адресации кольцевых буферов не используется.

В листинге 20 представлен пример кода, демонстрирующий поддержку кольцевых буферов в IALU, а описание последовательности обращений к памяти для него представлено на рисунке 114.

Как показано в листинге 20 для настройки кольцевого буфера в программе необходимо выполнить следующие действия:

- Загрузить начальный адрес внутри буфера в регистр указателя адреса выбранного J-IALU или K-IALU. В J-IALU регистрами указателя могут быть регистры J3-J0. В K-IALU - регистры K3-K0.
- Загрузить базовый адрес буфера в регистр базового адреса, соответствующий выбранному регистру указателя адреса. Например, JB0 соответствует J0.
- Загрузить длину буфера в регистр длины буфера, соответствующий выбранному регистру указателя адреса. Например, JL0 соответствует J0.
- Загрузить значение модификатора (размер приращения) в регистр регистрового файла того же IALU, что и регистр указателя адреса. Регистрами регистрового файла J-IALU являются J30-J0, K-IALU – K30-K0. В качестве значения модификатора можно использовать также непосредственное значение, задаваемое в команде.

Листинг 20 – Пример адресации кольцевых буферов

```
section program ;
JB0 = 0x100000 ;; JL0 = 11 ;;  /* Установка  базового адреса */
J0 = 0x100000 ;;  /*  Установка  начального адреса */ /*  Установка  длины буфера */
XR0 = CB [J0  +=  4] ;;  /*  Загружается  с адреса  0x100000  */
XR0 = CB [J0  +=  4] ;;  /*  Загружается  с адреса  0x100004  */
XR0 = CB [J0  +=  4] ;;  /*  Загружается  с адреса  0x100008  */
XR0 = CB [J0  +=  4] ;;  /*  Загружается  с адреса  0x100001  */
XR0 = CB [J0  +=  4] ;;  /*  Загружается  с адреса  0x100005  */
XR0 = CB [J0  +=  4] ;;  /*  Загружается  с адреса  0x100009  */
XR0 = CB [J0  +=  4] ;;  /*  Загружается  с адреса  0x100002  */
XR0 = CB [J0  +=  4] ;;  /*  Загружается  с адреса  0x100006  */
XR0 = CB [J0  +=  4] ;;  /*  Загружается  с адреса  0x10000A  */
XR0 = CB [J0  +=  4] ;;  /*  Загружается  с адреса  0x100003  */
XR0 = CB [J0  +=  4] ;;  /*  Загружается  с адреса  0x100007  */
XR0 =  CB[J0 += 4] ;; /* Возвращается для загрузки опять с адреса 0x100000 */
```

На рисунке 114 показан порядок, в котором программа, приведенная в листинге 20, читает 11 ячеек буфера за один проход. На 12-ом обращении происходит возврат к началу кольцевого буфера.

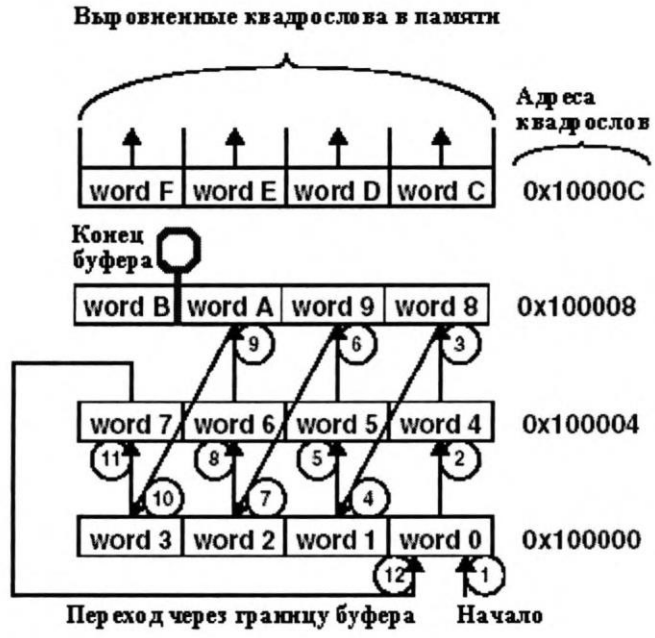


Рисунок 114 – Адресация кольцевых буферов – порядок обращений к памяти

Заметьте, что опция вычисления адреса в кольцевом буфере применима только к командам IALU сложения/вычитания (например, J5 = J7 + J10 (CB);;).

7.1.3.4 Бит-реверсивная адресация

IALU поддерживает бит-реверсивную адресацию посредством оператора бит-реверса (BR). Использование этого оператора с косвенным чтением или записью с постмодификацией приводит к тому, что при вычислении постмодификации бит переноса перемещается вправо (вместо того, чтобы перемещаться влево).

На рисунке 115 представлен пример операции с обратным переносом (бит-реверсом). При обычной операции сложения 0xA5A5 + 0x2121 в результате получается 0xC6C6. Та же операция, но с обратным переносом, даст 0x9494.

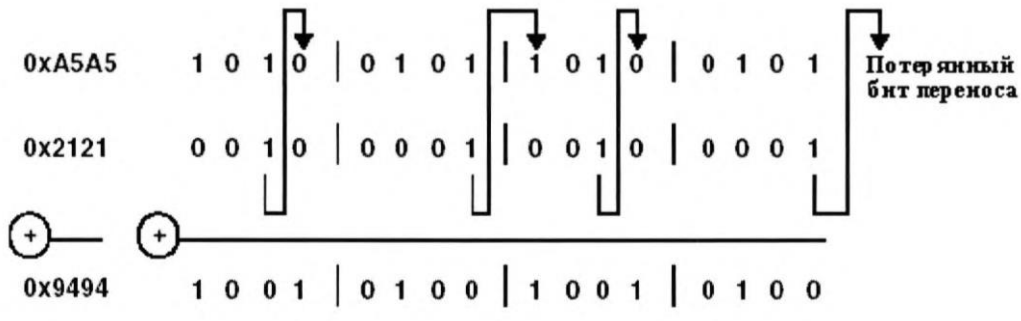


Рисунок 115 – Операция с обратным переносом (опция BR)

Как и операции с кольцевыми буферами, операция бит-реверсивной адресации может быть выполнена только с использованием регистров J3-J0 или K3-K0. Однако, в отличие от кольцевых буферов, в случае бит-реверсивной адресации нет необходимости предварительно устанавливать значения соответствующих регистров IALU базового адреса и длины.

В листинге 21 приведен пример программы, использующей бит-реверсивную адресацию. Последовательность обращений к памяти при использовании обратного переноса представлена на рисунке ниже.

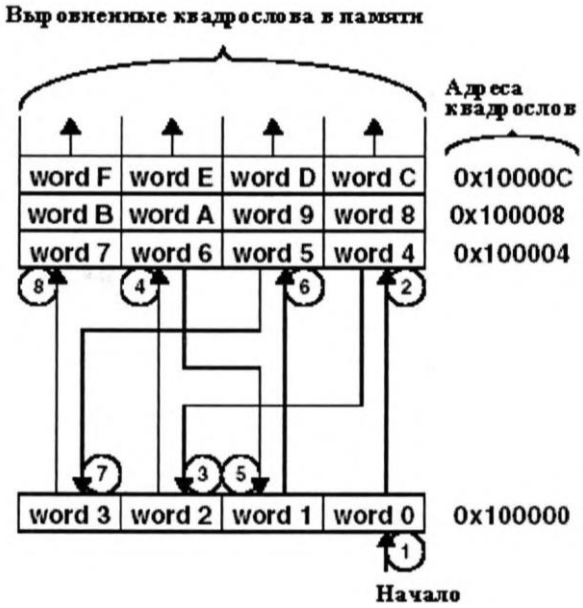


Рисунок 116 – Бит-реверсивная адресация - порядок обращений к памяти

Листинг 21 - Пример бит-реверсивной адресации

```
#define N8      / * N = 8 ;        количество ячеек для бит-реверсивной адресации;
N должно быть степенью 2 */
.section data1;
.align N;
/* выравнивание начала входного буфера на адрес, делящийся нацело на N; в этом
примере предполагается, что адрес равен 0x1000 0000 0000 0000 */
.var input[N]={0,1,2,3,4,5,6,7};
.var output[N];
ection program;
_main:
j0 = j31 + ADDRESS(input) ;;          /* Указатель на входные данные */
j4 = j31 + ADDRESS(output) ;;         /* Указатель на выходные данные*/
LC0 = N ;;                            /* Установка счетчика циклов */
_my_loop:
xr0 = BR [J0 += N/2] ;;               /* Чтение данных      с бит-реверсом; модификатор
                                      должен быть равен N/2 */
if NLC0E, jump _my_loop ; [j4+=1] =xr0 ;; /*Записываем данные линейно */
```

Несколько важных моментов, на которые стоит обратить внимание при чтении листинга 21:

- Количество ячеек для бит-реверсивной адресации (длина буфера) должно быть степенью двух (например, длине буфера = 2^n), однако, начальный адрес используемого буфера данных должен быть выровнен на адрес, нацело делящийся на число ячеек в буфере. В данном примере длина буфера равна восьми, а начальный адрес буфера выровнен на адрес, делящийся нацело на восемь.
- Ассемблер может вычислять значения выражений. Например, везде, где

в коде встречается N/2, ассемблер вычисляет это значение и подставляет четыре. Оператор ассемблера ADDRESS ( ) вычисляет адрес символа.

- В каждой из повторяющейся в цикле (_my_loop) операции бит-реверсивного доступа к памяти данные, записанные по адресу в J4 на каждой итерации цикла равны 0, 4, 2, 6, 1, 5, 3 и 7. Обратите внимание на то, что на каждом повторном чтении памяти используется бит-реверсивная адресация. Обратите также внимание и на то, как она влияет на вычисление адреса при каждом доступе (Таблица 20).
- На листинге для зацикливания чтения и записи памяти используется команда условного перехода. Более подробно об отработке условных команд см. раздел «Отработка условных команд».

Таблица 20 – Операции постмодификации при сложении с обратным переносом

Итерация цикла	**J0**	**XR0**	**BR [J0 += 4], случаи использования обратного переноса помечены (BRC, bit reverse carry) обозначает 1000 0000 0000**
0	0x1000 0000	0x0	b# ...0000+ b#0100 = b#...0100
1	0x1000 0004	0x4	b#. ..0100+b#0100 = b#...0010 (BRC)
2	0x1000 0002	0x2	b# ...0010+b#0100 = b#...0110
3	0x1000 0006	0x6	b# ...0110+b#0100 = b#...0001 (BRC)
4	0x1000 0001	0x1	b# ...0001 +b#0100 =b#...0101
5	0x1000 0004	0x4	b# ...0101 +b#0100 = b#...0011 (BRC)
6	0x1000 0003	0x3	b# ...0011 +b#0100 = b#...0111
7	0x1000 0007	0x7	b# ...0111 +b#0100 = b#...0000 (BRC)

Заметьте, что опция вычисления адреса с обратным переносом применима только к командам IALU сложения/вычитания (например, J5 = J7 + J10 (BR);;).

7.1.3.5 Операции пересылки универсальных регистров

IALU обеспечивает пересылку данных между универсальными регистрами. Также IALU обеспечивает загрузку 15- или 32-битных непосредственных данных в универсальные регистры. Команды пересылки и загрузки Ureg, поддерживаемые IALU, включают в себя:

```
Ureg_s = <Imm15>|<Imm32> ;      /* загрузка простого Ureg 15- или 32-битной непосредственной константой */
Ureg_s = Ureg_m ;   /* пересылка содержимого простого (32-битного) Ureg в другой Ureg */
Ureg_sd = Ureg_md ; /* пересылка содержимого двойного (64-битного) Ureg в другой Ureg */
/* Нумерованные регистры регистровых файлов вычислительных блоков или IALU могут использоваться как двойные регистры */
Ureg_sq = Ureg_mq ; /* пересылка содержимого квадро (128-битного) Ureg в другой Ureg */
/* Нумерованные регистры регистровых файлов вычислительных блоков или IALU могут использоваться как квадро-регистры */
```

Заметьте, что существует много вариантов пересылок, так как регистры регистровых файлов вычислительных блоков могут быть также использованы как квадро-регистры.

Например:

```
XYR3:0 = XYR31:28 ;;
/* Эта команда эквивалентна (XR3:0 = XR31:28; YR3:0 = YR31:28;), но занимает лишь одно поле команды и одно поле IALU. Операции вида XYR = YXR также допустимы. */
```

Операции непосредственного расширения

Многие команды IALU допускают непосредственные данные в качестве операнда. Если непосредственные данные занимают больше 8 бит для команд адресации или больше 15 бит для команд загрузки универсальных регистров, то эти данные не могут поместиться в одном 32-битном слове команды. Для хранения таких больших непосредственных данных DSP использует два поля команды: одно поле для самой команды, а второе – для непоместившихся данных.

Ассемблер поддерживает автоматическое использование непосредственного расширения, но программист должен иметь в виду, что та или иная команда требует непосредственного расширения, и резервировать поле для этого расширения. Например, использовать три (не четыре) поля в командной строке и поместить команду, требующую расширения, в первое поле командной строки, в этом случае DSP должен автоматически использовать второе поле для непосредственного расширения.

Заметьте, что в одной командной строке может быть использовано лишь одно непосредственное расширение. Например, если непосредственное расширение использовано командой перехода на непосредственный адрес, то это единственное допустимое непосредственное расширение в данной командной строке.

7.1.3.6 Примеры программирования IALU

В листингах 22 и 23 представлено несколько примеров арифметических команд и команд адресации данных IALU. Ключевые особенности каждой команды, такие как формат операндов с фиксированной или плавающей точкой, размер операндов, а также использование регистров, перечислены в комментариях к команде.

Листинг 22 – Примеры команд IALU

```
J1 = J0 + 0x81 ; /* Js = Jm + Imm8 данные */
K2 = ROTR K0 ; /* Ks = сдвинутый на 1 бит вправо Km */
SQCTL = [J3 + J5] ; /* Загрузить в Ureg данные с адреса Jm + Jn */
J5:4 = L [J2 += 0x81] ; /* Загрузить в Ureg_sd данные с адреса Jm и
постмодифицировать Jm на Imm8*/
J31:28 = Q [K2 + K5] ; /* Загрузить в Ureg_sq данные с адреса Km + Kn */
XR2 = CB Q [J0 += 0x8181] ; /* Загрузить в Rs данные с адреса Jm; пост-
модицировать Jm на Imm8, используя адресацию кольцевых буферов; команда
использует поле непосредственного расширения */

CJMP = 0x8181 ;     /* Загрузить в Ureg_s константу Imm32; */
K31 = J31 ;         /* Загрузить в Ureg_s содержимое Ureg_m */
YR3:0 = XR7:4 ;     /* Загрузить в Ureg_sq содержимое Ureg_mq */
```

Листинг 23 – Пример использования DAB

```
/* программа загружает все элементы input_buffer, используя простые
квадропересылки в регистры yr15:0. Все загрузки выполняются без использования
DAB так как input_buffer выровнен на квадрослово */
j0 = j 31 + input_buffer;;
yr3:0 = q[j0+=4];;
/* yr0 = 0x00001111, yr1 = 0x22223333, yr2 = 0x44445555, yr3 = 0x66667777 */
yr7:4 = q[j0+=4];; /* yr4 = 0x88889999, yr5 = 0xaaaabbbb, yr6 = 0xccccdddd,
yr7 = 0xeeeeffff */
yr11:8 = q[j0+=4];;
/* yr8 = 0x11110000, yr9 = 0x33332222, yr10 = 0x55554444, yr11 = 0x77776666 */
yr15:12 = q[j0+=4];;
/* yr12 = 0x99998888, yr13 = 0xbbbbaaaa, yr14 = 0xddddcccc, yr15 = 0xffffeeee */
```

```
/* теперь программа загружает все элементы input_buffer, интерпретируя тот как
кольцевой буфер, начиная с третьего элемента буфера и используя DAB, в
регистры XR15:0 */
j0 = j31 + input_buffer + 2;; jb0 = j31 + input_buffer;;
j10 = j31 + 16;;
xr3:0 = dab q[j0+=4];;
/* xDAB инициализируется данными по адресу ближайшего квадрослова;
xDAB=0x00001111, 0x22223333, 0x44445555, 0x66667777 */
xr3:0 = dab q[j0+=4];;
/* xr0 = 0x44445555,xr1 = 0x66667777, xr2 = 0x88889999, xr3 = 0xaaaabbbb */
xr7:4 = dab q[j0+=4];;
/* xr4 =0xccccdddd, xr5 = 0xeeeeffff, xr6 = 0x11110000, xr7 = 0x33332222 */
xr11:8 = dab q[j0+=4];;
/* xr8 =0x55554444, xr9 = 0x77776666, xr10 = 0x99998888, xr11 = 0xbbbbaaaa */
xr15:12 = dab q[j0+=4];;
/* xr12 =0xddddcccc, xr13 = 0xffffeeee, xr14 = 0x00001111, xr15 = 0x22223333 */
/* теперь программа загружает все короткие элементы input_buffer, интерпретируя его
как кольцевой буфер, начиная с четвертого короткого элемента буфера и, используя
DAB, в регистры YR31:16 */
j0 = j31 + 2*input_buffer + 3;; jb0 = j31 + 2*input_buffer;;
j10 = j31 + 32;;
yr19:16 = sdab q[j0+=8];;
/* yDAB инициализируется данными по адресу ближайшего квадрослова;
yDAB=0x00001111, 0x22223333, 0x44445555, 0x66667777; ysDAB=0x1111,
0x0000,0x3333, 0x2222, 0x5555, 0x4444, 0x7777, 0x6666 */

yr19:16 = sdab q[j0+=8];;
/* yDAB теперь содержит 0x88889999, 0xaaaabbbb, 0xccccdddd, 0xeeeeffff;
ysDAB=0x9999, 0x8888, 0xbbbb,
0xaaaa, 0xdddd, 0xcccc, 0xffff, 0xeeee */
/* 0x2222 и 0x5555 пересылаются в yr16=0x55552222 */
/* 0x4444 и 0x7777 пересылаются в yr17=0x77774444 */
/* 0x6666 и 0x9999 пересылаются в yr18=0x99996666 */
/* 0x8888 и 0xbbbb пересылаются в yr19=0xbbbb8888 */
/* yDAB теперь содержит 0x88889999, 0xaaaabbbb, 0xccccdddd, 0xeeeeffff;
ysDAB=0x9999, 0x8888, 0xbbbb, 0xaaaa, 0xdddd, 0xcccc, 0xffff, 0xeeee */

yr23:20 = sdab q[j0+=8];;
/* yDAB теперь содержит 0x11110000, 0x33332222, 0x55554444, 0x77776666;
ysDAB=0x000, 0x1111, 0x2222, 0x3333, 0x4444, 0x5555, 0x6666, 0x7777 */
/* 0xaaaa и 0xdddd пересылаются в yr20=0xddddaaaa */
/* 0xcccc и 0xffff пересылаются в yr21=0xffffcccc */
/* 0xeeee и 0x0000 пересылаются в yr22=0x0000eeee */
/* 0x1111 и 0x2222 пересылаются в yr23=0x22221111 */
yr27:24 = sdab q[j0+=8];;
/* yDAB теперь содержит 0x99998888, 0xbbbbaaaa, 0xddddcccc, 0xffffeeee;
ysDAB=0x8888, 0x9999, 0xaaaa, 0xbbbb, 0xcccc, 0xdddd, 0xeeee, 0xffff */

/* 0x3333 и 0x4444 пересылаются в yr24 = 0x44443333 */
/* 0x5555 и 0x6666 пересылаются в yr25 = 0x66665555 */
/* 0x7777 и 0x8888 пересылаются в yr26 = 0x88887777 */
/* 0x9999 и 0xaaaa пересылаются в yr27 = 0xaaaa9999 */
```

```
yr31:28 = sdab q[j0+=8];;
/* yDAB теперь содержит 0x00001111, 0x22223333, 0x44445555, 0x66667777;
ysDAB=0x1111, 0x0000, 0x3333, 0x2222, 0x5555, 0x4444, 0x7777, 0x6666 */
/* 0xbbbb и 0xcccc пересылаются в yr28 = 0xccccbbbb */
/* 0xdddd и 0xeeee пересылаются в yr29 = 0xeeeedddd */
/* 0xffff и 0x1111 пересылаются в yr30 = 0x1111ffff */
/* 0x0000 и 0x3333 пересылаются в yr31 = 0x33330000 */
```

7.2 Сводка команда IALU

Оператор L перед входным или выходным операндом (косвенная или прямая адресация) указывает на чтение или запись длинного слова (64-битного). Например, следующий синтаксис команды указывает на чтение длинного слова. В командах адресации данных IALU специальные операторы определяют размеры входных и выходных операндов по следующим правилам: (шаблоны *Rsd, Rmd, Ureg_sd, Ureg_md)* или квадрорегистров (шаблоны *Jm, Jn, Js, Rs, Ureg_s),* двойных регистров, вместо которых могут быть подставлены имена одиночных строк.

В следующих листингах показан синтаксис команд IALU:

Листинг 24 «Арифметические, логические и функциональные команды IALU»;

Листинг 25 «Команды IALU загрузки Ureg-регистров (адресация данных)»;

Листинг 26 «Команды IALU загрузки Dreg-регистров (адресация данных и операции с DAB)»;

Листинг 27 «Команды IALU сохранения Ureg-регистров (адресация данных)»;

Листинг 28 «Команды IALU загрузки Dreg-регистров (адресация данных)»;

Листинг 29 «Команды IALU пересылки универсальных регистров»;

Листинг 30 «Команды IALU байтовой и полусловной адресации».

Соглашения по обозначениям имён регистров, необязательных элементов или вариантов, используемые в справках на команды, подробно освещены в разделе «Регистры регистровых файлов». Вкратце эти соглашения таковы:

- { } - необязательные элементы указаны в фигурных скобках; фигурные скобки не являются частью синтаксиса команды;
- | - вертикальная черта разделяет варианты значений элементов; вертикальные черты не являются частью синтаксиса команды;
- *Jm* или *Km* - буква J или K в именах регистров, выделенных курсивом, указывает на регистр J-IALU или K-IALU.

Листинг 24 – Арифметические, логические и функциональные команды IALU

```
Js = Jm + |- Jn | <Imm8>|<Imm32> {({CJMP|CB|BR}{NF}) } ;
JB0|JB1|JB2|JB3|JL0|JL1|JL2|JL3 = Jm +|-Jn|<Imm8>\<Imm32> {(NF)};
Js = Jm + Jn|< Imm 8>|< Imm 32> + JC {(NF)} ;
Js = Jm - Jn | <Imm8> | <Imm32> + JC - 1 { (NF) ) ;
Js = {Jm +|- Jn | <Imm8>|<Imm32 >)/2 {(NF)} ;
COMP( Jm, Jn | <Imm8>|<Imm32>) {(U) } ;
Js = MAX|MIN (Jm, Jn |< Imm8>|< Imm32 >) {(NF)} ;
Js = ABS Jm {(NF)} ;
Js = Jm OR|AND|XOR|AND NOT Jn |< Imm8 >|< Imm32> {(NF)} ;
Js = NOT Jm {(NF)} ;
Js = ASHIFTR|LSHIFTR Jm {(NF)} ;
Js = ROTR|ROTL Jm {(NF)} ;
Ks = Km +|- Kn|<Imm8>|<Imm32> {({CJMP|CB|BR}{NF})} ;
KB0|KB1|KB2|KB3|KL0|KL1|KL2|KL3 = Km +|-Kn|<Imm8>|<Imm32> {(NF)};
Ks = Km + Kn | <Imm8> | <Imm32> + KC { (NF) } ;
Ks = Km - Kn | <Imm8> | <Imm32> + KC - 1 { (NF) } ;
```

```
Ks = (Km +|- Kn |<Imm8>|<Imm32>)/2 {(NF) } ;
COMP(Km, Kn|<Imm8>|<Imm32>) {(U)} ;
Ks = MAX|MIN (Km, Kn|<Imm8>|<Imm32>) {(NF)} ;
Ks = ABS Km {(NF)} ;
Ks = Km OR|AND|XOR|AND NOT Kn|<Imm8>\<Imm32> {(NF)} ;
Ks = NOT Km {(NF)} ;
Ks = ASHIFTR|LSHIFTR Km {(NF)} ;
Ks = ROTR|ROTL Km {(NF)} ;
Ks = LSHIFTL Km by <Imm5> {(NF)};
Ks = ASHIFTR Km by <Imm5> {(NF)};
Ks = LSHIFTR Jm by <Imm5> {(NF)};
Js = BITREV Jm {(NF)};
atomic; <операция J-АЛУ>; <операция XY-АЛУ>; <операция K-АЛУ >;;
```

Листинг 25 – Команды IALU загрузки Ureg-регистров (адресация данных)

```
Ureg_s = {CB|BR} [Jm +|+= Jn|<Imm8> | <Imm32>] ;
Ureg_sd = {CB|BR} L [Jm+ |+=Jn |<Imm8>|<Imm32>] ;
Ureg_sq = {CB|BR} Q [Jm+ |+=Jn |<Imm8>| <Imm32>] ;
Ureg_s = {CB|BR} [Km +|+= Kn |<Imm8>|<Imm32>] ;
Ureg_sd = {CB|BR} L [Km +|+= Kn |<Imm8>|<Imm32>] ;
Ureg_sq = {CB|BR} Q [Km + | += Kn|<Imm8>|<Imm32>] ;
/* суффикс при Ureg означает: _s = обычный, _sd = двойной, _sq = квадрорегистр */
```

Листинг 26 – Команды IALU сохранения Dreg-регистров (адресация данных и операции с DAB)

```
{X|Y|XY}Rs = {CB|BR} [Jm += Jn|<Imm8>|<Imm32>] ;
{X|Y|XY}Rsd = {CB|BR} L [Jm += Jn|<Imm8>|<Imm32>] ;
{XY|YX}Rs = {CB|BR} L [Jm += Jn |<Imm8>|<Imm32>] ;
{X|Y|XY}Rsq= {CB|BR|DAB|SDAB} Q [Jm += Jn | < Imm8>|< Imm32>] ;
{XY|YX}Rsd = {CB|BR} Q [Jm += Jn|<Imm8>|<Imm32>] ;
{XY}Rsq = XDAB Q [Jm += Jn |<Imm8>|<Imm32>] ;
{XY}Rsq = XSDAB Q [Jm += Jn|<Imm8>|<Imm32>] ;
{XY}Rsq = YDAB Q [Jm += Jn |<Imm8>|<Imm32>] ;
{XY}Rsq = YSDAB Q [Jm += Jn | < Imm8>|<Imm32>] ;
{X|Y|XY}Rs = {CB|BR} [Km += Kn|<Imm8>|<Imm32>] ;
{X|Y|XY}Rsd = {CB|BR} L [Km += Kn|<Imm8>|<Imm32>] ;
{XY|YX}Rs = {CB | BR} L [Km += Kn |<Imm8>| <Imm32>] ;
{X|Y|XY} Rsq = {CB|BR|DAB|SDAB} Q [Km += Kn|<Imm8>|<Imm32>] ;
{XY|YX}Rsd = {CB|BR) Q [Km += Kn |<Imm8>|<Imm32>] ;
{XY}Rsq = XDAB Q [Km += Kn |<Imm8>|<Imm32>] ;
{XY}Rsq = XSDAB Q [Km += Kn|<Imm8>|<Imm32>] ;
{XY}Rsq = YDAB Q [Km += Kn |<Imm8>|<Imm32>] ;
{XY}Rsq = YSDAB Q [Km += Kn|<Imm8>|<Imm32>] ;
/* суффикс при R означает: _s = обычный, _sd = двойной, _sq = квадро-регистр */
/* m должно быть равно 0, 1, 2 или 3 для бит-реверсивных и кольцевых буферов */
```

Листинг 27 – Команды IALU сохранения Ureg-регистров (адресация данных)

```
    [Jm+ |+= Jn|<Imm8>|<Imm32>] = Ureg_s ;
L   [Jm+ |+= Jn|<Imm8>|<Imm32>] = Ureg_sd ;
Q   [Jm+ |+= Jn|<Imm8>|<Imm32>] = Ureg_sq ;
    [Km + |+= Kn|<Imm8>|<Imm32>] = Ureg_s ;
L   [Km +|+= Kn|<Imm8>|<Imm32>] = Ureg_sd ;
```

```
Q  [Km +|+= Kn|< Imm8>|<Imm32>] = Ureg_sq ;
```

Листинг 28 – Команды IALU сохранения Dreg-регистров (адресация данных)

```
{CB|BR} [Jm  += Jn |<Imm8>|<Imm32>] = {X|Y}Rs ;
{CB|BR} L  [Jm  += Jn |<Imm8>|<Imm32>]= {X|Y}Rsd ;
{CB|BR} L [Jm += Jn |<Imm8>|<Imm32>] = {XY|YX}Rs ;
{CB|BR} Q  [Jm  += Jn |<Imm8>|<Imm32>] = {X|Y}Rsq ;
{CB|BR} Q [Jm += Jn |<Imm8>|<Imm32>] = {XY|YX}Rsd ;

{CB|BR} [Km  += Kn |<Imm8>|<Imm32>] = {X|Y}Rs ;
{CB|BR} L  [Km  += Kn |<Imm8>|<Imm32>] = {X|Y}Rsd ;
{CB|BR} L [Km  += Kn |<Imm8>|<Imm32>] = {XY|YX}Rs ;
{CB|BR} Q  [Km  += Kn |<Imm8>|<Imm32>] = {X|Y}Rsq ;
{CB|BR} Q [Km  += Kn |<Imm8>|<Imm32>] = {XY|YX}Rsd ;


/* суффикс при R означает: _s = обычный, _sd = двойной _sq = квадро-регистр */
/* m должно быть равно 0, 1, 2 или 3 для бит-реверсивных и кольцевых буферов */
```

Листинг 29 – Команды IALU пересылки универсальных регистров

```
Ureg_s = <Imm15> | <Imm32> ;
Ureg_s = Ureg_m ;
Ureg_sd = Ureg_md ;
Ureg_sq = Ureg_mq ;
```

Листинг 30 – Команды IALU байтовой и полусловной адресации

```
{X|Y|XY} Rs = BDAB [Jm +|+= Jn|<Imm8>|<Imm32>] ;
{X|Y|XY} Rs = BDAB L[Jm +|+= Jn|<Imm8>|<Imm32>] ;
{X|Y|XY} Rs = PB [Jm +|+= Jn|<Imm8>|<Imm32>] ;
{X|Y|XY} Rsd = LPB [Jm +|+= Jn|<Imm8>|<Imm32>] ;
PB [Jm +|+= Jn|<Imm8>|<Imm32>] = Ureg;
{X|Y|XY} Rs = SDAB [Jm +|+= Jn|<Imm8>|<Imm32>] ;
{X|Y|XY} Rs = SDAB L[Jm +|+= Jn|<Imm8>|<Imm32>]
{X|Y|XY} Rs = PS [Jm +|+= Jn|<Imm8>|<Imm32>] ;
{X|Y|XY} Rsd = LPS [Jm +|+= Jn|<Imm8>|<Imm32>] ;
PS [Jm +|+= Jn|<Imm8>|<Imm32>] = Ureg;
Ureg = B [Jm +|+= Jn|<Imm7>|<Imm31>] ({ZF}|{SE});
B [Jm +|+= Jn|<Imm7>|<Imm31>] = bUreg;
Ureg = S [Jm +|+= Jn|<Imm7>|<Imm31>] ({ZF}|{SE});
S [Jm +|+= Jn|<Imm7>|<Imm31>] = sUreg;
```

8 Устройство управления

Ядро процессора содержит устройство управления (УУ), которое в свою очередь, содержит буфер выравнивания команд (IAB – Instruction Alignment Buffer), программный счетчик (PC – Program Counter), буфер целевых адресов перехода (BTB – Branch Target Buffer), контроллер прерываний и механизм выборки команд по адресам. Используя эти средства и конвейер команд, устройство управления (оно выделено на рисунке 117) управляет ходом исполнения программы.

УУ осуществляет выборку команд из памяти, извлекает из выбранных данных на каждом такте командную строку, осуществляет декодирование, определяя для каждой команды в строке, какое устройство должно исполнять ее и отправляет каждую команду на соответствующее устройство. Устройство управления выполняет также команды управления потоком исполнения программ и контролирует исполнение программы, приостанавливая его там, где это требуется, на несколько тактов. Операции, поддерживаемые устройством управления, включают в себя:

- предоставление адреса следующего квадрослова программного кода для выборки в буфер выравнивания команд (IAB);
- извлечение из буфера выравнивания команд следующей исполняемой командной строки;
- исполнение команд управления потоком, в том числе:
  - вычисление предикатов;
  - обслуживание буфера целевых адресов перехода (BTB), позволяющего минимизировать задержку на исполнение перехода;
  - декрементирование счетчиков цикла;
  - приостановка конвейера там, где это требуется;
  - реакция на прерывания.

На рисунке 118 приведена блочная диаграмма устройства управления. Обратите внимание на следующие блоки диаграммы:

- программный счетчик (PC) инкрементирует адрес выборки для линейного потока исполнения и модифицирует адрес требуемым образом для нелинейного потока (переходы, циклы, прерывания и др.);
- буфер целевых адресов перехода (BTB) кэширует адреса переходов с тем, чтобы сократить затраты конвейера на прогнозируемые переходы. Дополнительную информацию см. в подразделе «Отработка переходов»;
- устройство выборки помещает на шину адрес следующего квадрослова, которое должно быть выбрано из памяти;
- буфер выравнивания команд (IAB) получает квадрослова команд из памяти, буферизует их и направляет команды в вычислительные блоки, целочисленные ALU и устройство управления.

УУ поддерживает несколько разновидностей потока исполнения программ, используя функциональные блоки, изображенные на рисунке 118. Преимущественно процессор занят выполнением последовательных команд программы, т.е. поток исполнения линеен. Время от времени линейный характер потока нарушается – это происходит тогда, когда используются программные конструкции, предусматривающие непоследовательное исполнение.

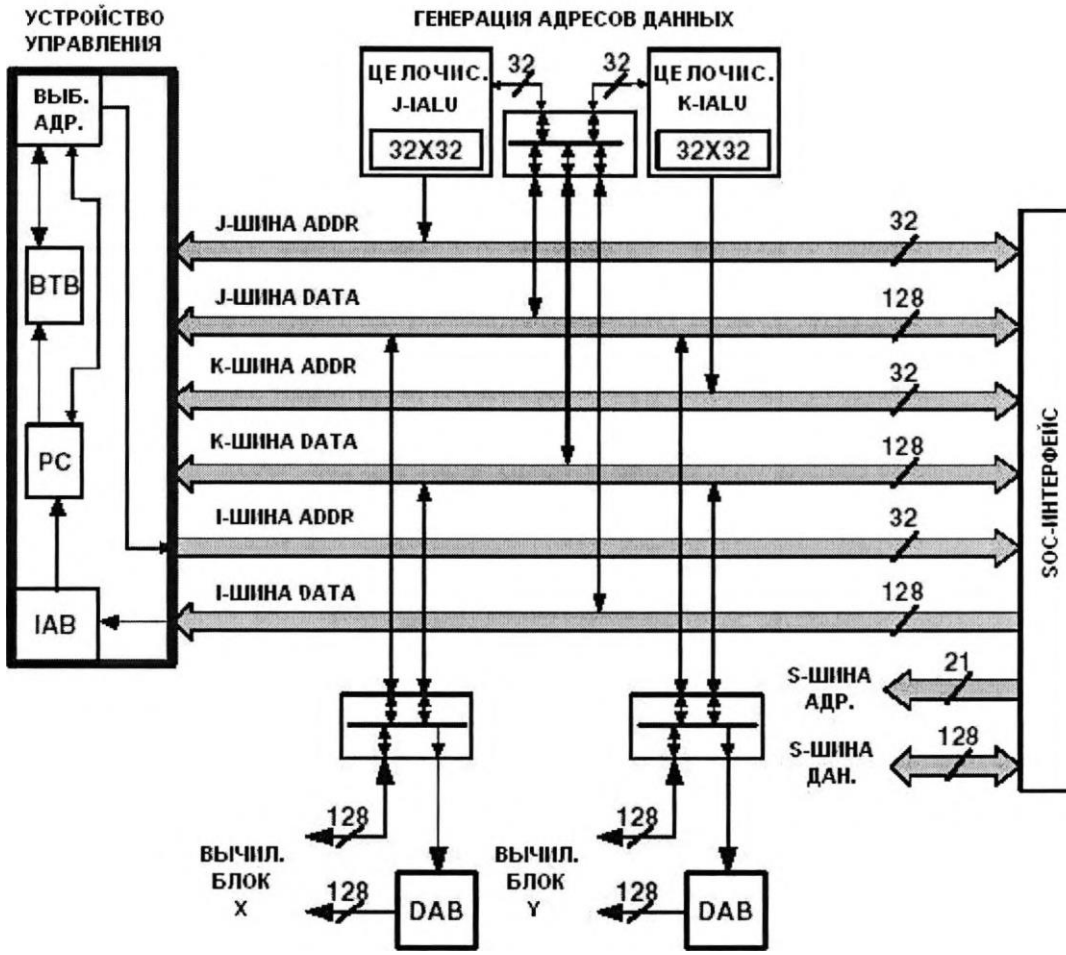


Рисунок 117 – Устройство управления

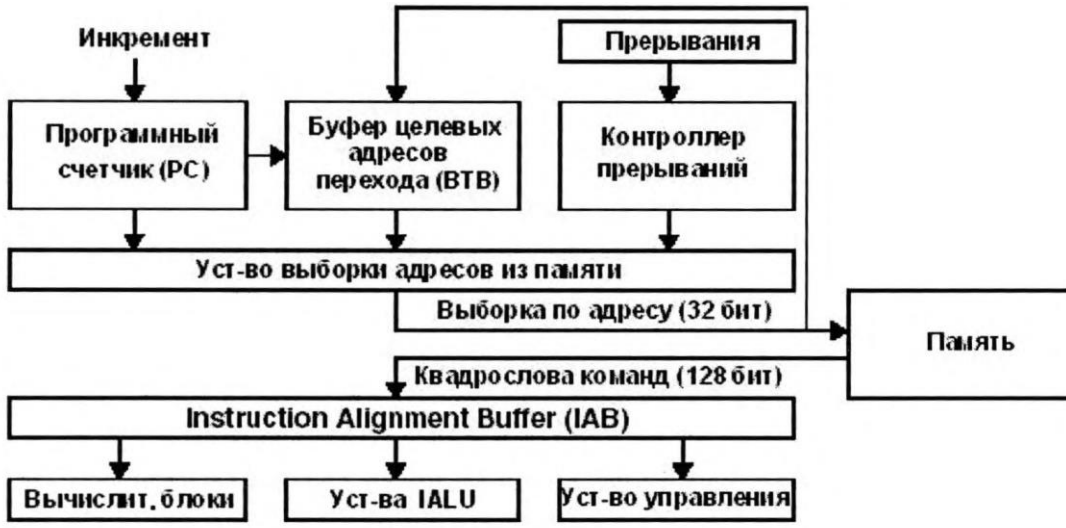


Рисунок 118 – Блочная схема устройства управления

Программные конструкции, предусматривающие непоследовательное исполнение, указывают процессору, что следующей исполняемой командой является не та команда, которая располагается по следующему последовательному адресу.

Такого рода конструкции включают в себя:

- *циклы* – одна и та же последовательность команд исполняется несколько раз с накладными расходами, близкими к нулю;
- *подпрограммы* – процессор временно переключает последовательный поток на исполнение команд из другого участка программной памяти;
- *переходы* – поток исполнения безвозвратно переключается на другой участок программной памяти;
- *прерывания* – подпрограммы, выполнение которых провоцируется событиями, происходящими во время исполнения;
- *бездействие* – команда, которая заставляет процессор прекратить все операции и сохранять текущее состояние до того момента, пока не произойдет прерывание. Далее процессор обрабатывает прерывание и возобновляет нормальное исполнение.

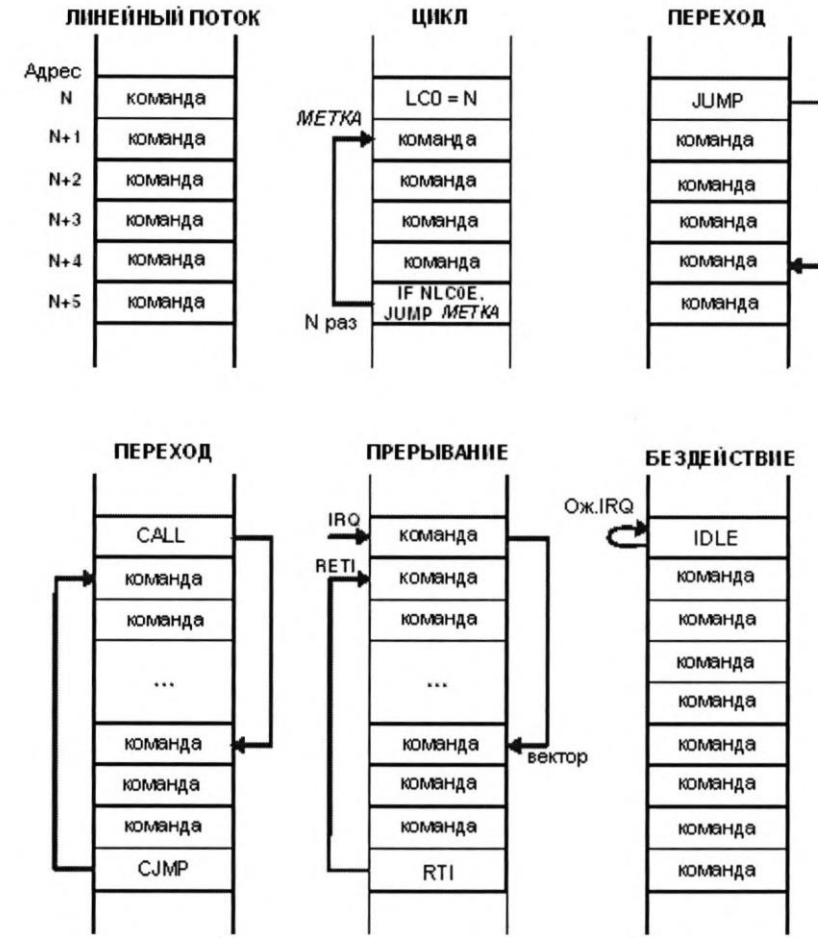


Рисунок 119 – Разновидности потока исполнения программы

Информацию об использовании каждого типа потока исполнения см. в подразделе «Операции устройства управления».

Чтобы обеспечить высокую скорость исполнения, процессор использует конвейер команд. Процессор производит выборку квадрослов из памяти, затем осуществляет их разбиение на командные строки (включающие в себя от одной до четырех команд), декодирует команды и исполняет их.

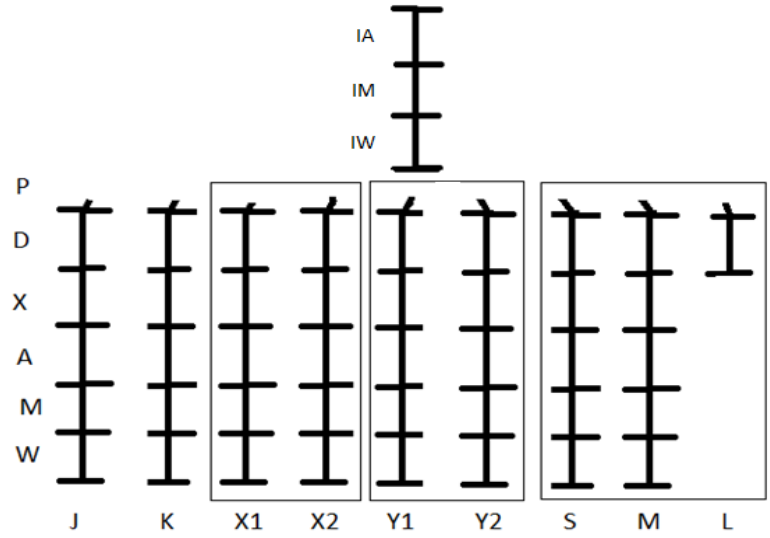


Рисунок 120 – Структура конвейера процессора.

На рисунке 120 горизонтальным линиям конвейера соответствуют регистры для хранения информации между стадиями конвейера, а вертикальные линии – задержке обрабатывающей логики. Самая верхняя горизонтальная линия соответствует регистру PC – указателю на текущую команду устройства управления. PC содержит адрес, который отправляется в подсистему памяти для выборки команды.

- Стадия адреса (стадия IA) соответствует логике декодирования адреса, а также логике арбитража доступа к памяти. На этой стадии адрес команды выдается устройству памяти и BTB. В этот момент BTB прогнозирует, будет ли следующая выборка последовательной или будет иметь место переход, прогнозируемый как «предпринимаемый».
- Следующая стадия IM соответствует чтению данных из накопителя в буфер памяти.
- На стадии IW происходит передача команды из буфера памяти в буфер выравнивания команд IAB. Шина команд имеет ширину 128 бит, что соответствует выборке четырех команд за такт.
- На стадии P одна линия конвейера чтения команд разветвляется на девять линий (ветвей) конвейера исполнения команд. На данной стадии выполняется предварительное декодирование команд.

Из рисунка 120 видно, что модули IALU (J и K) имеют по одной линии конвейера, вычислительные модули X и Y – по две линии конвейера каждый, а модуль YY включает три ветви: S, M, L. Ветвь L короткая, т.к. она обрабатывает команды-расширители константного поля, и по сути вся обработка этой команды сводится к определению, какой линии конвейера соответствует расширение и передачи данных на эту линию. Для модулей обработки J, K, X, Y последняя горизонтальная линия конвейера соответствует РОН, в который записывается результат операции. Название стадий конвейера обработки данных D, X, A, M, W выбрано исходя из логики выполнения команды загрузки из памяти модулей IALU:

D – декодирование команды;
X – исполнение команды (вычисление адреса доступа);
A – передача адреса в подсистему памяти;
M – чтение данных из накопителя в буфер;
W – передача данных из буфера памяти в регистр ядра.

В зависимости от типа команды, операции, выполняемые на данных стадиях конвейера, будут различны. Так для модулей X и У наиболее важными являются стадии A, M, W:

- стадия A – декодирование команды;
- стадия M – первый такт исполнения;
- стадия W – второй (заключительный) такт исполнения.

Стадии M и W для модулей X и У еще называются стадиями исполнения и имеют дополнительное обозначение EX1 и EX2 соответственно.

Одной из задач УУ является отслеживание зависимостей по операндам во всех линиях конвейера и приостановка конвейера в случае, если операнды не готовы. Все линии конвейера продвигаются синхронно. При этом максимум шесть линий из девяти могут исполнять одновременно команды.

Для УУ основной линией конвейера является линия S, на которой происходит обработка команд-переходов. Для такого глубоко конвейеризированного процессора скорость выполнения переходов является одним из ключевых факторов эффективности исполнения программы.

Переходы могут быть безусловными и условными. Условные переходы могут быть двух типов: переходы анализируемые на стадии X конвейера и на стадии W. Последние (на стадии W) соответствуют анализу флагов операций в модулях X и У. Можно видеть, что в случае выполнения перехода на стадии X, процессор потеряет пять тактов, прежде чем следующая команда достигнет стадии X. Для ветвлений на стадии W таких тактов будет уже восемь. Для уменьшения потерь, связанных с переходами, УУ содержит буфер целевых адресов перехода (BTB) и имеет механизм статического прогнозирования переходов (выполняется пользователем). Если переход помечен как прогнозируемый, то он выполняется на стадии D линии S конвейера. Информация о переходе заносится в строку BTB. При первом выполнении данного перехода будут потеряны четыре такта, однако, при последующих его выполнениях потери будут равны 0.

Для полного прохождения конвейера командной строке требуется, как минимум, девять тактов. Производительность исполнения равна одной командной строке за такт ядра процессора (CCLK). Дополнительную информацию см. в подразделе «Операции конвейера команд».

8.1 Операции устройства управления

Устройство управления поддерживает линейное исполнение (один последовательный адрес вслед за другим) и нелинейное исполнение (переключение исполнения на адрес, отличный от следующего). Такие операции описаны в следующих разделах:

- Отработка условных команд;
- Отработка переходов;
- Отработка циклов;
- Отработка прерываний.

Условные команды начинаются с синтаксической конструкции IF *Предикат.* Для условных команд *Предикат* может быть:

- любым предикатом вычислительного блока, таким как: AEQ, ALT, ALE, MEQ, MLT, MLE, SEQ, SLT, SF0 или SF1;
- любым IALU-предикатом, таким как: JEQ, JLT, JLE, KEQ, KLT, KLE, ISF0 или ISF1;
- любым УУ-предикатом, таким как: BM, LC0E, LC1E, FLAG0_IN, FLAG1_IN, FLAG2_IN или FLAG3_IN;
- любым из перечисленных предикатов, но с отрицанием; предикат

снабжается префиксом N (от NOT).

Полный список предикатов и примеры их использования можно найти в подразделе «Сводка команд устройства управления». Описание этих предикатов можно найти в подразделах «ALU-предикаты», «Предикаты умножителя», «Предикаты сдвигового устройства» и «IALU-предикаты».

Почти все команды процессора могут быть условными. Исключение составляет небольшой набор команд устройства управления. Всегда безусловными являются команды NOP, IDLE, BTBINV, BTBEN, BTBDIS, BTBLOCK, BTBELOCK, TRAP и EMUTRAP.

В устройстве управления имеется два регистра, обеспечивающих управление и индикацию статуса. Устройство управление последовательностью команд контролируется значением регистра SQCTL. После сброса значение регистра равно 0x0000_0004. Подробное описание всех бит регистра приведено в таблице 21.

Таблица 21 – Биты регистра управления SQCTL

Бит	Имя	Назначение
0-1	-	
2	GIE	Глобальное разрешение прерываний: 1 – прерывания разрешены; 0 – запрещены
3	SWIE	Разрешение генерации исключительных ситуаций (программных прерываний): 1 – разрешено; 0 – запрещено
4-6	-	
7	SPSEL	В режиме супервизора, при исполнении прикладной программы (отсутствие обработки прерывания, исключительной ситуации) определяет выбор указателя стека (если режим с раздельными стеками включен): 0 – указатель стека супервизора; 1 – указатель стека пользователя
8	DBGEN	Разрешение отладки: 1 – разрешено; 0 – запрещено
9	NMOD	Режим работы: 0 – пользователь; 1 – супервизор
10	TRCBEN	Включение буфера трасс: 0 – буфер трасс выключен и хранит старое значение; 1 – включен и следит за исполнением переходов
11	TRCBEXEN	Разрешение генерации исключительной ситуации от буфера трасс: 0 – запрещено; 1 – разрешено
12	DF_IEEE	Разрешает использование команд обработки данных с плавающей запятой двойной точности (при установленном в 1 бите EXT_MODE регистра SQSTAT) 0 – используется расширенная точность; 1 – код команды расширенной точности соответствует коду команды двойной точности

Бит	Имя	Назначение
13	BYTE_ON	Разрешение использование кодов команды TRAP 32-63 в качестве кодов префикса для создания новых команд (при установленном в 1 бите EXT_MODE регистра SQSTAT). 0 – команды TRAP 32-63 используются для генерации исключений; 1 – команды TRAP 32-63 используются в качестве префикса
14	USP	Разрешение использования раздельных указателей стеков (регистр JK27) (при установленном в 1 бите EXT_MODE регистра SQSTAT). 0 – регистр JK27 один для пользователя и супервизора; 1 – для пользователя и супервизора используются разные регистры
31-15	-	Зарезервированы и не используются. При чтении всегда равны нулю

Биты 12, 13, 14 регистра управления соответствуют дополнительным возможностям процессора, и их включение должно быть дополнительно разрешено.

Регистр SQCTLST является псевдонимом для SQCTL. При записи по данному адресу:

- ‘1’ в любом бите записываемых данных устанавливает соответствующий бит в SQCTL;
- ‘0’ в записываемых данных не меняет значение бита.

Регистр SQCTLCL является псевдонимом для записи в SQCTL. При записи по данному адресу:

- ‘0’ в любом бите записываемых данных сбрасывает соответствующий бит в SQCTL;
- ‘1’ в записываемых данных не меняет значение бита.

Регистр статуса SQSTAT доступен только для чтения и содержит информацию о текущем статусе устройства управления. Подробное описание разрядов регистра приведено в таблице 22.

Таблица 22 – Биты регистра SQSTAT

Бит	Имя	Назначение
1:0	MODE	Текущий режим работы процессора: 00 – пользователь 01 – супервизор 11 – эмулятор Другие состояния невозможны
2	IDLE	Процессор в состоянии ожидания прерывания (если 1)
7:3	SPVCMD	Значение параметра последней выполненной команды TRAP
11:8	EXCAUSE	Код последней исключительной ситуации: 0000 – команда TRAP 0001 – точка наблюдения или буфер трасс 0010 – команда плавающей запятой 0011 – ошибочная линия команд 0100 – невыровненный доступ 0101 – попытка записи защищенного регистра 0110 – переполнения счетчика статистики 0111 – ошибочный доступ в IALU

Бит	Имя	Назначение
		1000 – x 1001 – x 1010 – x 1011 – x 1100 – x 1101 – x 1110 – x 1111 – не было ошибок с момента сброса
15:12	EMCAUSE	Код ситуации, вызвавшей переход в режим эмулятора: 1111 – не было перехода в режим эмулятора после сброса 0000 – команда EMUTRAP 0001 – запрос JTAG 0010 – сработала точка наблюдения Все другие коды зарезервированы и не могут присутствовать
16	FLAG0	Состояние внешнего контакта FLAG0
17	FLAG1	Состояние внешнего контакта FLAG1
18	FLAG2	Состояние внешнего контакта FLAG2
19	FLAG3	Состояние внешнего контакта FLAG3
20	EXE_ISR	Флаг устанавливается в 1 при: - переходе процессора в состояние обработки прерывания, - записи данных в регистр RETIB. Установленный флаг запрещает прерывания. Флаг сбрасывается: - командой выхода из прерывания RTI, - командой RDS (если нет обработки исключительной ситуации), - чтением регистра RETIB. Сброшенный флаг означает возможность обработки нового прерывания процессором
21	EXE_SWI	Признак того что процессор находится в состоянии обработки исключительной ситуации (когда равен 1). Флаг устанавливается в 1 при: - переходе процессора в состояние исключительной ситуации. Флаг сбрасывается: - командой выхода из прерывания RTI, - командой RDS. Установленный в 1 флаг означает что процессор работает в режиме супервизора и использует указатель стека супервизора. Если флаг сброшен, режим работы определяется битом NMOD
22	EMUL	Признак того что процессор находится в режиме эмулятора (когда равен 1)
23	ISR_mode	Признак режима обработки прерывания. Флаг устанавливается в 1 при: - переходе процессора в состояние обработки прерывания. Флаг сбрасывается: - командой выхода из прерывания RTI, - командой RDS (если нет обработки исключительной ситуации),

Бит	Имя	Назначение
		Установленный в 1 флаг означает что процессор работает в режиме супервизора и использует указатель стека супервизора. Если флаг сброшен, режим работы определяется битом NMOD. Флаг работает только если бит EXT_MODE установлен
24	BTBEN	Дублирование бита 28
25	-	
26	-	
27	-	
28	BTBEN	Буфер предсказания переходов включен (если 1), либо выключен (если 0)
29	-	
30	EXT_MODE	Режим работы с дополнительными функциями: 1 – разрешено использование дополнительных функций процессора 0 – запрещено По сбросу значение 0. Всякий раз при выполнении команды 0x8B5E1A3C меняет свое значение на инверсное
31	I_LOCK	Флаг блокировки прерываний (1 – прерывания блокированы)

Регистр управления и регистр статуса устройства управления поддерживают:

- **Нормальный режим.** Используя бит нормального режима (NMOD), можно программным образом выбрать либо пользовательский (=0) режим исполнения, при котором программа может получить доступ только к регистрам вычислительного блока и IALU, либо режим супервизора (=1), при котором доступ программы к регистрам неограничен. После загрузки или реагируя на прерывание, процессор автоматически переходит в режим супервизора.
- **Глобальное разрешение прерываний.** Используя бит глобального разрешения прерываний(GIE), можно программным образом глобально маскировать аппаратные прерывания.
- **Маскирование исключений.** Используя бит SWIE в регистре SQCTL можно программным образом маскировать программные прерывания (исключения).
- **Значение дискретных сигналов.** Используя биты значений входных дискретных сигналов (FLGx), программа может отслеживать значения на входных линиях дискретных сигналов. Задавать направление передачи дискретных сигналов по внешним линиям (вход/выход) и устанавливать выходные значения можно с помощью регистра flagreg.
- **Управление буфером целевых адресов перехода (BTB).** Используя бит включения (BTBEN) BTB, можно программными средствами управлять функционированием BTB.

Доступна также и другая информация о состоянии.

8.1.1 *Отработка условных команд*

Все команды (за исключением NOP, IDLE, BTBINV, BTBEN, BTBDIS, BTBLOCK, BTBELOCK, TRAP и EMUTRAP) процессора могут исполняться условно (механизм, известный также, как предикативное исполнение). Поле предиката помещается в одном из командных полей (в первом) командной строки, и все остальные команды

строки либо исполняются, либо нет, в зависимости от результата вычисления предиката, либо исполняются безусловно.

В условных командах вычислительного блока или IALU (IF ... DO) исполнение каждого из командных полей командной строки, следующих за предикатом, может индивидуально зависеть от значения предиката. Условная вычислительная команда имеет вид:

IF Condition;
DO, Instruction; DO, Instruction; DO, Instruction;;

Такой синтаксис позволяет подчинить одному предикату (Condition) до трёх условных команд (instruction) в полях командной строки. Пропуск DO перед командным полем делает команду в этом поле безусловной.

В Листинге 31 приведено несколько примеров условных команд ALU. Описание ALU-предикатов, использованных в этих примерах, можно найти в подразделе «ALU-предикаты».

Листинг 31 – Условные команды вычислительного блока и IALU

```
IF XALT; DO, R3 = R1 + R2 ;;
/* условное исполнение сложения в вычислительных блоках X и Y определяется
значением предиката ALT вычислительного блока X */
IF YAEQ; DO, XR0= R1 + R2 ;;
/* условное исполнение сложения в вычислительном блоке X определяется
значением предиката AEQ вычислительного блока Y */
IF ALE; DO, R0 = R1 + R2 ;;
/* условное исполнение сложения в вычислительных блоках X и Y определяется
значением предиката ALE вычислительного блока X для исполнения в
вычислительном блоке X и значением предиката ALE вычислительного блока Y для
исполнения в вычислительном блоке Y */
IF JLE; DO, XR0 = R1 + R2 ;;
/* условное  исполнение сложения в вычислительном блоке Хопределяется
значением предиката JLE в J-IALU */
IF ALE; DO, R0 = [J0 + J1] ;;
/* условное исполнение загрузки определяется значением предикатов ALE
вычислительных блоков X и Y, объединенных логическим ИЛИ */
```

В условной команде устройства управления (if ... else) исполнение команды УУ из первого поля командной строки и каждой команды из следующих за ним полей зависит от значения помещенного в начале строки предиката. Такие условные команды устройства управления имеют вид:

IF Condition, Sequencer_Instruction;

ELSE, Instruction; ELSE, Instruction; ELSE, Instruction ;;

Такой синтаксис позволяет подчинить одному предикату (Condition) до трёх условных команд (Instruction) в полях командной строки, следующих за полем команды устройства управления. Пропуск else перед командным полем делает команду в этом поле безусловной.

Такой синтаксис позволяет подчинять команды УУ предикатам вычислительных блоков. В листинге 32 приведено несколько примеров условных команд устройства управления, подчиненных ALU-предикатам.

Листинг 32 – Условные команды устройства управления

```
IF ALE, JUMP label;;
/* условное исполнение перехода определяется значением предикатов ALE
вычислительных блоков X и Y, объединенных логическим ИЛИ */
```

```
IF XALE, JUMP label;;
/* условное исполнение перехода определяется значением предиката ALE
вычислительного блока X */

IF AEQ, JUMP label; ELSE, XR0 = R5 + R6; YR8 = R9 - R10;;
/* условное исполнение перехода определяется значением предикатов ALE
вычислительных блоков X и Y, объединенных логическим ИЛИ; сложение в
вычислительном блоке X выполняется только тогда, когда не выполняется переход (и
XAEQ есть «ложь»); вычитание в вычислительном блоке Y безусловно (т.е.
выполняется всегда) */

If MEQ, CJMP; ELSE R4 = R6 + R7;;
/* условное исполнение перехода определяется значением предикатов MEQ
вычислительных блоков X и Y, объединенных логическим ИЛИ; команда сложения
исполняется в каждом вычислительном блоке в зависимости от значения
соответствующего этому блоку флага - X-сложение выполняется, если не установлен
флаг XMEQ и Y-сложение выполняется, если не установлен флаг YMEQ */
```

Опция NF (не обновлять флаги) для условных команд (IF. . .DO и IF. . .ELSE) имеет ограничения по использованию: либо все командные поля в условной командной строке генерируют флаг состояния, либо все командные поля используют опцию (NF), чтобы предотвратить генерацию флагов состояния. Команды, к которым опция (NF) неприменима, свободны от этого ограничения. Например:

```
IF ALE;
DO, XR0 = R1 + R2 (NF) ;
DO, XR5 = LSHIFT R4 BY -4 (NF) ;
DO, J3 = J5 + J29 (NF) ;;
/* все поля в этой командной строке используют опцию (NF) */

IF AEQ, JUMP my_label ;
ELSE, J1 = J0 + J2 (NF) ;
ELSE, XR0 = R1 - R3(NF) ;
ELSE, YR0 = R5 * R6(NF) ;;
/* все поля в этой командной строке используют опцию (NF) */
IF AEQ, JUMP my_other_label;
ELSE, COMP(J3,J4)  ;
ELSE, XR0 = R1 - R3(NF) ;
ELSE, YR0 = R5 * R6(NF) ;;
/* COMP не поддерживает опцию (NF), но может присутствовать в командной строке с
NF */
```

Помимо того ограничения, что любая команда устройства управления должна использовать первое командное поле (каждая командная строка содержит до четырех полей), важно иметь в виду, что адрес, использованный в команде перехода (например, JUMP Address), определяет, будет ли команда занимать одно поле или два. Если команда устройства управления задает относительный или абсолютный адрес, больший по размеру, чем 16 бит, процессор автоматически использует непосредственное расширение (второе поле команды), чтобы разместить избыточные биты адреса.

Когда программная метка Label используется в качестве адресной ссылки, ассемблер превращает Label в PC-относительный адрес. Когда метка Label содержится в той же программной секции .section, что и команда перехода,

ссылающаяся на нее, ассемблер использует 15-битный адрес, если это возможно. Когда метка Label не содержится в той же программной секции .section, что и команда перехода, ассемблер использует 32-битный адрес. Дополнительную информацию о PC-относительных и абсолютных адресах и переходах см. в подразделе «Отработка переходов».

Для обеспечения поддержки условных команд для статических предикатов в устройстве управления имеется регистр SFREG статических флагов (Таблица 23). УУ использует этот регистр для хранения значений статических флагов вычислительных блоков и IALU для отложенного использования в условных командах.

Таблица 23 – Биты регистра статических флагов.

Бит	Имя	Назначение
0	GSCF0	Статический флаг 0 целочисленных АЛУ
1	GSCF1	Статический флаг 1 целочисленных АЛУ
2	XSCF0	Статический флаг 0 вычислительного модуля X
3	XSCF1	Статический флаг 1 вычислительного модуля X
4	YSCF0	Статический флаг 0 вычислительного модуля У
5	YSCF1	Статический флаг 1 вычислительного модуля У
31-6	-	Не используются и всегда равны нулю

8.1.2 *Отработка переходов*

Устройство управления поддерживает отработку переходов по командам JUMP, CALL, CJMP_CALL, CJMP, RETI, RDS и RTI. Из рисунка 121 можно получить представление об основных различиях между всеми вариантами переходов. Ниже отражены дополнительные подробности, касающиеся операций JUMP, CALL и CJMP_CALL.

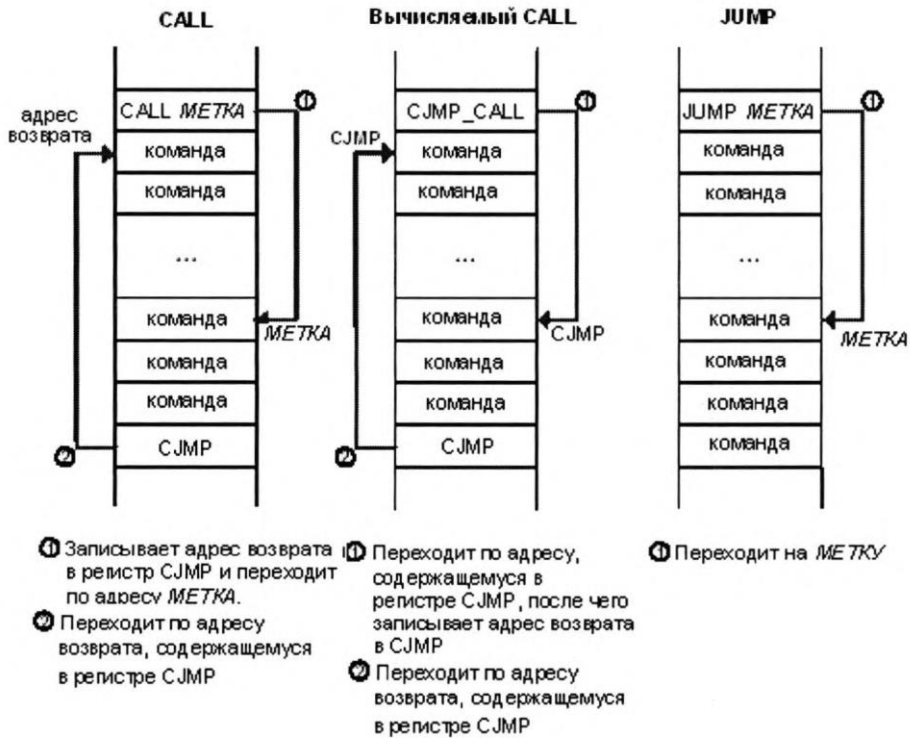


Рисунок 121 – Сравнение особенностей команд CALL, вычисляемый CALL и JUMP

Команда вызова CALL переводит исполнение на адрес *Метка*. При обработке вызова устройство управления записывает адрес возврата (следующий последовательный адрес за командой вызова) в регистр CJMP, а затем переходит по адресу подпрограммы. Команда CJMP в конце подпрограммы принуждает устройство управления перейти по адресу, содержащемуся в регистре CJMP.

Команда CJMP_CALL переводит исполнение на подпрограмму, используя адрес вычисляемого перехода (из регистра CJMP). Одним из способов загрузки вычисляемого адреса является использование опции CJMP в команде сложения/вычитания IALU. CJMP_CALL переводит исполнение на адрес, указанный в регистре CJMP, а затем в CJMP загружает адрес возврата. Команда CJMP в конце подпрограммы принуждает устройство управления перейти по адресу, содержащемуся в регистре CJMP.

Команда перехода JUMP переводит исполнение на адрес *Метка*.

Команды перехода устройства управления используют PC-относительные или абсолютные адреса. Для относительной адресации устройство управления базирует адрес перехода по 32-битному программному счетчику (Program Counter – PC). Для абсолютной адресации команды перехода предоставляют непосредственное значение 32-битного адреса. Программный счетчик допускает линейную адресацию во всем 32-битном диапазоне.

Командами CALL или JUMP по умолчанию подразумевается, что используемый ими адрес – *PC-относителъный.* Чтобы использовать *абсолютный адрес,* в командах перехода должна быть указана опция (abs). В следующем примере показаны команда перехода по PC-относительному адресу и команда перехода по абсолютному адресу.

```
JUMP fft_routine ;;
/* fft_routine – это программная метка, присутствующая внутри данной секции (.section); ассемблер конвертирует метку в 15-битный PC-относительный адрес (если это возможно) */
JUMP fft_routine (ABS) ;;
/* fft_routine – это программная метка, внешняя по отношению к данной секции (.section); ассемблер конвертирует метку в абсолютный адрес (поскольку использована опция ABS); эта метка конвертируется в 32-битный адрес также и потому, что лежит вне данной секции */
```

Еще одним предположением по умолчанию, применимым ко всем условным командам перехода, является то, что значение предиката есть **true** и переход должен быть предпринят, так называемый *прогнозируемый переход.* Когда по каким-то своим соображениям программист может предполагать, что в большинстве случаев переход не предпринимается, он должен снабдить такой переход опцией «непрогнозируемый» (NP). Такой переход называется непрогнозируемым. В следующем примере первый из условных переходов является *прогнозируемым переходом*, а второй – *непрогнозируемым переходом:*

```
IF AEQ, JUMP fft_routine ;;
/* эта условная команда является прогнозируемым переходом */
IF MEQ, CALL iir_routine (ABS) (NP) ;;
/* эта условная команда является непрогнозируемым переходом */
```

Правильное прогнозирование перехода как предпринимаемого или как не предпринимаемого чрезвычайно важно для производительности конвейера. УУ записывает информацию о каждом прогнозируемом переходе в BTB. BTB осуществляет проверку потока адресов на стадии конвейера IA. Если происходит попадание в BTB (т е. BTB опознает адрес команды, которая вызвала переход при предыдущем проходе программного кода), то BTB подставляет соответствующий

целевой адрес в качестве адреса выборки следующей команды. Если переход кэширован и правильно спрогнозирован, то потери производительности, вызванные ветвлением, снижаются с 4 – 8 тактов до 0 тактов. Дополнительную информацию см. в подразделе «Буфер целевых адресов перехода (BTB)».

8.1.3 *Отработка циклов*

Устройство управления поддерживает отработку циклов с минимальными накладными расходами. Как показано на рисунке 119 цикл позволяет программе исполнить группу команд многократно, пока не окажется выполненным определенное условие. После того, как это условие будет выполнено, исполнение продолжится с команды, следующей непосредственно за циклом.

Для организации цикла программа использует регистр счетчика цикла, команду декрементирования счетчика, а также условную команду перехода, которая осуществляет проверку, не выполнено ли условие окончания цикла и, в случае если оно не выполнено, выполняет переход на начало цикла.

Для организации циклов УУ располагает двумя выделенными счетчиками циклов и специальными предикатами окончания цикла для них, позволяющими выполнить условный переход на конец цикла:

- «отсчет не закончен»: IF NLC0E и IF NLC1E;
- «отсчет закончен»: IF LC0E и IF LC1E).

Кроме того, УУ автоматически декрементирует счетчик при исполнении специальной проверки счетчика цикла и условного перехода. См., к примеру, цикл:

```
LC0 = 5;;
Rep_loop:
If nlc0e, jump Rep_loop; k0 = k0 + 1;;
```

В данном случае мы имеем цикл инкремента регистра K0, повторяющийся пять раз. В идеальном случае нам хотелось бы, чтобы этот цикл выполнился за пять тактов. Однако, этого не происходит из-за наличия конвейера. Процессор потратит дополнительные такты на выполнение первой итерации (вход в цикл) и на выполнение последней итерации (выход из цикла). Поскольку тело цикла будет содержать две команды, также имеет значение выровнен ли адрес метки Rep_loop на границе квадрослова (в данном случае точнее двойного слова). В случае если две команды тела цикла лежат в разных квадрословах, каждая итерация будет выполняться за два такта. Если в одном квадрослове – один такт. Теперь рассмотрим вход в цикл. Когда команда перехода jump Rep_loop попадает на стадию дешифрации D, выполняется переход, т.к. данная команда имеет опцию предсказания перехода. Далее команда переходит на стадию исполнения X, и здесь она притормаживается на три такта, т.к. счетчик LC0 еще не загрузился (команда LC0 = 5 в этот момент на стадии A). Однако, это не влияет на быстродействие, т.к. переход уже запущен, и пока команда перехода ждет загрузки регистра LC0, конвейер загрузки команд выполняет подзагрузку команды по адресу перехода. Таким образом, вход в цикл (исполнение первой итерации) займет пять тактов, в течение которых процессор запомнит адрес перехода в BTB, выполнит переход, и первая команда цикла достигнет стадии исполнения X. Последующие итерации, благодаря устройству предсказания ветвлений BTB, будут выполняться за один такт. Выход из цикла (достижение счетчиком цикла значения 0) потребует несколько тактов. Для последнего цикла процессор спрогнозирует переход (BTB не отслеживает значения выделенных счетчиков), но он окажется ложным, и после вычитания из счетчика (когда результат вычитания равен 0) процессор перезапустит конвейер выборки команд по адресу, следующему за циклом. Из-за этого будет потеряно пять тактов. Таким образом, общее число тактов исполнения цикла будет равно десяти плюс количество тактов для исполнения команд тела цикла.

Проверка значений предикатов окончания цикла «отсчет закончен» и «отсчет не закончен» влияет на значения счетчиков циклов. Каждая ссылка на NLCOE, NLC1E, LC0E или LC1E декрементирует соответствующий счетчик цикла и вычисленное значение предиката есть **true**, когда счетчик обнуляется. Счетчики сначала декрементируются и затем проверяются.

Удобство организации циклов с помощью LC0 и LC1 состоит в отсутствии накладных расходов на вычитание счетчика цикла. Однако, можно организовывать циклы, используя регистр IALU в качестве счетчика цикла. В этом случае необходимо включать в цикл команду декрементирования счетчика, и использовать предикат для проверки результата операции декрементирования в условном переходе на конец цикла. Пример, аналогичный предыдущему, приведен ниже:

```
J0 = 5;;
.align 4;
Rep_loop:
J0 = J0 – 1;;
If njeq, jump Rep_loop; k0 = k0 + 1;;
```

Из-за необходимости добавления декремента регистра J0 в теле цикла появляется дополнительный такт. Это уменьшает производительность. Однако, в случаях, когда тело цикла состоит не из одной линии команд и в некоторой из линий команд есть свободный слот, и в него можно поместить команду декремента J0, то в этом случае с помощью регистров IALU можно организовывать циклы, не уступающие по производительности циклам с LC.

На циклах с использованием регистров IALU программы достигают лучшей производительности, чем в случае если бы в качестве счетчика использовался регистр вычислительного блока. Причина такой разницы в производительности кроется в различии исполнения команд IALU и команд вычислительных блоков в конвейере команд. Хотя, в случаях, когда тело цикла требует большого числа тактов для своего исполнения, разница может быть и не существенной. Дополнительную информацию см. в подразделе «Операции конвейера команд».

Поскольку проверка окончания цикла приводит к повторяющимся переходам на метку (адрес) начала цикла, выравнивание адреса начала цикла влияет на производительность цикла. Если командная строка, на которую осуществляется переход, пересекает границу квадрослова, то вставляется такт приостановки. Во избежание такой приостановки метка начала цикла должна быть выровнена на границу квадрослова, как это показано в примере выше. Такое выравнивание гарантирует, что граница квадрослова не будет пересекаться при переходе на очередную итерацию цикла. Отметим, что такое выравнивание нужно только, если линия команд начала цикла содержит больше одной команды, и эти команды попадают в разные квадрослова, т.е. в предыдущем примере выравнивание не дает никакого эффекта.

8.1.4 *Отработка прерываний*

Прерывания – это события, которые вынуждают ядро приостановить текущий процесс, осуществить переход и запустить другой процесс. Эти события могут происходить в любое время и могут быть как внутренними, так и внешними по отношению к процессору. Прерывания предназначены для:

- синхронизации операций ядра и не-ядра;
- обнаружения ошибок;
- средств отладки;
- управления приложениями.

Подробно описание контроллера прерываний, его регистров, действий процессора по обработке прерывания приведено в Спецификации процессора. Здесь приведено описание ресурсов процессора, используемых для обработки прерываний, т.к. работа устройства управления тесным образом связана с работой контроллера прерываний.

За каждым прерыванием закреплен векторный регистр в таблице векторов прерываний (IVT – Interrupt Vector Table) и бит в регистрах флагов и масок прерываний (ILAT, IMASK и PMASK). Векторный регистр содержит определяемый пользователем адрес процедуры, обслуживающей прерывание. Четыре прерывания представляют собой прерывания общего назначения от внешних линий запроса (IRQ3-0) и одно прерывание представляет собой векторное прерывание общего назначения (VIRPT). Остальные прерывания являются специальными.

В регистре IMASK значение бита равное “1” означает, что прерывание размаскировано (процессор распознает и обслуживает прерывание). Значение “0” соответствует замаскированному прерыванию (процессор не распознает прерывание). Биты с одинаковыми номерами в регистрах ILAT, IMASK и PMASK закреплены за одними и теми же прерываниями.

Таблица 24 – Биты регистров ILATL, IMASKL, PMASKL

Бит	Имя	Описание
0	INT_KERNEL	Прерывание ядра VDK
1	-	-
2	INT_TIMER0LP	Низкий приоритет таймера #0
3	INT_TIMER1LP	Низкий приоритет таймера #1
5:4	-	-
6	INT_LINK0	порт связи #0
7	INT_LINK1	порт связи #1
8	INT_LINK2	порт связи #2
9	INT_LINK3	порт связи #3
13:10	-	-
14	INT_DMA0	Регистр DMA #0
15	INT_DMA1	Регистр DMA #1
16	INT_DMA2	Регистр DMA #2
17	INT_DMA3	Регистр DMA #3
21:18	-	-
22	INT_DMA4	Регистр DMA #4
23	INT_DMA5	Регистр DMA #5
24	INT_DMA6	Регистр DMA #6
25	INT_DMA7	Регистр DMA #7
28:26	-	-
29	INT_DMA8	Регистр DMA #8
30	INT_DMA9	Регистр DMA #9
31	INT_DMA10	Регистр DMA #10

Таблица 25 – Биты регистров ILATH, IMASKH, PMASKH

Бит	Имя	Описание
0	INT_DMA11	Регистр DMA #11
4:1	-	-
5	INT_DMA12	Регистр DMA #12
6	INT_DMA13	Регистр DMA #13

Бит	Имя	Описание
8:7	-	-
9	INT_IRQ0	Вывод регистра IRQ0
10	INT_IRQ1	Вывод регистра IRQ1
11	INT_IRQ2	Вывод регистра IRQ2
12	INT_IRQ3	Вывод регистра IRQ3
15:13	-	-
16	INT_VIRPT	Векторный регистр VIRPT
17	-	-
18	INT_BUSLOCK	Вектор блокировки шины
19	-	-
20	INT_TIMER0HP	Высокий приоритет таймера 0
21	INT_TIMER1HP	Высокий приоритет таймера 1
24:22	-	-
25	INT_HWERR	Аппаратная ошибка
31:26	-	-

УУ манипулирует прерываниями, используя регистры управления ILAT, IMASK и PMASK. Регистры управления ILAT, IMASK и PMASK используются следующим образом:

- ILAT – регистр защелкивания прерываний – защелкивает прерывания, чувствительные к фронту (бит прерывания = 1), и отображает активные запросы прерываний, чувствительных к уровню (бит прерывания = 1);
- IMASK – регистр масок прерываний – маскирует каждое прерывание индивидуально; все прерывания могут быть также замаскированы одновременно путем использования бита GIE в регистре sqctl;
- PMASK – регистр указателей масок прерываний – используется для индикации прерываний, которые в данный момент обслуживаются (бит прерывания = 1).

Приоритет прерываний от высокого к низкому соответствует порядку их появления в регистрах ILAT, IMASK и PMASK. Дополнительную информацию о чувствительности прерываний, защелкивании, IVT и связанных вопросах можно найти в Спецификации процессора.

Среди прерываний от линий запроса различают прерывания, чувствительные к фронту и чувствительные к уровню сигнала. Прерывания, чувствительные к фронту, защелкиваются на момент своего запроса и остаются защелкнутыми до тех пор, пока они не будут обслужены или сброшены командой. Прерывания, чувствительные к уровню, отличаются от них тем, что, прерывание, не обслуженное до момента снятия запроса, окажется потерянным, а также тем, что сохраняющийся после обслуживания прерывания запрос рассматривается как новое прерывание.

В устройстве управления имеется два регистра, обеспечивающие управление прерываниями, а также линиями дискретных сигналов и таймеров.

Регистры управления дискретными сигналами (FLAGREG) и прерываниями (INTCTL) описаны в таблицах 26 и 27 соответственно и обеспечивают:

- **управление чувствительностью прерываний**. Используя бит прерываний-по-фронту (irq_edge), можно программно указать, какой должна быть чувствительность прерываний: к фронту или к уровню для каждой линии запроса IRQ3-0 независимо;
- **управление и опрос статуса дискретных сигналов**. Используя бит направления дискретного сигнала (flagx_en), биты установки выходного значения (FLAGx_OUT) и значения входного дискретного сигнала (FLGx), можно программно указать является ли линия дискретного сигнала

(FLAG3-0) входной или выходной. Если выходной, то следует установить выходное значение 1 или 0; если входной, то можно наблюдать за входным значением;

- **управление таймерами.** Используя биты запуска таймеров (TMR0RN и TMR1RN), можно программным образом независимо включать два таймера.

Таблица 26 – Биты регистра управления флагом

Бит	Имя	Назначение
3:0	FLAGx_EN	Направление флага. 0 – прием 1 – выдача
7:4	FLAGx_OUT	Значение флага при выдаче
31-8	-	Не используются и всегда равны нулю

Регистр управления флагами относится к группе регистров УУ и тактируется клоком ядра. Выходы регистра непосредственно управляют внешними контактами микросхемы. Этот фактор нужно учитывать при формировании сигналов определенной длительности.

Таблица 27 – Регистр INTCTL

Бит	Имя	Назначение
0	IRQ0_EDGE	Запрос прерывания по входу IRQ0 вызывается: 0 – фронтом сигнала (из высокого в низкий) 1 – уровнем (низкий)
1	IRQ1_EDGE	Запрос прерывания по входу IRQ1 вызывается: 0 – фронтом сигнала (из высокого в низкий) 1 – уровнем (низкий)
2	IRQ2_EDGE	Запрос прерывания по входу IRQ2 вызывается: 0 – фронтом сигнала (из высокого в низкий) 1 – уровнем (низкий)
3	IRQ3_EDGE	Запрос прерывания по входу IRQ3 вызывается: 0 – фронтом сигнала (из высокого в низкий) 1 – уровнем (низкий)
4	TMR0RN	Разрешение работы таймера 0: 0 – таймер остановлен 1 – таймер работает
5	TMR1RN	Разрешение работы таймера 1: 0 – таймер остановлен 1 – таймер работает
31:6	-	Не используются и при чтении всегда равны нулю

На рисунке 122 изображены два случая обработки прерывания. Первый случай для не повторно-входимых прерываний и второй для повторно-входимых. Разница между двумя типами обработки заключается в том, что не повторно- входимая процедура не может заново защелкнуть такое же прерывание до завершения обработки, в то время как повторно-входимая процедура обработки (будучи пониженной в статусе до статуса подпрограммы командой RDS) может заново защелкнуть такое же прерывание, пока обработка все еще продолжается.

Чтобы лучше понять разницу, проследим шаги обработки прерывания для обоих типов обслуживания.

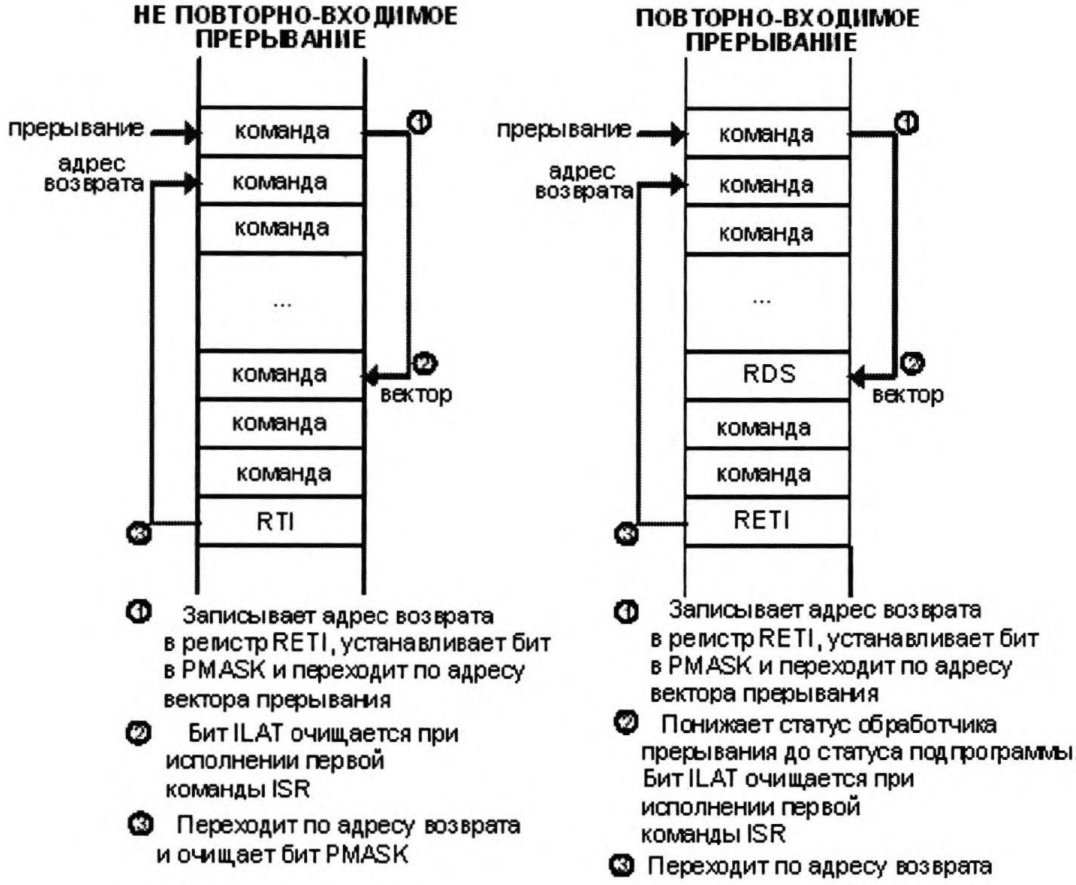


Рисунок 122 – Сравнение повторно-входимых и не повторно-входимых процедур обслуживания прерывания

Для не повторно-входимой процедуры обслуживания шаги таковы:

- УУ распознает прерывание, когда бит прерывания защелкнут в регистре ILAT, прерывание размаскировано соответствующими битами в регистрах IMASK и PMASK и, когда при этом разрешены глобальные прерывания битом GIE в регистре SQCTL.
- УУ переводит процессор в режим супервизора, осуществляет переход по адресу, установленному в векторном регистре, загружает адрес возврата (следующий адрес за последней командной строкой, выполненной до прерывания) в регистр RETI и, наконец, устанавливает бит, соответствующий прерыванию в регистре PMASK с тем, чтобы заблокировать повторное использование прерывания.
- Процессор исполняет процедуру обработки прерывания. Исполнение первой команды обработчика прерывания сбрасывает бит прерывания в регистре ILAT.
- УУ по достижению команды RTI в конце процедуры обработки прерывания осуществляет переход по адресу возврата и сбрасывает бит прерывания в регистре PMASK.

Для повторно-входимой процедуры обслуживания шаги таковы:

- УУ распознает прерывание, когда бит прерывания защелкнут в регистре ILAT, прерывание размаскировано соответствующими битами в регистрах

IMASK и PMASK и, когда при этом разрешены глобальные прерывания битом GIE в регистре SQCTL.

- УУ переводит процессор в режим супервизора, осуществляет переход по адресу, установленному в векторном регистре, загружает адрес возврата (следующий адрес за последней командной строкой, выполненной до прерывания) в регистр RETI и, наконец, устанавливает бит, соответствующий прерыванию в регистре PMASK. Программа должна сохранить содержимое регистра RETI в стеке, прежде, чем будет выполнен следующий шаг.
- УУ приводит статус обработчика прерывания к статусу подпрограммы по достижению команды RDS в начале процедуры обработки прерывания и сбрасывает бит прерывания в регистре PMASK. Исполнение первой команды обработчика прерывания сбрасывает бит прерывания в регистре ILAT.
- Поскольку биты прерывания в регистрах ILAT и PMASK сброшены, прерывание может быть защелкнуто заново до того, как процедура обработки прерывания будет завершена.
- Процессор исполняет процедуру обработки прерывания.
- УУ по достижению команды RETI в конце процедуры обработки прерывания осуществляет переход по адресу возврата.

8.2 Операции конвейера команд

Конвейер процессора содержит девять ступеней (стадий). Для достижения лучшего понимания механизма прохождения потока команд по конвейеру и времени, необходимого для каждой стадии, в данном разделе используется набор диаграмм в виде таблиц, подобных таблице 28.

Стадии конвейера команд изображаются вертикальными столбцами, а такты CCLK (тактовой частоты ядра процессора) изображаются горизонтальными строками. Как правило, каждая стадия конвейера требует одного такта на обработку командной строки, содержащей до четырех команд. Строка(линия) команд, исполняющаяся на определенной стадии конвейера отмечается номером в соответствующей клетке таблицы. При этом значение #2 соответствует линии команд с номером 2. Конвейер выборки команд оперирует не с линиями команд, а с квадрословами команд. Будем обозначать квадрослова команд как $x, что означает выборку квадрослова по адресу x. В примере таблицы 28 первая команда #1 расположена по адресу 0.

Таблица 28 – Последовательность исполнения программы 1

Код программы
#1: XR0 = R1 + R2 ; NOP; NOP; NOP;; /* выч. блок X */
#2: YR3 = R4 * R5 ; NOP; NOP; NOP;; /* выч. блок Y */
#3: J0 = J1 + J2 ; NOP; NOP; NOP;; /*J-IALU */
#4: K0 = K1 - K2 ; NOP; NOP;NOP;; /* K-IALU */

Такт	IA	IM	IW	P	D	X	A	M	W
1	$0								
2	$4	$0							
3	$8	$4	$0						
4	$12	$8	$4	#1					
5		$12	$8	#2	#1				
6			$12	#3	#2	#1			
7				#4	#3	#2	#1		

8					#4	#3	#2	#1	
9						#4	#3	#2	#1
10							#4	#3	#2
11								#4	#3
12									#4

В таблице 29 приведен пример конвейера команд для случая, когда в одном квадрослове команд помещается несколько линий команд. В данном случае мы имеем одно квадрослово команд, включающее четыре линии команд #1 – #4. Адрес первой команды равен 0. Чтение других квадрослов не показано.

Таблица 29 – Последовательность исполнения программы 2

#1: XR0 = R1 + R2 ;;
#2: YR3 = R4 * R5 ;;
#3: J0 = J1 + J2 ;;
#4: K0 = K1 - K2 ;;

Такт	**IA**	**IM**	**IW**	**P**	**D**	**X**	**A**	**M**	**W**
1	$0								
2		$0							
3			$0						
4				#1					
5				#2	#1				
6				#3	#2	#1			
7				#4	#3	#2	#1		
8					#4	#3	#2	#1	
9						#4	#3	#2	#1
10							#4	#3	#2
11								#4	#3
12									#4

Первые три стадии конвейера команд (IA, IM и IW) называются *конвейером выборки (чтения) команд.* Такты выборки представляют собой первый конвейер и привязаны к доступу к памяти. Функционирование этого конвейера управляется памятью, а не командами. Инициирование чтения нового квадрослова команд зависит от состояния буфера IAB. IAB представляет собой FIFO глубиной пять квадрослов и инициирует выборку команд всякий раз, когда имеется свободное место в буфере. При этом также имеет значение, сколько запросов чтения находится на конвейере выборки команд. Например, если в IAB имеется два свободных входа, а на стадиях конвейера IM и IW имеются активные запросы, буфер не формирует нового запроса чтения (стадия IA будет неактивна). Это связано с тем, что конвейер обработки команд может остановиться, но конвейер выборки команд продолжит движение, и буфер должен иметь место, чтобы принять два прочитанных квадрослова команд.

Когда выборка осуществляется из внешней памяти, поток аналогичен описанному, хотя и значительно медленней. Темп выборки будет зависеть от конфигурации внешней памяти, частоты внешней шины и др.

Следующие шесть стадий конвейера команд (P, D, X, A, M и W) называются *конвейером обработки команд.* Стадия P (Predecode – предварительное декодирование) является первой стадией этого конвейера. В такте Predecode из IAB извлекается следующая полная командная строка, и команды этой строки распределяются по различным исполнительным устройствам.

Исполнительные устройства включают в себя:

- J-IALU или K-IALU – целочисленные команды, загрузка/сохранение и пересылка регистров;
- вычислительный блок X и Y, либо оба одновременно – команды обработки данных;
- устройство управления – переходы, условные команды и др.

Если конвейер выборки команд имеет одну линию выборки, то конвейер обработки команд имеет девять линий, т.е. теоретически возможна обработка до 9-ти команд в одном такте. На практике реально достижимое число одновременно обрабатываемых команд не больше шести (предполагается, что одна и та же команда на двух линиях конвейера – это две команды). Представление конвейера обработки в виде 9-ти независимых линий обработки позволяет понять какие ограничения накладываются на возможные комбинации команд в одной линии. Также анализируя отдельную линию конвейера обработки и тип команды, которая находится на конвейере, можно сказать какие операции выполняются на каждой стадии конвейера. Из рисунка 121 видно, что модули IALU имеют по одной линии конвейера, вычислительные модули две линии в каждом модуле. Устройство управления имеет три линии конвейера. Приведенные обозначения стадий обработки D, X, A, M, W соответствуют обработке команды загрузки данных на линии конвейера IALU и обозначают:

- D декодирование команды;
- X вычисление адреса доступа;
- A передача адреса в подсистему памяти (декодирование и арбитраж);
- M чтение накопителя памяти;
- W запись данных в регистр назначения.

Обозначим через N выполняемые действия на какой-то стадии конвейера, которые сводятся лишь к передаче входной информации стадии конвейера на следующую стадию без изменения. Тогда конвейеры для различных типов команд можно описать так, как это приведено в таблице 30.

Таблица 30 – Стадии конвейера различных типов команд

Тип команды	**Линия конвейера**	**Стадии конвейера**				
		1	**2**	**3**	**4**	**5**
IALU. Команды загрузки	**J, K**	**D**	**X**	**A**	**M**	**W**
IALU. Команды сохранения	J, K	D	X	N	N	W
IALU. Арифметические и логические команды	J, K	D	X	N	N	W
IALU. Команды пересылки.	J, K	N	N	N	N	W
IALU. Команды загрузки константы	J, K	N	N	N	N	W
XY. Все типы команд	X1, X2, Y1, Y2	N	N	D	eX1	eX2
Устройство управления. Команды перехода. Предикат IALU	S	D	X	-	-	-
Устройство управления. Команды перехода. Предикат XY	S	D	N	N	N	X
Устройство управления. Дополнительные команды	M	D	N	N	N	X
Устройство управления. Команда расширения константы	L	D	-	-	-	-

Для линий конвейера вычислительных модулей обработка команды начинается только на третьей стадии, где выполняется декодирование команды. Само выполнение команды занимает два такта, что отражено в названиях EX1 и EX2. Для всех типов команд стадия X означает исполнение команды. Правда действия выполняются в зависимости от типа команды: для загрузки или сохранения – вычисление адреса, для команд перехода – выполнение перехода (если истина) и т.д.

Тоже самое верно и для стадии W. Для команд загрузки стадия W – это запись данных из буфера памяти в регистр-приемник. Для команд сохранения – это запись значения регистра-источника в буфер записи. Для команд пересылки – выполнение пересылки из одного регистра в другой. Для команд загрузки константы – запись константы в регистр. Для арифметических и логических команд IALU стадия W означает запись результата в регистр общего назначения.

Линия конвейера L короткая и включает одну стадию. На ней выполняется анализ, какой из линий конвейера предназначена константа расширения. На этой стадии будет происходить передача константы на одну из линий: J, K или S.

Стадия исполнения на IALU занимает один такт и результат операции доступен сразу для следующей команды. Несмотря на то, что запись результата в РОН будет выполнена позже. В отличие от IALU исполнение в модулях XY занимает две стадии и промежуточный результат первой стадии недоступен для следующей команды. Только итоговый результат второй стадии доступен для следующей команды. Это означает, что если следующая команда использует результат текущей команды, то это будет приводить к дополнительному такту останова конвейера. Команды IALU требуют готовности своих операндов на стадии декодирования. Это же верно и для команд вычислительных модулей. Поэтому если текущая команда выполняет загрузку регистра IALU из памяти, а следующая команда использует этот регистр как операнд, то будет происходить следующая ситуация: команда загрузки будет находится на стадии W и загружать регистр, а команда обработки будет в это время находится на стадии D и перехватывать загружаемое значение. Таким образом между этими командами образуется «пузырь» в три такта. Подобная ситуация вызовет образования «пузыря» и в вычислительном модуле, но он будет длиной только в один такт.

Все результаты записываются в регистры-приемники и флаги состояния на стадии конвейера W или EX2. Из рисунка 120 видно, что каждый вычислительный модуль имеет две линии конвейера обработки. Однако, это не значит – что по обеим линиям можно запустить выполнение произвольных команд. Имеются следующие ограничения:

- команды АЛУ могут выполняться только на линии X1(Y1);
- команды умножения могут выполняться только на линии X2(Y2);
- команды сдвигателя могут обрабатываться на линии X1(Y1) либо X2(Y2). Здесь все зависит от занятости соседней линии. Если линия 1 занята командой АЛУ – сдвиг выполняется на линии 2. Иначе команда сдвига выполянется на линии 1;
- команды CLU также не имеют фиксированной линии обработки. В основном для команд CLU используется линия 2. Линия 1 используется только если на линии 2 выполняется умножение.

На стадии исполнения команды, принадлежащие одной командной строке, исполняются параллельно. Если в одной и той же командной строке присутствуют две команды, использующие один и тот же регистр (одна в качестве операнда, а другая в качестве результата), операнд определяется значением регистра до исполнения этой строки. Например:

/* Начальные значения: R0 = 2, R1 = 3, R2 = 3, R3 = 8 */ R2 = R0 + R1 ; R6 = R2 * R3 (I) ;;

В этом примере R2 модифицируется первой командой и приобретает значение 5, и все же вторая команда видит входное значение R2 как 3, и результат, записываемый в R6 есть 24. Это правило гарантируется и для команд сохранения в памяти. В следующем примере с теми же начальными значениями регистр результата команды в первом поле используется во втором поле.

R2 = R0 + R1 ; [Address] = R2 ;;

Здесь результат использования R2 в качестве входного значения для команды сохранения в памяти приведет к сохранению значения 3. А в конце такта в регистр загрузится значение суммы 5.

Конвейер создает разного рода осложнения из-за перекрытия времен исполнения команд из различных командных строк. Например, возьмите последовательность двух командных строк, где вторая командная строка использует результат первой командной строки в качестве входного операнда. Из-за длинного конвейера результат может быть не готов к тому моменту, когда вторая команда осуществляет выборку своих операндов. В таком случае выполняется приостановка между первой и второй командными строками. Поскольку это может привести к потере производительности, программист или компилятор должны стараться минимизировать число таких случаев насколько это возможно. Такие комбинации, однако, вполне законны и результат корректен. Такого рода проблемы детально обсуждаются в разделе «Роль зависимостей и ресурсов в работе конвейера».

8.2.1 *Буфер выравнивания команд (IAB)*

IAB представляет собой FIFO на пять квадрослов, как это показано ниже (см. рисунок 123). При осуществлении устройством управления выборки квадрослова команд из памяти это квадрослово помещается в следующую позицию IAB. Если в IAB есть хотя бы одна полная командная строка, устройство управления может извлечь ее для исполнения.

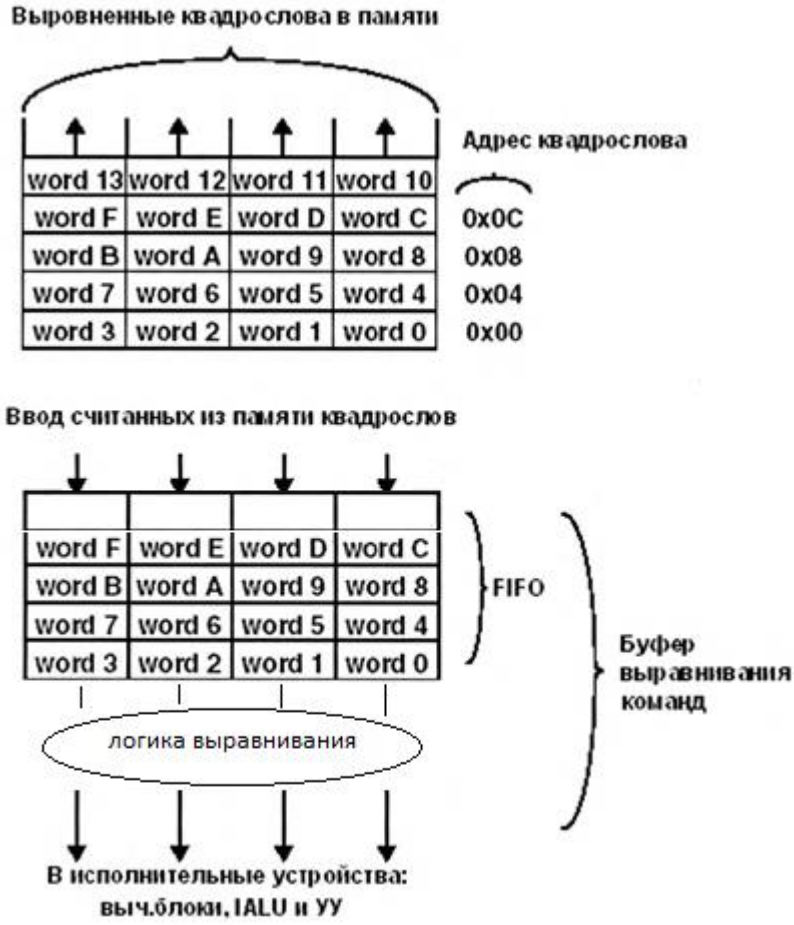


Рисунок 123 – Структура буфера выравнивания команд (IAB)

IAB выполняет следующие функции:

- Буферизует ввод (выбранных команд) из конвейера устройства выборки, обеспечивая независимость этого устройства от остальной части конвейера команд. Эта независимость позволяет устройству выборки работать в то время, как другие части конвейера приостановлены.
- Выравнивает ввод (невыровненных квадрослов) для формирования полных командных строк для исполнения. Такое выравнивание гарантирует, что полные командные строки, содержащие от одной до четырех команд, могут исполняться параллельно.
  Команды являются 32-битными словами, которые хранятся в памяти безотносительно к выравниванию квадрослов (128 бит) или длине командной строки (от одной до четырех команд). Это позволяет исключить потери памяти. Какие команды исполняются параллельно, определяется старшим значащим битом (MSB) кода операции каждой команды. Единичное значение этого бита является признаком того, что данный код операции замыкает командную строку.
- Распределяет команды командных строк по исполнительным устройствам: целочисленным ALU, вычислительным блокам и УУ.

Реализуя перечисленные функции, IAB обеспечивает исполнение целой командной строки без вставки дополнительных тактов приостановки или накладывания ограничений по выравниванию командных строк по границам квадрослов в памяти.

Пояснение значений функций буферизации, выравнивания и распределения, которые выполняются IAB, приведены на рисунке 124. На этом рисунке приведен пример того, как обычные (32-битные) слова команд хранятся невыровненными в памяти и как IAB буферизует, выравнивает и распределяет эти слова. В данном примере видно, что из памяти будет прочитано только 4 квадрослова за 4 такта, но процессор примет на обработку 8-ми линий команд, потратив минимум 8 тактов.

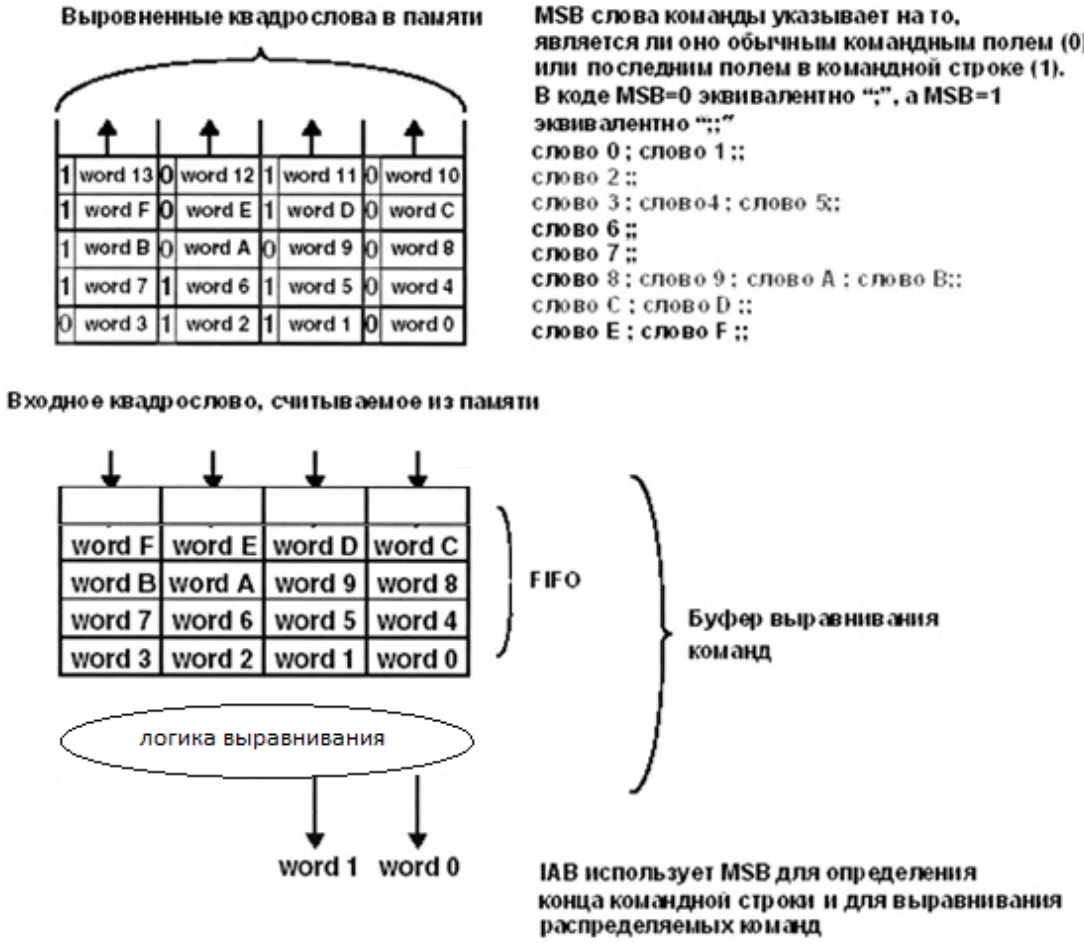


Рисунок 124 – IAB выравнивает командные строки для распределения/исполнения

8.2.2 *Буфер целевых адресов перехода (BTB)*

Элементы BTB образуют ассоциативную память (кэш), в которой записывается история переходов, прогнозировавшихся как предпринимаемых. BTB становится активным после включения командой BTBEN. Дополнительную информацию см. в подразделе «Включить/Выключить BTB».

Элементы BTB содержат тег перехода (адрес команды, замыкающей строку, содержащую команду перехода) и связанные с ним данные перехода (целевой адрес перехода). Тег перехода хранится в поле Tag BTB, а данные перехода хранятся в поле Target BTB.

На стадии IA конвейера, когда устройство управления отправляет текущий адрес в подсистему памяти для выборки очередного квадрослова, BTB также принимает этот адрес и сравнивает его с полями тегов всех своих линий в поисках прогнозируемых переходов. BTB фиксирует ситуацию промах BTB, если при этом не обнаруживается совпадения адреса на шине команд ни с одним из полей тегов BTB. BTB фиксирует ситуацию попадание в BTB, если он обнаруживает совпадение адреса

с одним из тегов, т.е. обнаруживает прогнозируемый переход, адрес которого уже помещен в BTB.

Когда BTB опознает адрес команды, которая ранее вызвала переход (попадание в BTB), BTB подставляет соответствующий целевой адрес (из поля Target BTB) в качестве следующего(!) адреса выборки для дальнейших команд. В течение времени исполнения программы BTB заполняется кэшированными прогнозируемыми переходами. В результате сохранения и использования этой информации о переходах, потери в производительности конвейера, вызванные переходами, снижаются с восьми или пяти тактов приостановки до нуля.

Косвенные переходы (RTI, RETI и CJMP) обслуживаются BTB специальным образом. Для этих переходов, данные перехода (целевой адрес) поступают из регистра, а не из BTB. Присутствующие в BTB биты CJMP и RTI указывают на регистр целевого адреса (cjmp, rti). Чтобы получить попадание в BTB на прогнозируемом косвенном переходе, программа должна позаботиться о том, чтобы значение в регистре было подготовлено заблаговременно, т.е. чтобы на стадии IA конвейера регистр содержал корректное значение. В случае, если регистр не подготовлен заранее, накладные расходы, обусловленные переходом, составят до 8-ми тактов, но какой-либо ошибки исполнения не возникнет. Накладные расходы связаны с тем, что BTB спрогнозирует переход, но адрес в регистре будет изменен после перехода. В этом случае позже будет обнаружен данный факт и процессор будет вынужден перезагрузить конвейер.

Кэш BTB содержит 128 записей и является 8-ми канальным блоком множественно ассоциативной кэш-памяти. Механизм замещения по LRU (Least Recently Used – дольше всех не используемый) без возможности защелкивания в BTB. Хотя процессор и поддерживает команды защелкивания BTBLOK и BTBELOK, но они не оказывают никакого влияния на BTB и исполняются как NOP команды.

Всякий раз, когда в программах с перекрытиями программные сегменты загружаются и выгружаются из внутренней памяти, BTB должен очищаться командой BTBINV с тем, чтобы сделать содержимое BTB недействительным.

В BTB кэшируются только команды перехода, располагающиеся во внутренней памяти и осуществляющие переход во внутреннюю память. Ширина кэшируемых целевых адресов составляет 21 бит. В BTB хранится только один тег на выровненное квадрослово команд. Вследствие этого в выровненном квадрослове может прогнозироваться только один переход. Если в программе требуется, чтобы два или более близкорасположенных переходов могли прогнозироваться, то между переходами должно быть вставлено от одной до трех команд NOP для гарантии того, что оба перехода не попадут в одно и то же выровненное квадрослово.

Структура BTB приведена на рисунке 125. Эта структура состоит из регистров трех типов; при этом каждый элемент BTB составлен из трех регистров различных типов. Элементы разбиты на 16 множеств, а внутри каждого множества имеется 8 каналов. Составными частями BTB являются множество, канал, тег, цель и LRU.

Индекс определяет множество для элемента BTB и представляет собой биты 5-2 адреса команды перехода. Использование четырех бит приводит к 16-ти множествам. Каждое множество содержит восемь каналов (адреса команд перехода с одними и теми же битами 5-2). Таким образом информация о некотором переходе может находится в одном из восьми каналов конкретного множества.

Тег определяет, является ли выбранная команда попаданием в BTB (т.е. присутствует ли ее адрес в BTB) или промахом BTB (т е. отсутствует ли ее адрес в BTB) при сравнении выбранной команды с содержимым каналов в подходящем множестве. В поле Тег BTB записаны биты 20-6 и 1-0 адреса замыкающей команды из командной строки, которой принадлежит переход.

Цель – это целевой адрес перехода. Устройство управления начинает выборку с целевого адреса при попадании в BTB. В поле Цели BTB записаны биты 20-0 адреса команды, на которую указывает команда перехода.

Бит достоверности указывает на то, является ли элемент (канал) действительным (=1) или недействительным (=0).

Бит CJMP указывает на то, является ли переход командой CJMP (=1) или нет (=0).

Бит RTI указывает на то, является ли переход командой RTI (=1) или нет (=0).

Бит ABS указывает на то, использует ли переход абсолютный адрес (=1) или PC-относительный адрес (=0).

LRU – это поле «Least Recently Used» («дольше всех неиспользуемый»), которое указывает на то, какой из каналов должен быть перезаписан в случае промаха BTB. Каждое множество имеет поле LRU, которое содержит значения LRU для всех каналов множества.

Рисунок 125 – Структура Буфера целевых адресов (BTB)

BTB и прогнозирование переходов связаны с конвейером команд следующим оюразом. Блок-схема функционирования устройства управления в части, касающейся BTB и операций прогнозирования переходов, с указанием затрат на переход (число теряемых тактов) приведена ниже (рисунок 126).

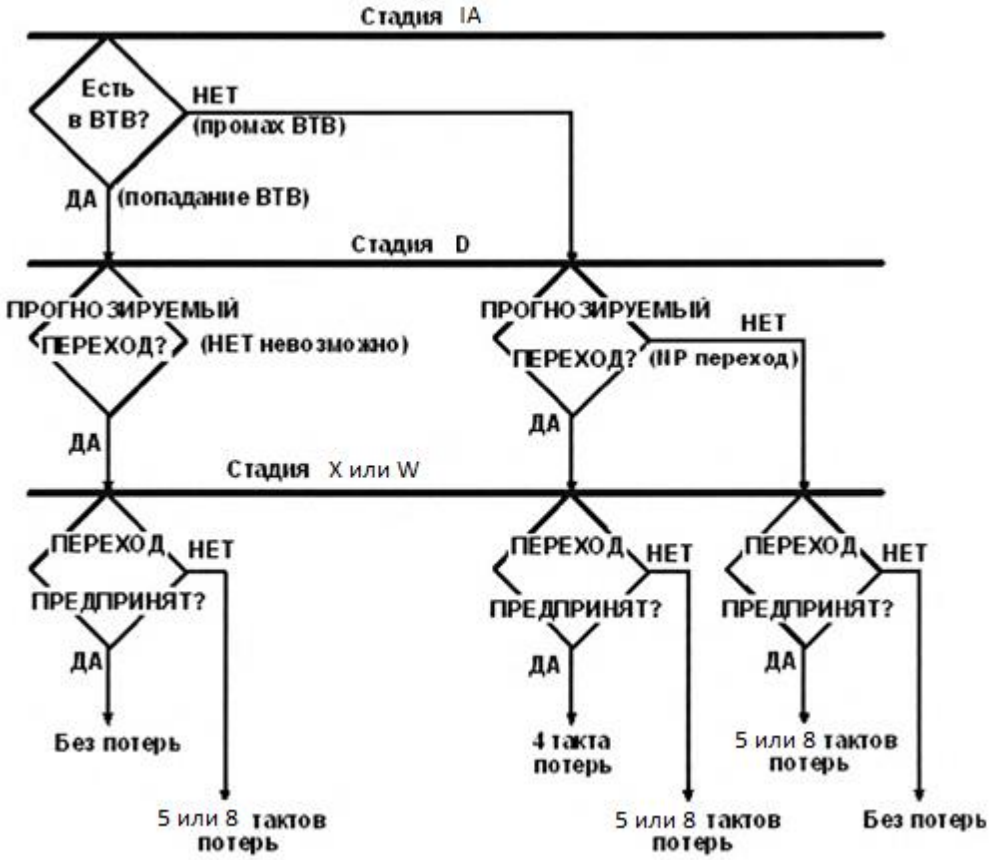


Рисунок 126 – Дерево потерь при прогнозировании перехода

Как показано на рисунке, использование BTB и прогнозирование переходов независимы. Устройство управления всегда сравнивает адрес выбираемой команды с содержимым BTB тогда, когда команда находится на стадии конвейера IA. Если происходит промах BTB, то какое следует делать предположение о переходе и когда принимать окончательное решение по нему, схема прогнозирования сообщает устройству управления на более поздних стадиях. На стадии конвейера D устройству управления известно, является ли команда переходом и является ли переход прогнозируемым или непрогнозируемым (NP). Именно на этой стадии конвейера принимается решение, откуда делать следующую выборку. Этот прогноз влияет также на то, будет ли переход помещен в BTB с тем, чтобы на следующем проходе произошло попадание в BTB.

По умолчанию все условные переходы являются прогнозируемыми (т.е. они прогнозируются как те, которые будут предприняты), какими являются переходы безусловные (последние трактуются так, как если бы они имели префикс IF TRUE). Только условные переходы с опцией «непрогнозируемый» (NP) прогнозируются как непредпринимаемые.

Такты потерь (приостановок) для переходов включают в себя:

- Когда переход является прогнозируемым и имеется попадание в BTB, устройство управления предполагает, что переход будет предпринят и начинает выборку с целевого адреса перехода при перемещении команды перехода со стадии конвейера IA на стадию IM. В случае правильного прогноза число теряемых тактов окажется равным нулю.
- Когда переход является прогнозируемым, но имеется промах BTB (т.е. BTB отключен или в нем нет соответствующего элемента), устройство

управления предполагает, что переход будет предпринят и начинает выборку с целевого адреса перехода при перемещении команды перехода со стадии конвейера D на стадию X. В случае правильного прогноза число теряемых тактов окажется равным четырем.

- Если условный переход является непрогнозируемым (NP) и имеется промах BTB (что всегда справедливо в отношении (NP)-переходов), устройство управления предполагает, что переход предпринят не будет, и не начинает выборку с целевого адреса перехода до момента вычисления значения предиката. Это вычисление производится на стадии конвейера X для IALU-предикатов и на стадии конвейера EX2 для предикатов вычислительных блоков. В случае правильного прогноза число теряемых тактов окажется равным нулю.
- Если условный переход неправильно прогнозируется (прогнозируемый переход не предпринимается, либо предпринимается непрогнозируемый (NP)-переход), то устройство управления не обнаруживает ошибочного прогноза вплоть до того момента, пока не окажется вычисленным предикат. Вычисление происходит на стадии конвейера X для IALU-предикатов и на стадии конвейера EX2 для предикатов вычислительных блоков. В этом случае имеется пять (для IALU-предикатов) или восемь (для предикатов вычислительных блоков) тактов потерь.

Когда выбираемая командная строка пересекает границу квадрослова, следует добавлять один такт потери во всех случаях. Использование директивы ассемблера. align 4 для команд JUMP и CALL на метки *label,* выровненные на границу квадрослова, исключает такты потерь для таких команд перехода.

Даже в предположении, что BTB отключен, что всегда приводит к промаху BTB, прогнозирование перехода воздействует на принятие решения конвейером команд (см. все случаи промаха BTB, упомянутые выше).

Помимо понимания того, в каких пределах BTB воздействует на конвейер, важно понимать и то, какие ограничения накладываются на размещение команд перехода в памяти, обусловленные использованием BTB.

В BTB кэшируются только переходы во внутренней памяти. Ширина кэшируемых целевых адресов составляет 21 бит. Допускается хранение в BTB только одного тега на выровненное квадрослово команд программы и, следовательно, может прогнозироваться только один переход на выровненное квадрослово. Если программисту требуется прогнозирование двух или более близколежащих переходов, то ему следует вставить от одной до четырех команд NOP между переходами для гарантии того, что оба перехода не попадут в одно и то же выровненное квадрослово. Может быть использована также директива ассемблера .align.

Чтобы избежать возможности помещения в одно и то же квадрослово памяти более чем одной команды, содержащей прогнозируемый переход, и тем спровоцировать непредсказуемое поведение BTB, такая комбинация команд и их размещения порождает предупреждение ассемблера. Ассемблер предупреждает, что им обнаружены два прогнозируемых перехода в командных строках, замыкающие команды которых расположены в пределах 4-х слов друг от друга. Кроме того, ассемблер сообщает, что в зависимости от выравнивания секций такая комбинация прогнозируемых переходов и их размещения в памяти может нарушить ограничение, согласно которому командные строки с такими переходами не могут завершаться в одном и том же квадрослове.

Полезно исследовать, к какому содержимому BTB приводит различное размещение слов в памяти. К примеру, фрагмент кода в Листинге 33 содержит прогнозируемый переход.

Листинг 33 – Прогнозируемые переходы, выровненные квадрослова и BTB

```
nop; nop; nop; nop;;
jump СЮДА; nop;;
nop; nop; nop; nop;;
```

В памяти каждая команда занимает адрес, и набор из четырех адресов составляет квадрослово. Размещение квадрослов в памяти показано на рисунке 127 и рассмотрено в подразделе «Буфер выравнивания команд (IAB)». Адрес квадрослова есть адрес первой команды в квадрослове.

В зависимости от того, как код из Листинга 33 выровнен в памяти, может оказаться, что по квадрословному адресу 0x00000004 располагается квадрослово, показанное на рисунке ниже:

Рисунок 127 –Квадрослова и команды перехода (1)

Если так, то элемент BTB для перехода будет содержать:

Тег = 0x00000004, Целевой адрес = СЮДА

Однако, код из Листинга 30 может оказаться выровненным в памяти иначе. Например, этот код может оказаться выровненным так, что квадрословный адрес 0x00000004 (первая строка) и 0x00000008 (вторая строка) содержат квадрослова, показанные рисунке ниже.

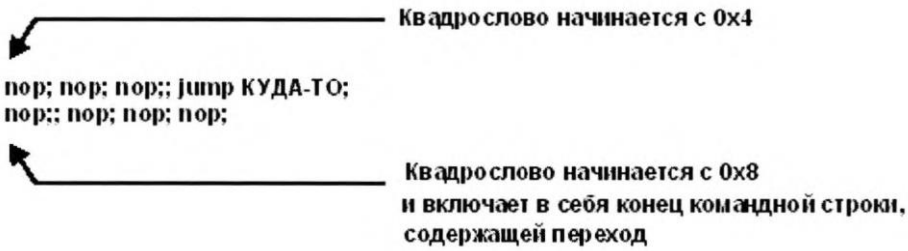


Рисунок 128 –Квадрослова и команды перехода (2)

Если так, то элемент BTB для перехода будет содержать:

Тег = 0x00000008, Целевой адрес = КУДА-ТО

Если BTB включен, текущее значение PC (биты 20-6 и 1-0) сравнивается со значениями тегов BTB на стадии IA конвейера. Если обнаружено совпадение, процессор модифицирует PC, чтобы он отражал целевой адрес перехода, хранящийся в BTB, а устройство управления продолжает выборку последующих квадрослов с адреса, равного модифицированному PC. Если соответствие не обнаружено, процессор не модифицирует PC продолжает выборку последующих квадрослов с адреса, равного немодифицированному PC.

Когда та же команда достигает стадии конвейера D, она опознается как команда перехода. Если было совпадение с BTB, никаких исключительных действий

по переходу не предпринимается. Значение программного счетчика PC уже было модифицировано и устройство управления уже осуществило выборку с целевого адреса перехода. Если не было совпадения с BTB, устройство управления абортирует четыре команды, выбранные до достижения стадии D (четыре такта приостановки) и процессор модифицирует PC, чтобы он отражал целевой адрес перехода и начинает выборку квадрослов с адреса, равного модифицированному PC. Устройство управления обновляет BTB, пополняя его целевым адресом перехода, так чтобы в следующий раз, когда снова встретится эта команда перехода, с большой вероятностью наблюдалось совпадение с BTB.

Содержимое BTB варьируется в зависимости от размещения команд в памяти, поскольку:

- устройство управления осуществляет выборку полного квадрослова за один раз;
- командная строка может занимать меньше полного квадрослова, занимать полное квадрослово или перекрывать два квадрослова;
- командная строка может начинаться с адреса, отличного от адреса, выровненного на границу квадрослова.

Поскольку BTB может хранить только один целевой адрес перехода для каждого выровненного квадрослова программного кода, важно освоить технику кодирования, учитывающую эту особенность BTB. Пример кода на рисунке 129 порождает непредсказуемые результаты в аппаратуре, поскольку этот код (в зависимости от размещения в памяти) может попытаться принудить BTB к хранению нескольких целевых адресов перехода для одного выровненного квадрослова.

Рисунок 129 – Квадрослова и команды перехода (3). Недопустимая близость

Ситуация может быть исправлена использованием команд NOP, чтобы принудить команды перехода отстоять друг от друга по крайней мере на четыре команды.

```
jump ПЕРВЫЙ_ПЕРЕХОД; LC1 = yR16;;
jump ВТОРОЙ_ПЕРЕХОД; R29 = R27; NOP; NOP;;
/* Добавление NOPob отодвигает конец строки для 2-ой команды */
```

Хотя добавление этих команд NOP и увеличивает размер кода, оно не влияет на производительность этого кода.

Другой способ управлять взаимоотношением выравнивания кода внутри квадрослов и содержимым BTB состоит в использовании директивы ассемблера .align_code 4. Эта директива вынуждает непосредственно следующий за ней код быть выровненным на границу квадрослова:

```
jump ПЕРВЫЙ_ПЕРЕХОД; LC1 = yR16;;
.align 4;
/* Вынуждает выравниванием на границу квадрослова сдвинуть конец следующей командной строки */
jump ВТОРОЙ_ПЕРЕХОД; R29 = R27;;
```

Если попадание BTB является вычисляемым переходом, то в качестве целевого адреса используется значение регистра reti или cjmp (в зависимости от команды). В этом случае любое изменение значения регистра до того, как переход

фактически произойдет, заставляет процессор удалять выбранные команды и повторить прохождение потока команд, как если бы попадания не было.

8.2.3 *Воздействие условных переходов на конвейер*

Правильное прогнозирование условных переходов критично для производительности конвейера. На прогнозирование влияют и подвержены его влиянию все линии (выборки, IALU, вычислительного блока) конвейера команд. Потоки конвейера для различных переходов различаются между собой и определяется следующими критериями:

- прогнозирование переходов (см. подраздел «Отработка условных команд»);
- попадание и промах BTB (см. подраздел «Буфер целевых адресов перехода (BTB)»);
- стадия конвейера, на которой принимается решение по значению предиката, т.е. стадия X или W (см. рисунок 126);
- прогнозирование задается программистом или компилятором. Обычно признак прогнозирования есть TRUE или «переход предпринимается». Когда программист использует опцию (NP) в команде управления потоком, признак прогнозирования принимает значение «переход не предпринимается». Вообще говоря, признак прогнозирования указывает на предположение по умолчанию: предпринимается ли переход или нет. К примеру, представьте себе цикл, в котором переход предпринимается n-l раз, и всегда более чем один раз. Установка такого бита (признак прогнозирования) имеет два следствия:
  - переход помещается в BTB;
  - на стадии D процессор идентифицирует команду как команду перехода и продолжает выборку с целевого адреса перехода, вне зависимости от значения предиката.

Если команда перехода идентифицируется как попадание в BTB, то процессор осуществляет выборку по целевому адресу непосредственно вслед за выборкой команды перехода. В этом случае накладные расходы на правильное прогнозирование отсутствуют.

По различным предикатам решения принимаются на различных стадиях. Решения по IALU-предикатам (таким как LC0E, LC1E, TRUE, JEQ, JLE, JLT, KEQ, KLE, KLT.) принимаются на стадии X. Решение по всем(!) другим предикатам принимается на стадии W(EX2). Предикаты LC0E, LC1E, JEQ, JLE, JLT, KEQ, KLE, KLT обновляются на стадии X. Флаги признаков вычислительных блоков обновляются на стадии конвейера EX2. Другие флаги (BM, FLAG0_IN, FLAG1_IN, FLAG2_IN, FLAG3_IN и TRUE) асинхронны по отношению к конвейеру, поскольку они устанавливаются внешними событиями. В дальнейшем при выполнении условных команд всякая ссылка на IALU-предикаты означает предикаты LC0E, LC1E, TRUE, JEQ, JLE, JLT, KEQ, KLE, KLT. Ссылка на предикаты вычислительных модулей означает все другие предикаты кроме IALU.

В различных ситуациях порождаются различные потоки конвейера и, как следствие, реализуется разная производительность. Параметрами, от которых зависят затраты на переход, являются следующие:

- прогнозирование – переход должен предприниматься или нет;
- переход по предикату IALU или вычислительного блока;
- попадание в BTB – промах BTB;
- реальный переход – предпринимается или нет.

Возможны 16 комбинаций этих параметров. Следующие комбинации игнорируются:

- если прогноз «не предпринимается», то попадание в BTB невозможно;
- если прогноз «не предпринимается» и переход не предпринимается в самом деле, то поток конвейера точно такой, как если бы никакой команды перехода не было (т.е. команда перехода есть NOP команда);
- если прогноз «предпринимается» и переход предпринимается в самом деле, то поток идентичен потоку для команд IALU и вычислительного блока.

Различные потоки конвейера показаны в таблицах ниже. На каждой диаграмме показан поток для каждой комбинации и значение затрат для него. Затраты на переход можно подытожить таким образом:

- прогноз «не предпринимается» и переход не предпринимается – затрат нет;
- попадание в BTB, переход предпринимается – затрат нет;
- промах BTB, прогноз «предпринимается», переход предпринимается – четыре такта;
- прогноз «предпринимается», переход не предпринимается (либо попадание или промах BTB); либо переход предпринимается по IALU-предикату (пять тактов) или предикату вычислительного блока (восемь тактов).

Если прогноз «не предпринимается», то не может быть попадания в BTB, поскольку «прогноз «предпринимается»» есть условие для помещения нового элемента в BTB.

К затратам на переход должен добавляться один такт, если имеет место ситуация, когда переход предпринимается и целевая командная строка пересекает границу квадрослова.

В таблице 31 показан прогнозируемый переход, который основывается на значении IALU-предиката. BTB идентифицирует команду #2 как прогнозируемый переход на стадии конвейера IA. Поскольку переход правильно спрогнозирован как предпринимаемый и BTB содержит команду перехода (попадание в BTB), то содержимое конвейера исполняется непрерывно (ни одна из ступеней конвейера не опустошена) и потерянных тактов нет (т.е. затрат на переход). Отметьте, что конвейер вычисляет значение IALU-предиката (флаг JEQ, установленный командой #1), когда условная команда достигает стадии конвейера X. В данном примере команда #1 расположена по адресу 0.

Таблица 31 – Прогнозируемый условный переход по IALU-предикату (переход предпринимается) и имеется попадание в BTB

Код программы
#1: J0 = J1+J2;; /* J0=0 */
#2:IF JEQ, JUMP 20;; /* переход на 20*/

Такт	IA	IM	IW	P	D	X	A	M	W
1	$0								
Команда #2 есть прогнозируемый переход по предикату IALU с реально предпринимаемым переходом и попаданием в BTB									
2	$20	$0							
3		$20	$0						
4			$20	#1					

Такт	IA	IM	IW	P	D	X	A	M	W
5				#2	#1				
6				#20	#2	#1			
7					#20	#2	#1		
8						#20	#2	#1	
9							#20	#2	#1
10								#20	#2
11									#20

Случай, описанный в таблице 31, соответствует ситуации, когда код программы выполняется уже не первый раз. Только в этом случае информация о переходе может быть в BTB. В таблице 32 описан прогнозируемый переход, который основывается на значении IALU-предиката. Поскольку переход правильно спрогнозирован как предпринимаемый и поскольку BTB не содержит команды перехода (промах BTB), содержимое конвейера не является непрерывно исполняемым (четыре ступени опустошаются) и имеется четыре потерянных такта (затраты на переход). Отметьте, что процессор выполняет переход на стадии D как прогнозируемый. Конвейер вычисляет значение IALU-предиката (флаг JEQ, установленный командой #1), когда условная команда достигает стадии конвейера X.

Таблица 32 – Прогнозируемый условный переход по IALU-предикату (переход предпринимается) и имеется промах BTB

Код программы
#1: J0 = J1+J2;; /* J0=0. адрес команды равен 0 */
#2:IF JEQ, JUMP 20;; /* переход на адрес 20 */
#3: …..

Такт	IA	IM	IW	P	D	X	A	M	W
1	$0								
2	$4	$0							
3	$8	$4	$0						
4	$12	$8	$4	#1					
5	$16	$12	$8	#2	#1				
6		$16	$12	#3	#2 Jump 20	#1			
Команда #2 есть прогнозируемый переход по предикату IALU с реально предпринимаемым переходом и промахом в BTB									
7	$20					#2	#1		
8		$20					#2	#1	
9			$20					#2	#1
10				#20					#2
11					#20				
12						#20			
13							#20		
14								#20	
15									#20

Описанный в таблице 32 случай соответствует ситуации, когда код программы выполняется первый раз. В этом случае в BTB еще нет информации о переходе и она (информация) заносится в момент выполнения перехода на стадии D. В таблице видно, как образуются такты потерь при подобных ситуациях.

В таблице 33 показан непрогнозируемый (NP) переход, который основывается на значении предиката вычислительного блока.

Таблица 33 – Непрогнозируемый (NP) условный переход по предикату вычислительного блока-переход предпринимается

Код программы
#1: R0 = R1+R2;; /* R0=0, адрес команды равен 0 */
#2:IF AEQ, JUMP 20 (NP);; /* переход на адрес 20 */
#3:
…

Такт	**IA**	**IM**	**IW**	**P**	**D**	**X**	**A**	**M**	**W**
1	$0								
2	$4	$0							
3	$8	$4	$0						
4	$12	$8	$4	#1					
5	$16	$12	$8	#2	#1				
6		$16	$12	#3	#2	#1			
7			$16	#4	#3	#2	#1		
8	$20			#5	#4	#3	#2	#1	
9		$20		#6	#5	#4	#3	#2	#1
10			$20	#7	#6	#5	#4	#3	#2 Jump 20
Команда #2 есть непрогнозируемый переход по предикату вычислительного блока, но с реально предпринимаемым переходом									
11	$20								
12		$20							
13			$20						
14				#20					
15					#20				
16						#20			
17							#20		
18								#20	
19									#20

Поскольку переход неправильно спрогнозирован как непредпринимаемый, содержимое конвейера не является непрерывно исполняемым (восемь ступеней опустошаются) и имеется восемь потерянных тактов (затраты на переход с 11 по 18 такт). Отметьте, что конвейер вычисляет значение предиката вычислительного блока (флаг AEQ, установленный командой #1), когда условная команда достигает стадии конвейера EX2. Интересной особенностью данного примера также является процесс чтения квадрослов команд. Предположим, что в нашем примере все линии команд за #2 состоят также из одной команды. Видно, что после чтения 5-го квадрослова по адресу 16, процесс чтения прекращается, т.к. буфер IAB заполнен. Когда конвейер обработки принимает команду #4, в IAB появляется одна свободная запись и устройство управления тут же инициирует чтение 6-го квадрослова. Однако,

при достижении командой #2 стадии W(EX2), все что выполнил процессор за командой #2 удаляется как напрасно сделанная работа.

Таблица 34 – Непрогнозируемый (NP) условный переход по предикату IALU-переход предпринимается.

Код программы
#1: J0 = J1+J2;; /* J0=0. адрес команды равен 0 */
#2:IF JEQ, JUMP 50 (NP);; /* переход на адрес 50 */
#3: …

Такт	**IA**	**IM**	**IW**	**P**	**D**	**X**	**A**	**M**	**W**
1	$0								
2	$4	$0							
3	$8	$4	$0						
4	$12	$8	$4	#1					
5	$16	$12	$8	#2	#1				
6		$16	$12	#3	#2	#1			
7			$16	#4	#3	#2 Jump 50	#1		
Команда #2 есть непрогнозируемый переход по предикату IALU, но с реально предпринимаемым переходом									
8	$50						#2	#1	
9		$50						#2	#1
10			$50						#2
11				$50					
12					$50				
13						$50			
14							$50		
15								$50	
16									$50

В таблице 34 показан непрогнозируемый (NP) переход, который основывается на значении IALU-предиката. Поскольку переход неправильно спрогнозирован как непредпринимаемый, содержимое конвейера не является непрерывно исполняемым (пять ступеней опустошаются) и имеется пять потерянных тактов (затраты на переход с 11 по 15 такт). Отметьте, что конвейер вычисляет значение IALU-предиката (флаг JEQ, установленный командой #1) и выполняет переход, когда условная команда достигает стадии конвейера X. Таким образом, можно заметить, что если переход отмечен как прогнозируемый, то, в случае промаха в BTB, переход будет выполняться всегда на стадии конвейера D. На этой же стадии происходит и запись информации в BTB. Если же переход не прогнозируемый, то реально переход может выполнятся на стадии X или на стадии W.

Таблица 35 показывает прогнозируемый переход, который основывается на значении предиката вычислительного блока. Рассмотрен случай, когда информация о переходе уже находится в BTB (имеем попадание в BTB), но в действительности переход не выполняется, т.к. значение предиката оказывается ложным. В этом примере мы имеем потери от неправильного прогнозирования в размере 8 тактов (такты с 11 по 18). Обратите особое внимание на такт 11, где процессор опять начинает выборку квадрослова 0, но адрес команды равен 3. Буфер BTB в данном случае не

сработает, т.к. в поле тега хранятся два младших бита адреса команды предсказания (они равны 2) и их значение меньше адреса 3 (для попадания нужно tag[1:0]>=IA[1:0] и совпадение остальных бит).

Таблица 35 – Прогнозируемый условный переход по предикату вычислительного блока (переход не предпринимается)

Код программы
#1: R0 = R1+R2;; /* R0!=0, адрес команды равен 0 */
#2:IF AEQ, JUMP 50;; /* нет перехода на адрес 50 */
#3: …

Такт	IA	IM	IW	P	D	X	A	M	W
1	$0								
В такте чтения квадрослова по адресу 0 мы имеем попадание в BTB, что указывает на то, что в квадрослове по адресу 0 находится команда перехода с целевым адресом 50. Значение целевого адреса используется в следующем такте для выборки команд									
2	$50	$0							
3	$54	$50	$0						
4	$58	$54	$50	#1					
5	$62	$58	$54	#2	#1				
6		$62	$58	#50	#2	#1			
7			$62	#51	#50	#2	#1		
8				#52	#51	#50	#2	#1	
9				#53	#52	#51	#50	#2	#1
10				#54	#53	#52	#51	#50	#2 Jump 3
Команда #2 есть прогнозируемый переход по предикату вычислительного блока с попаданием в BTB, но с реально не предпринимаемым переходом									
11	$3								
12	$4	$3							
13		$4	$3						
14			$4	#3					
15					#3				
16						#3			
17							#3		
18								#3	
19									#3

Таблица 36 – Прогнозируемый условный переход по предикату IALU - переход не предпринимается

Код программы
#1: J0 = J1+J2;; /* J0!=0, адрес команды равен 0 */
#2:IF JEQ, JUMP 50 ;; /* нет перехода на адрес 50 */
#3: …

Такт	IA	IM	IW	P	D	X	A	M	W
1	$0								
2	$50	$0							
3	$54	$50	$0						
4	$58	$54	$50	#1					
5	$62	$58	$54	#2	#1				
6		$62	$58	#50	#2	#1			
7			$62	#51	#50	#2 Jump 3	#1		
Команда #2 есть прогнозируемый переход по предикату IALU с попаданием в BTB, но с реально не предпринимаемым переходом									
8	$3						#2	#1	
9	$4	$3						#2	#1
10		$4	$3						#2
11			$4	#3					
12					#3				
13						#3			
14							#3		
15								#3	
16									#3

Рассмотрим подобную ситуацию, но в случае с предикатом IALU. Пример приведен в таблице 36. Видно, что в случае предиката IALU потери составляют 5 тактов против 8-ми для предиката вычислительного модуля.

Далее рассматриваются ситуации более сложные, когда прогнозируемые переходы дают промах в BTB, но при этом реально переходы не выполняются. Таблица 37 показывает прогнозируемый переход, который основывается на значении предиката вычислительного блока. Поскольку переход неправильно спрогнозирован как предпринимаемый и поскольку BTB не содержит команду перехода (промах BTB), то содержимое конвейера не является непрерывно исполняемым (восемь ступеней опустошаются) и имеется восемь потерянных тактов (затраты на переход).

Таблица 37 – Прогнозируемый условный переход по предикату вычислительного блока (переход не предпринимается) и имеется промах в BTB

Код программы
#1: R0 = R1+R2;; /* R0!=0, адрес команды равен 0 */
#2:IF AEQ, JUMP 50;; /* нет перехода на адрес 50 */
#3: …

Такт	IA	IM	IW	P	D	X	A	M	W
1	$0								
2	$4	$0							
3	$8	$4	$0						
4	$12	$8	$4	#1					
5	$16	$12	$8	#2	#1				
6		$16	$12	#3	#2 Jump 50	#1			
Переход прогнозируемый и имеется промах в BTB, поэтому процессор выполняет переход по адресу 50 и записывает информацию в BTB.									
7	$50					#2	#1		
8	$54	$50					#2	#1	
9	$58	$54	$50					#2	#1
10	$62	$58	$54	#50					#2 Jump 3
Поскольку команда #2 обнаруживает, что предсказанный переход оказался ложным, она вызывает переход по адресу 3, чтобы восстановить правильный ход программы									
11	$3								
12	$4	$3							
13		$4	$3						
14			$4	#3					
15					#3				
16						#3			
17							#3		
18								#3	
19									#3

Таблица 38 показывает прогнозируемый переход, который основывается на значении IALU-предиката. Поскольку переход неправильно спрогнозирован как предпринимаемый и поскольку BTB не содержит команду перехода (промах BTB), то содержимое конвейера не является непрерывно исполняемым и имеется пять потерянных тактов.

Таблица 38 – Прогнозируемый условный переход по предикату IALU (переход не предпринимается) и имеется промах в BTB

Код программы
#1: J0 = J1+J2;; /* J0!=0, адрес команды равен 0 */
#2:IF JEQ, JUMP 50 ;; /* нет перехода на адрес 50 */
#3: …

Такт	IA	IM	IW	P	D	X	A	M	W
1	$0								
2	$4	$0							
3	$8	$4	$0						
4	$12	$8	$4	#1					
5	$16	$12	$8	#2	#1				
6		$16	$12	#3	#2 Jump 50	#1			
В такте 6 срабатывает прогнозируемый переход.									
7	$50					#2 Jump 3	#1		
В такте 7 команда #2 обнаруживает ошибочность прогноза и исправляет ситуацию.									
8	$3						#2	#1	
9	$4	$3						#2	#1
10		$4	$3						#2
11			$4	#3					
12					#3				
13						#3			
14							#3		
15								#3	
16									#3

8.2.4 *Роль зависимостей и ресурсов в работе конвейера*

Процессор допускает любую последовательность командных строк, при условии, что каждая отдельная строка корректна. Конвейеризованное исполнение команд вызывает перекрытие исполнения различных строк по времени. Это перекрытие может привести к следующим двум проблемам:

- зависимость;
- конфликт по ресурсам.

Ситуация зависимости вызывается командой, которая использует в качестве входа результат предыдущей команды, если данные предыдущей команды не готовы к тому моменту, когда текущей команде требуется операнд.

Конфликт по ресурсам вызывается конфликтом по доступу к блоку памяти или задержкой, обусловленной занятостью памяти.

Доступ к памяти использует конвейер памяти, который функционирует параллельно с конвейером команд. Во время выборки команд конфликты по доступу к памяти проявляются как задержки исполнения во время стадий IA и IM памяти I-шины. При другом доступе к памяти (для данных) конфликты по доступу проявляются как задержки исполнения во время стадий A и M конвейера памяти J/K-шин. Дополнительную информацию о конвейере памяти см. в разделе «Память и шины».

В данном разделе подробно описываются различные варианты приостановок. *Приостановка* – это любая задержка исполнения команды, вызванная одной из двух упомянутых выше причин. Хотя приводимая в данном руководстве информация достаточно подробна, может быть несколько случаев, которые здесь не определены либо ситуации, которые не всегда осознаются разработчиком системы. Точное поведение может быть воспроизведено только путем использования симулятора процессора.

В таблице 38 перечислены приостановки для процессоров, обусловленные зависимостями. Приостановки по зависимости при исполнении командной строки 2 происходят, когда ресурс или шина, используемая командной строкой 1, требуется для командной строки 2. Если не оговорено противное, тип данных или размер не влияют на приостановки, обусловленные зависимостями. Количество тактов приостановки в каждом конкретном случае приведено в столбце КТП. В такты приостановки не включается такт исполнения команды. Для случаев обращения к внешней памяти такты приостановки не приводятся ввиду их зависимости от настроек памяти и соотношения между частотами ядра и внешней шины. Для случая чтения регистра SOC-шины значение приостановки будет составлять 8 либо 9 тактов ядра, если при доступе к шине не будет конфликта (параметр SOC = 8 или 9). Выбор между 8-ю и 9-ю тактами обусловлен тем, что запись в буфер SOC-интерфейса идет с частотой ядра, а чтение с частотой SOC-шины. Подробно доступ к регистру SOC-шины для чтения можно представить как:

- Запись запроса на чтение в буфер SOC-интерфейса – 1 такт ядра. Данный такт не является тактом приостановки, а соответствует исполнению команды. В следующем такте ядро приостанавливается до получения данных.
- Чтение буфера запросов SOC-интерфейса во временный буфер SOC-шины – 1 или 2 такта ядра в зависимости от случайного соотношения клоков ядра и SOC-шины (клок SOC-шины в 2 раза меньше клока ядра).
- Арбитраж доступа к SOC-шине – 2 такта ядра (такт SOC-шины).
- Чтение регистра периферийного устройства, подключенного к SOC-шине и запись прочитанного значения во входной буфер SOC-интерфейса – 2 такта ядра (такт SOC-шины).
- Чтение возвращаемого значения из входного буфера SOC-интерфейса во временный буфер ядра – 2 такта ядра (такт SOC-шины).
- Запись значения в регистр приемника – 1 такт ядра.
- В следующем такте конвейер ядра может продолжить движение.

Таблица 39 – Перечень приостановок по зависимости от ресурсов

Команда 1	Команда 2	Тип зависимости	КТП
Любая выч.команда Rs1 = F (Rm, Rn);	**Любая выч.команда** Rs = F (Rm2, Rn2) ;	**Использование Dreg** Rs1 = Rm2 Rs1 = Rn2	1
Любая команда загрузки Dreg (из внутренней памяти или ядра) Rs1 = [address] ;; Rsl = Ureg ;;	**Любая выч.команда** Rs = F(Rm2, Rn2);	**Использование того же Dreg** Rsl = Rm2 Rs1 = Rn2	**1**
Любая команда загрузки Dreg из SOC-шины Rs1 **=** SOC_Ureg ;;	**Любая выч.команда Rs2 = F (Rm2, Rn2);**	**Использование того же регистра Dreg Rs1= Rm2 Rs1 = Rn2**	**2+SOC**

Команда 1	Команда 2	Тип зависимости	КТП
Команды загрузки/пересылки регистров IALU Jsl = [address] ; Jsl = Ureg ; Jsl = <Imm> ;	**Любая целочисленная команда IALU** Js2 = F(Jm2, Jn2);	**Использование того же Jx** Jm2 = Js1 Jn2 = Js1	**3**

Условная команда: целочисленная IALU или загрузка/сохр. Ureg (с предикатом выч.блока)
if<compute_cond.>;
do,Jsl = F(Jm,Jn);
if (compute_cond.);
do,Ureg =[Jsl+=] ;
if (compute_cond.);
do,[Jsl +=]= Ureq;
	Любая целочисленная команда IALU
	Js = F (Jm2, Jn2) ;
		Использование того же Jx
		Jm2 = Js1
		Jn2 = Js1
			3

Условная целочисленная команда IALU (с предикатом выч.блока) if <compute_cond.>; do,Jsl = F (Jm, Jn) ;	**Сложение/вычитание IALU с переносом/займом** Js = Jm + Jn + Jc; Js = Jm - Jn + Jc -1;	**Ожидание бита переноса Jc**	3
Условная целочисленная команда IALU (с предикатом выч.блока) if <compute_cond.>; do,Js1 = F(Jm,Jn) ;	**Условная команда с IALU- предикатом** if <IALU_cond> ;;	**Всегда**	3
Команды загрузки/пересылки J31 6 J31 = [address] ; J31 = Ureg ; J31 = <Imm> ;	**Сложение/вычитание IALU с переносом/займом** Js = Jm + Jn + Jc; Js = Jm - Jn + Jc -1;	**Ожидание бита переноса Jc на стадии X**	**3**
Команды загрузки/пересылки J31 J31 = [address] ; J31 = Ureg ; J31 =<Imm> ;	**Условная команда с J-IALU предикатом** if <J-IALU_cond>, jump ; if <J-IALU_cond>;do ;	**Всегда ожидание на стадии X**	**3**
Команда загрузки/пересылки J31 из SOC J31 = SOC_Ureg ;	**Сложение/вычитание IALU с переносом/займом** Js = Jm + Jn + Jc; Js = Jm - Jn + Jc -1;	**Всегда ожидание на стадии X**	**4+ SOC**
Команда загрузки/пересылки J31 из SOC J31 = SOC_Ureg ;	**Условная команда с J-IALU предикатом** if <J-IALU_cond>, jump ; if <J-IALU_cond>;do ;	**Всегда ожидание на стадии X**	**4+ SOC**
Команда загрузки/пересыпки J31 из SOC J31 = SOC_Ureg ;	**Любая целочисленная команда IALU** Js2 = F (Jm2, Jn2);	**Всегда ожидание на стадии X**	**4+ SOC**

Команда 1	Команда 2	Тип зависимости	КТП
Команда загрузки/пересылки Jx из SOC Js1 = SOC_Ureg ;	**Любая целочисленная команда IALU** Js2 = F (Jm2, Jn2) ;	**Использование тех же регистров Jx** Js1 = Jm2 Js1 = Jn2	**5+ SOC**
Команды слож/вычит IALU с результатом, помещаемым в JB/JL[6] JB = Jm +| - Jn ; JL = Jm +| - Jn ;	**Команды слож/вычит IALU с кольцевым буфером** Js = Jm2 +|-Jn2 (CB);	**Использование JB/JL, принадлежащего Jm2**	**4**
Команды слож/вычит IALU с результатом, помещаемым в JB/JL[6] JB = Jm +| - Jn ; JL = Jm +| - Jn ;	**Загрузка Dreg, использующая постмодификацию и кольцевой буфер** Dreg =(CB)[Jm2+=Jn2];	**Использование JB/JL, принадлежащего Jm2**	**4**
Команды загрузки/пересылки JB/JL JB|JL = [address] ; JB|JL = Ureg ; JB|JL = <Imm> ;	**Команды слож/вычит IALU с кольцевым буфером** Js = Jm2 +| -Jn2 (CB);	**Использование JB/JL, принадлежащего Jm2**	**4**
Команды загрузки/пересылки JB/JL JB|JL = [address] ; JB|JL = Ureg ; JB|JL = <Imm> ;	**Загрузка Dreg, использующая постмодификацию и кольцевой буфер** Dreg =(CB)[Jm2+=Jn2];	**Использование JB/JL, принадлежащего Jm2**	**4**
Команды загрузки/пересылки JB/JL из SOC[6] JB|JL = SOC_Ureg ;	**Любая целочисленная команда IALU, использующая кольцевой буфер**	**Использование JB/JL, принадлежащего Jm2**	**4+ SOC**
Команда загрузки счетчика цикла LCx = [address] ; LCx = Ureg ;	**Условная команда (NP), использующая предикат счетчика цикла** if LCxE, jump lab (NP);	**Использование того же LCx**	**3**
Любая команда управления BTB BTBEN ; BTBDIS ; BTBLOCK ; BTBELOCK ; BTBINV ;	**Любая команда**	**Всегда**	**9**
Команда загрузки/пересылки SQCTL SQCTL = [address] ; SQCTL = Ureg ; SQCTL = <Imm> ;	**Любая команда**	**Всегда**	**9**
Команда CLU либо загрузка регистра TRx TRsq1 = F( ); TRsq1 = Rmq	**Команда CLU** TRsq2 = F(TRmd2,TRnd2,Rm);	**Использование того же TRx** TRmd2 = TRsq1 TRnd2 = TRsq1	1
команда загрузки CMCTL CMCTL =Rm;	**Команда XCORRS**	**Всегда зависим. CMCTL**	2
Команда XCORRS( );	**Команда CLU** TRs1 = F( );	**Если есть зависимость по регистрам**	2

Команда 1	Команда 2	Тип зависимости	КТП
Примечания: 1 SOC_Ureg – это универсальный регистр Ureg из групп с номерами в диапазоне от 0x20 до 0x3F, т.е. Ureg-регистрлюбого из модулей, выходящих на SOC-шину. 2 Предикат IALU означает одно из условий LC0E, LC1E, TRUE, JEQ, JLE, JLT, KEQ, KLE, KLT. 3 Предикат вычислительного блока означает любой другой предикат, кроме предиката IALU			

Имеются некоторые особенности генерации приостановок конвейера при выполнении отдельных групп команд:

- Если текущая команда есть Jn = <Imm>, а следующая команда IALU использует Jn в качестве операнда, то согласно таблицы мы имеем 3 такта приостановки конвейера. При этом для выполнения команды Jn = <Imm>; компилятор может выбрать линию конвейера J либо K. Если для загрузки J регистра компилятор выбрал линию J, то приостановок не будет. Если же была выбрана линия K, то будет приостановка согласно таблицы. Правда, компилятор обычно заменять команду Jn = <Imm> на эквивалентную команду Jn = J31 + <Imm> (NF);. Данная команда не вызывает никаких тактов приостановки.
- Запись в группу регистров модуля отладки не вызывает перезапуска конвейера.

8.2.4.1 Приостановки, обусловленные зависимостями вычислительных блоков

Эти зависимости наиболее часто встречаются в приложениях и имеют место при операциях вычислительного блока над его регистровым файлом. Вычислительный блок осуществляет доступ к регистровому файлу при выборке операнда на стадии конвейера D, использует операнд на стадии EX1, и записывает результат обратно в файл на стадии EX2. Задержка составляет в основном два такта, однако, по обходному пути результат (который записывается в конце стадии EX2) доставляется непосредственно в использующее его вычислительное устройство в конце стадии конвейера D. Вследствие этого при зависимых операциях вставляется один такт приостановки.

В таблице 40 приведен пример зависимости следующей команды от результата выполнения текущей команды. Команда #2, достигнув стадии декодирования D на 8-м такте, обнаруживает, что один из её операндов еще не готов, т.к. вычисляется командой, находящейся на стадии EX1. Команда #2 остается на стадии D еще на один такт. В 9-м такте команда #2 обнаруживает, что требуемый ей операнд доступен со стадии EX2. Конвейер продолжает движение. В результате такой зависимости на конвейере образовался так называемый «пузырь» (такт 9 стадия EX1).

Таблица 40 – Операции вычислительного блока (приостановки по зависимости от результата)

Код программы
#1: R0 = R1+R2;; /* */
#2: R5 = R0 * R3;; /* зависимость по R0 */
#3: …

Такт	IA	IM	IW	P	N	N	D	EX1	EX2
1	$0								
2		$0							

Такт	IA	IM	IW	P	N	N	D	EX1	EX2
3			$0						
4				#1					
5				#2	#1				
6					#2	#1			
7						#2	#1		
8							#2	#1	
9							#2	-	#1
10								#2	-
11									#2

Как и другие команды вычислительного блока, все команды коммуникационно-логического устройства (CLU) исполняются в вычислительном конвейере. Подобно другим вычислительным командам, все команды CLU проверяются на зависимость. Всякое использование результата предыдущей команды вызывает приостановку на один такт. Более подробно информацию о зависимостях можно узнать из таблицы 39. Пересылка данных из регистра CLU в регистровый файл вычислительного блока зависимостей не имеет. Пересылка данных выполняется на стадии EX2. Хотя регистр thr является скрытым операндом и/или результатом в командах ACS и despread, зависимость по нему отсутствует. Загрузка регистра CLU может выполняться параллельно с другими командами CLU. Код загрузки регистра CLU есть код команды сдвигового устройства.

8.2.4.2 Приостановки, обусловленные конфликтами на шинах

Шины в процессоре привязаны к конкретным ведущим на шине (J-IALU – к J-шине, K-IALU – к K-шине, устройство управления – к I-шине, а SOC-интерфейс – к S-шине) и могут получить доступ к любому блоку памяти. Если команда выполняется на линии J конвейера, она будет использовать J-шину для доступа к памяти или для пересылки регистров, или для загрузки регистров. Если команда выполняется на линии K конвейера, она будет использовать K-шину для доступа к памяти или для пересылки регистров, или для загрузки регистров. Соответственно конвейер выборки памяти использует I-шину. Если внешнее ведущее устройство выполняет обмен с внутренней памятью ядра через SOC-интерфейс, оно будет использовать S-шину. Никаких конфликтов доступа на шинах нет, т.к. хозяином на шине может быть только одно устройство. Если по J, K, I и S шинах в одном и том же такте имеются запросы к разным банкам памяти, конфликтов доступа не возникает и нет тактов приостановки. Имеются конфликты одновременного доступа к одному ресурсу ядра от разных шин. Особым случаем стоит операция доступа к регистрам ядра по чтению или записи со стороны внешнего ведущего устройства. Доступ осуществляется по шине S и всегда приводит к приостановке конвейера процессора на один такт в момент выполнения внешней операции с регистром.

Конфликт доступа возникает, когда по нескольким шинам поступают запросы к одному и тому же блоку памяти. В этом случае возникает такт приостановки. Пример такого конфликта описан в таблице 41. В данном случае линия команд #1 достигает стадии A и отправляет в один и тот же блок памяти два запроса на чтение – от J и K модулей. Память принимает все запросы и конвейер продвигается. Однако, в следующем такте (такт 8 в таблице 41) память обслужит только один запрос (J-запрос), а второй запрос (K) будет обслужен в такте 9. Это принуждает команду #2 приостановиться на стадии A на один такт (приостанавливаются все стадии конвейера от P до A на всех линиях конвейера обработки). В такте 9 на стадии W образуется «пузырь». В 10-м такте оба прочитанных значения загружаются в регистры-приемники.

Таблица 41 – Конфликт доступа при чтении блока памяти

Код программы

```
#1: XR0 = [J0+=5]; YR5 = [K0+=4];; /* J0 и K0 указывают на один банк памяти*/
#2: …
```

Такт	IA	IM	IW	P	D	X	A	M	W
1	$0								
2		$0							
3			$0						
4				#1					
5				#2	#1				
6					#2	#1			
7						#2	#1		
8							#2	#1-J	
9							#2	#1-K	-
10								#2	#1
11									#2

Такты приостановки также возникают в случае доступа к одному и тому же слову внутренней памяти, если за командой записи слова сразу же идет команда чтения того же слова. Данная зависимость связана с тем, что адрес доступа к внутренней памяти готов уже на стадии A конвейера и в случае команды чтения он тут же отправляется в память. В случае команды записи адрес готов на стадии A, но данные готовы только на стадии W. В связи с этим команда записи продвигается до стадии W и только на ней осуществляется передача адреса и данных в память. При этом данные поступают не прямо в память, а предварительно загружаются в буфер записи. Из буфера записи данные передаются в нужный блок памяти. Понятно, что пока данные не запишутся в память, чтение из данной ячейки памяти невозможно. Никаких обходных путей для данного случая не предусмотрено. Пример временной диаграммы для случая зависимости чтения после записи приведен в таблице ниже.

Таблица 42 – Конфликт доступа при чтении памяти после записи

Код программы

```
#1: [J0+0] = XR0;; /*запись слова по адресу J0 */
#2: YR5:4 = L[J0+=2];; /* чтение двойного слова по адресу J0 и J0+1 */
```

Такт	IA	IM	IW	P	D	X	A	M	W
1	$0								
2		$0							
3			$0						
4				#1					
5				#2	#1				
6					#2	#1			
7						#2	#1		
8	Команда #2 обнаруживает что данные по адресу чтения еще не записаны в память						#2	#1	
9	Команда записи #1 передает данные в буфер записи						#2		#1
10	Данные из буфера записи поступают в память						#2		
11							#2		
12								#2	
13									#2

Из примера видно, что данная зависимость порождает приостановку конвейера на 3 такта. Такая приостановка возможна только при доступе во внутреннюю память. В случае доступа во внешнюю память или SOC-шину приостановок между командами не возникает. Из примера видно, что для исключения потерь в производительности, между командой записи и чтения по одному и тому же адресу должно быть как минимум три линии команд. Более подробно конвейер обмена с внутренней памятью описан в следующей главе.

8.2.4.3 Приостановки, обусловленные зависимостями при загрузке в вычислительных блоках

Данные в командах загрузки пересылаются в регистры на стадии конвейера W (EX2), в точности, как в операциях вычислительного блока. В случае зависимости между командой загрузки и вычислительной операцией, которая использует эти данные, поведение аналогично поведению в случае зависимости вычислительного блока.

Например, следующая последовательность

```
#1: XR0 = [J31 + memory_access] ;;
#2: XR5 = R0 * R4 ;; /* Один такт приостановки */
```

Это вызывает один такт задержки, т.к. команда #2 на стадии D обнаружит, что один из операндов (R0) недоступен, и ей придется сделать такт приостановки. В следующем такте команда обнаружит необходимый ей операнд, который в этот момент пересылается из памяти в регистр. Команда 2 перехватывает нужный операнд и конвейер продолжает движение.

Если загрузка осуществляется из внешней памяти или из регистра на SOC-шине, то вторая команда продолжит движение только после того как данные будут загружены в регистр.

8.2.4.4 Приостановки, обусловленные зависимостями при загрузке в IALU

Зависимость между командами загрузки и командами IALU более сложна, чем в предыдущем случае, поскольку данные загружаются на стадии конвейера W, а используются на стадии D. Чтобы преодолеть такое запаздывание, вставляется три такта приостановки перед командой, которая использует загруженные данные, как показано в таблице 43. Команда #2 в такте 6 на стадии D обнаруживает, что один из операндов недоступен для выполнения операции. Операнд становится доступным только когда команда #1 на стадии W выполняет пересылку данных из памяти в регистр (такт 9). В этом такте команда 2 перехватывает загружаемый операнд и конвейер продолжает движение. Видно, что между первой и второй командой образуется пузырь размером в три такта.

Если загрузка осуществляется из внешней памяти или из регистра на SOC-шине, то вторая команда продолжит движение только после того как данные будут загружены в регистр.

Таблица 43 – Конфликт зависимости в IALU при загрузке данных

Код программы

```
#1: J5 = [J0+=5];;
#2: J3 = J5 + J4;;
```

Такт	IA	IM	IW	P	D	X	A	M	W
1	$0								
2		$0							
3			$0						

4				#1					
5				#2	#1				
6					#2	#1			
7					#2		#1		
8					#2			#1	
9					#2				#1
10						#2			
11							#2		
12								#2	
13									#2

8.2.4.5 Приостановки, обусловленные зависимостями при загрузке (из внешней памяти)

Комбинация любой исполнительной команды, за которой следует команда сохранения, свободна от зависимостей, поскольку данные пересылаются командой сохранения на стадии конвейера W. Единственным исключением из этого правила является сохранение данных, полученных загрузкой из внешней памяти. Например:

XR0 = [J31 + внешний_адрес] ;;

[J0+ = 0] = XR0 ;; /* приостановка до готовности XR0 */

В подобных случаях приостановка продолжается до тех пор, пока XR0 не загрузится фактически.

8.2.4.6 Приостановки, обусловленные зависимостями при условной загрузке IALU

Обычно команды IALU исполняются за единственный такт на стадии конвейера X. Результат конвейеризуется и записывается в результирующий регистр на стадии конвейера W. Если следующая команда использует результат этой (используется либо результат, либо флаг состояния), то она извлекает результат из конвейера (через обходной путь). Единственный случай, когда обход не может быть использован, связан с условным исполнением команд, когда в качестве предиката используется предикат вычислительного модуля. Пример такой ситуации показан в таблице 44. В данном случае команда 2 на стадии D обнаруживает, что необходимый ей операнд имеется на стадии X, но истинность его значения еще не установлена, т.к. в качестве предиката используется предикат вычислительного модуля. Команда 2 остается на стадии D и ждет пока команда 1 достигнет стадии W, на которой уже достоверно будет известно, откуда необходимо извлечь операнд (если истина – со стадии W, а если ложь – из РОН).

Таблица 44 – Конфликт зависимости в IALU при условном выполнении

Код программы

```
#1: IF AEQ; do, J0 = J2 + J3;;
#2: J5 = J0 + 4;;
```

Такт	IA	IM	IW	P	D	X	A	M	W
1	$0								
2		$0							
3			$0						
4				#1					
5				#2	#1				

6					#2	#1			
7					#2		#1		
8					#2			#1	
9					#2				#1
10						#2			
11							#2		
12								#2	
13									#2

Из таблицы видно, что в данном случае между первой и второй командой образуется пузырь в три такта. Отметим, что в случае использования предикатов IALU (точнее – предикатов которые обрабатываются на стадии X), никаких тактов приостановки не будет.

8.2.5 *Воздействие прерываний на конвейер*

Прерывания (и исключения) нарушают поток исполнения и вызывают конвейерные эффекты, подобные тем, которые имеют место при других типах ветвления исполнения. Типы прерываний описаны в разделе «Прерывания» Спецификации.

Отработка прерываний в некоторых приложениях критична. Следующие разделы описывают различные потоки отработки прерываний.

Обработка прерывания начинается в контроллере прерываний. Контроллер прерываний принимает запрос в регистр ILAT, далее смотрит разрешен ли полученный запрос в регистре IMASK. Если прерывание разрешено, анализируется регистр PMASK, который содержит в себе информацию о текущих обрабатываемых прерываниях. Если приоритет поступившего запроса прерывания выше приоритета обрабатываемого прерывания (либо нет обрабатываемых прерываний), контроллер формирует номер прерывания, считывает из таблицы векторов прерываний соответствующее значение адреса процедуры обслуживания прерывания и отсылает всю эту информацию в устройство управления.

Когда запрос прерывания (номер прерывания и адрес обработчика) поступает в устройство управления, первым действием процессора является проверка бита глобального разрешения прерываний GIE регистра управления SQCTL. Если бит равен 1, прерывания разрешены, и процессор выполняет следующие действия:

- Запоминается номер (N) и адрес-вектор (AV) обработки пришедшего прерывания.
- Выполняет сброс всего конвейера выборки команд вместе с буфером IAB.
- Запускает чтение команд по адресу AV.
- Конвейер обработки команд продолжает и заканчивает обработку всех команд, которые находились на конвейере в момент сброса конвейера чтения команд.
- Когда последняя команда конвейера обработки завершает свое выполнение, процессор повторно проверяет бит разрешения прерываний GIE и если прерывания по-прежнему разрешены, первая команда процедуры обработки прерывания поступает из буфера команд на конвейер обработки.
- При достижении первой командой последней стадии конвейера (W) происходит формирование сигнала о том, что данное прерывание взято на обработку. Этот факт приводит к установке бита N регистра PMASK и сброс бита N в регистре ILAT. Установленный бит в регистре PMASK запрещает все запросы прерывания уровнем ниже N+1. Вместе с этим, процессор запоминает в специальном регистре RETI адрес следующей

команды, которая должна быть выполнена при выходе из прерывания. Дополнительно ко всему, процессор устанавливает внутренний флаг EXE_ISR, который запрещает все прерывания. Флаг EXE_ISR указывает на то, что процессор находится в состоянии обработки прерывания. Флаг доступен для чтения как 20-й бит регистра SQSTAT.

- Переход на обработку прерывания вызывает установку режима супервизора (если до этого был режим пользователя).
- Сброс бита N в регистре ILAT означает, что бит стал доступен для приема нового запроса прерывания (если прерывание чувствительно к фронту сигнала).
- Далее процессор выполняет все команды, которые соответствуют процедуре обслуживания данного прерывания. Для выхода из процедуры обработки используется команда RTI.
- Выполнение команды RTI приводит к сбросу бита N в регистре PMASK, что разрешает прохождение всех прерываний, которые ранее блокировались. Также данная команда вызывает сброс флага EXE_ISR, что снимает блокировку прерываний и включает функции бита GIE. Процессор переходит к выполнению команд по адресу из регистра RETI.

Действия на конвейере, которые происходят при обработке прерывания, описаны в таблице ниже.

Таблица 45 – Обработка прерывания

Код программы
#1: команда 1;; // программа пользователя по адресу 0
#2: команда 2;;
#3:
#i1: команда 1;; // обработчик прерывания по адресу AV
#i2:

Такт	**IA**	**IM**	**IW**	**P**	**D**	**X**	**A**	**M**	**W**
1	$0								
2	$4	$0							
3	$8	$4	$0						
4	$12	$8	$4	#1					
5	$16	$12	$8	#2	#1				
К 5-му такту процессор получил запрос прерывания, проверил бит GIE и готов начать обработку. Одновременно с передачей команды **#**2 на стадию D, процессор производит сброс конвейера выборки команд и всего содержимого буфера IAB									
6	$AV				#2	#1			
7	$AV+4	$AV				#2	#1		
8	$AV+8	$AV+4	$AV				#2	#1	
9	$AV+12	$AV+8	$AV+4	#i1				#2	#1
10	$AV+16	$AV+12	$AV+8	#i1					#2
11				#i1					
12				#i2	#i1				
13					#i2	#i1			
14						#i2	#i1		
15							#i2	#i1	
16								#i2	#i1
17									#i2

Из данного примера видно, что все команды, которые поступили на конвейер обработки, завершают свое выполнения до начала поступления команд процедуры обработки прерывания на конвейер обработки. Пример из таблицы 44 соответствует идеальной ситуации, когда при выборке команд обработчика прерывания из памяти *нет конфликтов доступа*, а также *нет приостановок* на конвейере обработки команд. По сути потери в реакции на прерывание составляют два такта (такты 9 и 10) когда обработчик прерывания ожидает завершения всех команд пользователя. Общими потерями можно считать образования пузыря из пяти тактов.

В такте 11 (Таблица 45) конвейер обработки свободен и процессор еще раз проверяет все условия для начала исполнения кода обработчика (команды пользователя могут запретить прерывания или вызвать сброс всей информации на конвейере).

Если считать время реакции на прерывание как время от момента установки бита в регистре ILAT, до момента завершения выполнения первой команды обработчика прерывания, то это время может составить минимум 15 тактов ядра. Именно через 15 тактов команда #i1 покинет стадию W конвейера и затем произойдет сброс бита запроса прерывания в ILAT, а также установка бита обслуживания прерывания в PMASK. Из этих 15 тактов затраты на конвейер процессора составляют 9 тактов на стадиях конвейера + 2 такта ожидания освобождения конвейера. Оставшиеся такты используются для анализа прерывания в контроллере прерываний, выборки вектора прерывания и передачи информации в устройство управления.

Отметим, что вышеописанная процедура соответствует типичной ситуации обработки при отсутствии каких-то аномалий. Вместе с тем может быть множество нюансов, например:

- Если перед повторной проверкой разрешения прерываний (после полного освобождения конвейера обработки) оказывается, что прерывания запрещены (это может быть сделано командой сбрасывающей бит GIE), тогда все действия по обработке прерывания отменяются и конвейер перезапускается с адреса, следующего за адресом последней выполненной команды (в приведенном примере это будет команда #3).
- Если запрос прерывания из контроллера пришел в процессор, но бит GIE равен нулю, прерывание будет ждать. В это время в контроллере прерываний может появиться запрос с более высоким приоритетом чем N. Контроллер прерываний обработает его и передаст в процессор номер и адрес-вектор нового прерывания.
- Возможна ситуация, когда запрос пришел из контроллера, но в это же время процессор выполнял сброс бита запроса прерывания в регистре ILAT. Эта ситуация приведет к тому что запрос из контроллера прерываний будет стабильным в течение одного или нескольких тактов. Тем не менее, в такой ситуации запрос может быть обработан. Если снятие запроса произошло после или во время сброса конвейера чтения команд (в 5-м такте примера или позже), то это означает, что запрос будет обслужен. Для надежного выполнения подобных операций необходимо предварительно запрещать прерывания битом GIE, а затем производить сброс в контроллере прерываний.

Одновременно с выполнением первой команды обработки прерывания, процессор сохраняет в регистре RETI адрес возврата из прерывания. В нашем примере это будет адрес команды #3. Обычно процедура обработки прерывания начинается с сохранения контекста процессора, т.е. сохранения на стеке всех регистров процессора, которые будут использованы программой обработки прерывания. Все это делается для того чтобы после обработки прерывания восстановить прежнее состояние процессора и дать возможность прерванной

программе корректно возобновить работу. Напомним, что факт обработки прерывания отражается в установке флага EXE_ISR и это запрещает все прерывания, в том числе и более высокоприоритетные чем наше. Если время обработки прерывания значительно и это может повредить обработке запроса более приоритетного прерывания, программа может разрешить обслуживание более высокоприоритетных прерываний. Для этого необходимо обязательно сохранить на стеке содержимое регистра RETI, который, как упоминалось ранее, хранит адрес возврата из прерывания. Для сохранения содержимого RETI используется специальный псевдоним этого регистра – RETIB. Выполнение команды j0 = RETIB;; вызовет копирование регистра RETI в регистр j0 и сброс флага EXE_ISR. Понятно, что сам регистр j0 должен быть до этого сохранен на стеке. После выполнения указанной команды процессор имеет возможность обслуживать более высокоприоритетные запросы, чем запрос N. Описываемая ситуация вносит коррективы в последовательность действий при выходе из обработки прерывания. Понятно, что перед выходом мы имеем ситуацию, когда содержимое регистра возврата RETI неопределенно (новые прерывания могли изменить его). Поэтому мы должны взять наш адрес возврата из регистра j0 (или другого места где он хранился) и вернуть обратно в RETI. Однако. выполнение обычной пересылки RETI = j0;; будет некорректным, т.к. после этой команды возможно возникновение прерывания и мы можем потерять адрес возврата. Поэтому нам на помощь опять приходит регистр-псевдоним RETIB. Пересылка RETIB = j0;; не только пересылает адрес возврата в RETI, но и устанавливает флаг EXE_ISR. А это запрещает прерывания и мы можем безопасно восстановить весь контекст, а затем выполнить финишную команду RTI.

Остановимся подробнее на команде выхода из прерывания, т.к. правильное её использование позволяет ускорить выход из прерывания. В таблице 45 показан один из вариантов выхода из прерывания. Отметим, что независимо от предиката, команда RTI выполняет переход на последней стадии конвейера. Если команда RTI используется с опцией (NP), мы всегда будем иметь потери на переход равными 8-ми тактов, т.к. при выполнении команды происходит очистка всего конвейера процессора. Однако. потери могут быть связаны еще и с тем, что регистр RETI непосредственно загружается перед командой RTI. При использовании предсказания перехода, адрес перехода формируется на стадии D. Поэтому процессор «привычно» ждет, когда регистр RETI загрузится. В данном случае это приводит к 4-м тактам приостановки. Поэтому в приложениях желательна более ранняя загрузка регистра RETI (если она нужна).

Таблица 46 – Выход из прерывания. Вариант 1

Код программы

```
// программа пользователя по адресу 0
#1: RETI = [address];;
#2: RTI (ABS) (NP);;
#3:
```

Такт	IA	IM	IW	P	D	X	A	M	W
1	$0								
2		$0							
3			$0						
4				#1					
5				#2	#1				
6					#2	#1			
7					#2		#1		
8					#2			#1	

Такт	IA	IM	IW	P	D	X	A	M	W
9					#2				#1
10					#2				
11	($RETI*)					#2			
12							#2		
13								#2	
14									#2
15	$RETI (NP)								
16		$RETI							
17			$RETI						
18				#u1					
19					#u1				
20						#u1			
21							#u1		
22								#u1	
23									#u1

В описанном выше примере переход использовался с опцией (NP), т.е. не предпренимаемый.

Что произойдет, если мы не будем использовать опцию (NP) и наш переход будет верно рассматриваться как предпринимаемый? Здесь нужно рассматривать уже две ситуации: случай первого исполнения команды, когда в BTB нет информации о переходе и случай повторного исполнения кода, когда информация может храниться в BTB.

В первом случае команда RTI видит, что имеется промах в BTB, дождется в 10-м такте загрузки регистра RETI и выполнит переход. В этом случае поток на конвейере, который начинается с 15-го такта, начнется с 11-го (отмечено RETI*). Итого, мы ускорим выход из прерывания на 4 такта. Если мы отодвинем загрузку RETI на расстояние как минимум в 4 линии команд от RTI, то мы еще сэкономим 4 такта. Остаются еще 4 такта.

Теперь рассмотрим, что же произойдет при повторной обработке прерывания в случае, когда информация о выходе из прерывания находится в BTB. В этом случае при выборке квадрослова с командой RTI сработает предсказание и процессор сразу перенаправит поток по адресу RETI. Все было бы просто идеально, если бы значение регистра RETI в момент предсказания было достоверно. Что нужно сделать чтобы значение RETI действительно было достоверно? Если адрес возврата из прерывания не сохранялся в памяти или в каком-то регистре ядра, то проблемы нет. Если же он сохранялся, то нам необходимо выполнить его загрузку обратно. И это должно быть сделано до момента предсказания, иначе предсказание будет ошибочно, и мы потеряем много тактов. На каком же расстоянии до команды RTI должна располагаться команда загрузки RETI? Рассмотрим случай, когда все линии команд состоят из 4-х команд, т.е. каждое квадрослово содержит одну линию команд. Пример приведен в таблице ниже.

Таблица 47 – Выход из прерывания. Вариант 2

Код программы

```
// программа пользователя по адресу 0
$0: #1: xr0 = [external_address];NOP; NOP; NOP;;
$4: #2: RETI = [address];NOP; NOP; NOP;;
$8: #3: NOP; NOP; NOP; NOP;;
```

$12: #4: NOP; NOP; NOP; NOP;;
$16: #5: NOP; NOP; NOP; NOP;;
$20: #6: NOP; NOP; NOP; NOP;;
$24: #7: NOP; NOP; NOP; NOP;;
$28: #8: NOP; NOP; NOP; NOP;;
$32: #9: NOP; NOP; NOP; NOP;;
$36: #10: NOP; NOP; NOP; NOP;;
$40: #11: NOP; NOP; NOP; NOP;;
$44: #12: RTI (ABS);NOP; NOP; NOP;;

Такт	IA	IM	IW	P	D	X	A	M	W
1	$0								
2	$4	$0							
3	$8	$4	$0						
4	$12	$8	$4	#1					
5	$16	$12	$8	#2	#1				
6	$20	$16	$12	#3	#2	#1			
7	$24	$20	$16	#4	#3	#2	#1		
8	$28	$24	$20	#5	#4	#3	#2	#1	
9	$32	$28	$24	#6	#5	#4	#3	#2	#1
11	$36	$32	$28	#7	#6	#5	#4	#3	#2
12	$40	$36	$32	#7	#6	#5	#4	#3	#2
13		$40	$36	#7	#6	#5	#4	#3	#2
14			$40	#7	#6	#5	#4	#3	#2
15				#7	#6	#5	#4	#3	#2
Процессор в состоянии ожидания данных из внешней памяти. Буфер IAB заполнен. В такте К будет произведена загрузка данных в регистр XR0									
К+1				#7	#6	#5	#4	#3	#2
К+2	$44*			#8	#7	#6	#5	#4	#3

Из примера видно, что для надежного предсказания загрузка RETI должна опережать момент предсказания на 9 квадрослов, т.е. если загрузка в 1-м квадрослове, то команда RTI должна быть в 11-м. Такой вариант можно назвать самым надежным, т.к. если в одном квадрослове будет более одной линии команд, то буфер IAB будет заполнен раньше и процессор не будет так быстро загружать новые квадрослова. Из указанных девяти, 5 квадрослов соответствуют стадиям конвейера W, M, A, X, D. Это наихудший случай, когда линия команд состоит из 4 слов. В лучшем случае это может быть 5 слов, и тогда на конвейере исполнения будет находиться только одно квадрослово. Четыре из 9-ти квадрослов соответствуют конвейеру выборки команд. Стадии P, IW, IM, IA. Таким образом, процесс корректного предсказания носит случайный характер.

Что же будет происходить в процессоре в случае некорректного предсказания перехода по регистру RETI (все это справедливо и для регистра CJMP)? Пример такой ситуации приведен в таблице 48. Здесь мы видим, что в такте 1 происходит переход по неизвестному значению в RETI. На стадии D команда RTI видит, что было попадание в BTB и что загружается новое значение RETI. Команда ждет пока регистр загрузится (если регистр уже загружен, то тактов ожидания не будет). Далее команда RTI желает

удостовериться, по правильному ли адресу было предсказание, и сравнивает значение адреса предсказания BTB с текущим содержимым RETI. Если адреса совпадают, то предсказание было сделано верно. Если же адреса не совпадают (как в нашем примере), то ход программы будет скорректирован.

Таблица 48 – Выход из прерывания. Вариант 3

Код программы

```
// программа пользователя по адресу 0
#1: RETI = [address];;
#2: RTI (ABS);;
#3:
#4:
```

Такт	IA	IM	IW	P	D	X	A	M	W
1	$0	В этом такте срабатывает предсказание и выполняется переход по неизвестному значению в RETI							
2	$RETI?	$0							
3		$RETI?	$0						
4			$RETI?	#1					
5				#2	#1				
6					#2	#1			
7					#2		#1		
8					#2			#1	
9					#2				#1
10					#2				
11						#2			
12							#2		
13								#2	
14									#2
15	$RETI								
16		$RETI							
17			$RETI						
18				#u1					
19					#u1				
20						#u1			
21							#u1		
22								#u1	
23									#u1

Отметим опасность использования команды RTI с предсказанием перехода. При ложном срабатывании перехода процессор начинает выборку команды из неизвестного адреса в RETI. Нужно гарантировать, что в RETI не будет «мусора» и значение адреса будет всегда указывать на внутреннюю память.

8.2.5.1 Прерывание во время исполнения условной команды

Из примера в таблице видно, что процедура обработки прерывания ожидает выполнения всех команд, переданных на конвейер обработки. Таких команд может быть не более пяти. Среди этих команд могут быть команды различные команды. Что будет происходить, если среди команд будет команда перехода?

В случае, если это условный (безусловный) переход и переход выполняется, то все команды до команды перехода и включая команду перехода корректно выполнятся, а команды, следующие за переходом, будут сброшены. Адресом возврата из прерывания в данном случае будет адрес перехода. Команда перехода в этой ситуации может ускорить реакцию на прерывание (если переход быстрее очистит конвейер). Если переход не выполняется, то это равносильно команде NOP.

8.2.5.2 Прерывание во время исполнения команды запрещения прерываний

Иногда программисту требуется, чтобы определенная часть кода исполнялась без прерываний. В этом случае запрещение всех аппаратных прерываний путем сброса бита глобального разрешения прерываний (GIE) в регистре SQCTL влечет немедленный эффект (в отличие от сброса бита разрешения прерывания отдельного типа). Следует осознавать, что за использование этой возможности приходится платить ценой потери производительности. Всякий раз, когда происходит запись в регистр SQCTL, процессор очищает все стадии конвейера и начинает повторное извлечение команды следующей за командой записи в SQCTL. Это вызывает образование пузыря в 9 тактов. Если команда, сбрасывающая бит GIE, оказывается в числе тех команд, которые успели попасть на конвейер обработки до начала выборки команд процедуры обслуживания прерывания, то в следующем за записью в SQCTL такте произойдет очистка всего конвейера, включая и команды обработчика прерывания. Процессор восстановит выполнение программы, начиная с команды следующей за записью в SQCTL.

8.2.6 *Воздействие исключений на конвейер*

Исключение обычно вызывается конкретной командной строкой. Источником некоторых исключений является состояние вычислительного блока (переполнение, потеря точности и недопустимая операция с плавающей точкой), в то время как для других источником является обстоятельства, не связанные с вычислительным блоком (недопустимый доступ к памяти, операции эмуляции и др.). Следующей исполняемой командой за командой, вызвавшей исключение, является первая команда процедуры обработки исключения. Адрес процедуры обработки исключения всегда берется из регистра IVSW. Пример обработки исключения показан в таблице 49 (AV=IVSW).

Таблица 49 – Обработка исключения

Код программы
// программа пользователя по адресу 0
#1: команда 1;; // данная команда вызывает исключительную ситуацию
#2: команда 2;;
#3:

Такт	IA	IM	IW	P	D	X	A	M	W
1	$0								
2	$4	$0							
3	$8	$4	$0						
4	$12	$8	$4	#1					
5	$16	$12	$8	#2	#1				
6					#2	#1			
7						#2	#1		
8							#2	#1	

9								#2	#1
10	В этом такте происходит сброс всех стадий конвейера (включая W)								#2
11	$AV								
12	$AV+4	$AV							
13	$AV+8	$AV+4	$AV						
14	$AV+12	$AV+8	$AV+4	#i1					
15	$AV+16	$AV+12	$AV+8	#i2	#i1				
16					#i2	#i1			
17						#i2	#i1		
							#i2	#i1	
								#i2	#i1
									#i2

Необходимо отметить, что команда #2 хоть и достигла последней стадии конвейера, но все её результаты будут сброшены. Адрес возврата будет указывать на команду #2.

8.2.7 *Использование регистра CJMP в командах перехода*

Регистр CJMP выполняет функцию регистра связи при вызове подпрограмм и может быть использован для организации переходов и вызовов по вычисляемым адресам. В связи с активным использованием регистра CJMP, рассмотрим детально особенности, которые возникают при его использовании в командах перехода. Поскольку регистр участвует в формировании адреса перехода, то важно знать на какой стадии конвейера используется значение регистра. Адрес любого возможного перехода вычисляется на стадии декодирования D. В случае использования счетчика команд PC и смещения в коде команды, значение счетчика и смещения всегда готово и истинно на стадии D. В случае же использования регистра CJMP, значение регистра может быть еще не готовым для команды ветвления, находящейся на стадии D, т.к. на последующих стадиях (X, A, M, W) могут находиться команды загрузки в регистр CJMP. Это могут быть команды загрузки регистра из памяти, команды пересылки регистра в регистр, команды загрузки константы в регистр, команда IALU с опцией (CJMP), команда вызова CALL, которая сохраняет адрес возврата в CJMP. В связи с этим, команда перехода, использующая регистр CJMP, на стадии D анализирует все последующие стадии на предмет модификации регистра CJMP. Если таких модификаций не видно, команда перехода берет содержимое CJMP и вычисляет адрес перехода. Если же команда перехода видит впереди себя одну из команд модифицирующую CJMP, команда перехода будет ждать, пока все загрузки в CJMP не будут выполнены. Ожидание может достигать не более 4-х тактов.

Таким образом, переход типа

```
CJMP = J0;;
CJMP(ABS)(NP);;
```

потратит 4 такта на ожидание загрузки регистра CJMP и еще 5 тактов на выполнение перехода, т.к. переход был непрогнозируемый. В случае использования прогнозируемых переходов (без опции NP), возможно большее многообразие ситуаций. Это связано с тем, имели ли мы попадание в буфер BTB или нет, а также было ли значение CJMP достоверно в момент предсказания перехода. Рассмотрим этот же пример, но без опции NP.

CJMP = J0;;
CJMP(ABS);;

В этом случае команда перехода будет ожидать те же 4 такта, что и раньше. Однако, переход будут выполнен на стадии D, а не на стадии X, и таким образом мы сэкономим один такт на переходе, т.е. будем иметь потери на переход 4+4. Предположим, что тот же код выполняется повторно, и мы имели попадание в буфер BTB. В этом случае команда перехода на стадии D видит, что переход по регистру CJMP и было выполнено предсказание перехода по какому-то адресу. Адрес предсказания команда перехода имеет. Одновременно с этим команда перехода видит, что на конвейере выполняется загрузка в CJMP. Происходит ожидание загрузки (4 такта) и затем выполняется сравнение предсказанного адреса перехода и адреса в CJMP. Если значения совпадают, то принимается решение что переход был предсказан верно и больше потерь на переход не будет. Если же значения не совпали, принимается решение о ложном предсказании и на стадии X будет выполнен переход по корректному адресу. В этом случае мы будем иметь дополнительно 5 тактов потерь. Как сделать так чтобы тактов потерь не было вообще? Очевидно, что нужно как минимум отодвинуть модификацию регистра CJMP на расстояние 4-х линий команд от команды перехода, использующей CJMP.

В примере:
CJMP = J0;;
Линия команд W;;
Линия команд M;;
Линия команд A;;
Линия команд X;;
CJMP(ABS);;

не будет потерь на ожидание загрузки регистра, т.к. в этом случае регистр готов. Осталось еще одна проблема – корректное предсказание перехода модулем BTB. Исправить данную проблему куда сложнее чем первую. Дело в том, что предсказание перехода выполняется на стадии IA конвейера выборки команд и может происходить намного раньше, чем команда перехода достигнет стадии D. В примере с обработкой прерываний и использованием регистра RETI было отмечено, что корректное предсказание перехода по регистру RETI может быть только если модификация этого регистра и момент предсказания находятся на расстоянии не менее 9-ти квадрослов команд. Это же справедливо и для регистра CJMP. Рассмотрим пример, который показывает насколько медленным может быть выполнение программы, если предсказание перехода будет некорректным.

Пример кода:

```
lc0 = 100;;
.align 4;
DO_CALL:
 CALL INC_J;NOP;NOP;NOP;; // адрес A
 CALL INC_K;NOP;NOP;NOP;; // адрес B
 IF NLC0E, JUMP DO_CALL;; // адрес C
.align 4;
INC_J: CJMP (ABS); j5 = j5+1;; // адрес H
.align 4;
INC_K: CJMP (ABS); k5 = k5+1;; // адрес M
```

В этом примере пользователь использует вызов двух коротких подпрограмм в цикле. Поскольку все переходы с предсказанием и все адреса в памяти корректно выровнены и не мешают выполнению предсказания, то логично ожидать, что цикл выполнится приблизительно (отбросим потери от входа в цикл и выхода из цикла) за 500 тактов ((3+2)*100). На самом деле этот цикл выполнится приблизительно за 2300 тактов, т.е. вместо 5-ти предполагаемых тактов на один цикл мы будем иметь 23. Все

проблемы из-за невозможности корректного предсказания перехода. Пример последовательности обработки команд на конвейере приведен в таблице 50. В данном примере мы видим, как из-за модификации регистра CJMP при вызове двух функций происходит предсказание по ложному адресу, и как всякий раз процессору приходится восстановлять корректный ход программы. При этом добавляются ожидания завершения команды вызова из-за необходимости использования регистра CJMP. Если бы в этом примере в цикле присутствовал только один вызов (например, call inc_J), то мы имели бы в среднем 7 тактов на цикл.

Таблица 50 – Использование CJMP- переходов

Такт	**IA**	**IM**	**IW**	**P**	**D**	**X**	**A**	**M**	**W**
1	$A								
2		$A							
3			$A						
4				#A					
5					#A BTB set				
6	$H					#A			
7		$H					#A		
8			$H					#A	
9				#H					#A cjmp= B
10					#H BTB set				
11	$B					#H			
12		$B					#H		
13			$B					#H	
14				#B					#H
15					#B BTB set				
16	$M					#B			
17		$M					#B		
18			$M					#B	
19				#M					#B cjmp =C
20					#M BTB set				
21	$C					#M			
22		$C					#M		
23			$C					#M	
24				#C					#M

Такт	IA	IM	IW	P	D	X	A	M	W
25					#C BTB set				
После 25-го такта мы завершили первый проход цикла и все команды перехода с предсказанием записались в буфер BTB. Далее буфер предсказания будет вмешиваться в последовательность выборки команд. Корректно предсказанный адрес будет выделяться зеленым цветом, а некорректный – красным									
26	$A								
27	$H	$A							
28	$C	$H	$A						
29		$C	$H	#A					
30			$C	#H	#A				
31				#C	#H	#A			
32				#C	#H		#A		
33				#C	#H			#A	
34				#C	#H				#A cjmp= B
35				#C	#H				
36					#C	#H			
37	$B								
38	$M	$B							
39	$B	$M	$B						
40		$B	$M	#B					
41			$B	#M	#B				
42				#B	#M	#B			
43				#B	#M		#B		
44				#B	#M			#B	
45				#B	#M				#B cjmp =C
46				#B	#M				
47					#B	#M			
48	$C								
49	$A	$C							
50	$H	$A	$C						
51	$C	$H	$A	#C					
52		$C	$H	#A	#C				
53			$C	#H	#A				
54				#C	#H	#A			
55				#C	#H		#A		
56				#C	#H			#A	
57				#C	#H				#A cjmp= B
58				#C	#H				

Такт	IA	IM	IW	P	D	X	A	M	W
59					#C	#H			
60	$B								
Далее процесс будет повторяться, начиная с такта 37 и заканчивая тактом 59, т.е цикл будет состоять из 23 тактов									

8.3 Примеры программирования устройства управления

В листингах этого раздела содержатся примеры использования команд устройства управления. Комментарии, сопровождающие команды, отмечают ключевые особенности команд, например, для команды перехода указывается, является ли переход прогнозируемым или нет, или что команда участвует в организации цикла и др.

Листинг 34 - Команды устройства управления, поля и строки

```
IF XAEQ, JUMP Label_INP;;
/* Это командная строка с единственной командой, которая содержит одно
«поле» или 32-битное слово */ NOP; NOP; NOP; NOP;;
IF XMEQ; DO, XR3:2 = R5:4 + R7:6;;
/* Это командная строка, состоящая из двух команд, причем первая представляет
собой предикат XMEQ, а вторая является сложением в вычислительном блоке X */
```

Листинг 35 - Пример команды JUMP

```
.SECTION program ;
CALL test ;;  /* Адрес:0x0; */
/* Переход на 0x3. Сохраняет 0x1 в регистре CJMP */
NOP ;; /* Адрес: 0x1 */
endhere:
JUMP endhere ;; /* Адрес: 0x2 */
test:
NOP ;; /* Адрес: 0x3 */
CJMP (ABS) ;;/* Адрес: 0x4; */
/* Конец подпрограммы. Переходит обратно на Адрес 0x1. */
```

Листинг 36 - Пример команды CJMP CALL

```
.SECTION program;
J0 = ADDRESS(endhere) ;; /* Адрес:0x0 */
CJMP = ADDRESS(test) ;;  /* Адрес: 0x1 */
/* Предзагрузка регистра   cjmp значением 0x6 */
CJMP_CALL (ABS) ;;                  /* Адрес: 0x2 */
/* Переход по адресу, хранимому в регистре CJMP (0x6)
/* Заносит в регистр CJMP новое значение 0x3. */
NOP ;;/* Адрес: 0x3 */
endhere:
NOP ;;/* Адрес: 0x4 */
JUMP endhere ;;      /* Адрес: 0x5 */ test:
J31 = J0 + J31 (CJMP) ;;    /* Адрес: 0x6 */
/* Использует опцию (CJMP) сложения IALU */
/* Загружает результат (0x4) в регистр CJMP. */
CJMP (ABS) ;; /* Адрес: 0x7 */
/* Конец подпрограммы. Переходит обратно на 0x4. */
```

Листинг 37 - Пример циклов

```
J6 = J31 + 10;; /* загрузка счетчика для _outer_loop (внешнего цикла) */
.align_code 4;
_outer_loop:
NOP ;; /* вход в _outer_loop (во внешний цикл) */
NOP ;;
LC1 = 5;; /* загрузка счетчика для _middle_loop (среднего цикла)*/
.align_code 4;
_middle_loop:
NOP ;; /* вход в _middle_loop (средний цикл) */
NOP ;;
LC0 = 6;; /* загрузка счетчика для _inner_loop (внутреннего цикла) */

.align_code 4;
_inner_loop:
NOP ;; /* вход в _ inner_loop (внутренний цикл) */
NOP ;NOP ; NOP ; NOP ;;
NOP ;NOP ; NOP ; NOP ;;
NOP ;NOP ; NOP ; NOP ;;
NOP ;NOP ; NOP ; NOP ;;
NOP ;NOP ; NOP ; NOP ;;
. align_code 4;

IF NLC0E, JUMP _inner_loop ; NOP ; NOP ; NOP ;;
/* проверка/выход из _inner_loop (внутреннего цикла)
NOP ;NOP ; NOP ; NOP ;

. align_code 4;
NOP ;NOP ; NOP ; NOP ;;
NOP ;NOP ; NOP ; NOP ;;
NOP ;NOP ; NOP ; NOP ;;
NOP ;NOP ; NOP ; NOP ;;
NOP ;NOP ; NOP ; NOP ;;

align_code 4;
IF NLC1E, JUMP _middle_loop; NOP ; NOP ; NOP ;;
/* проверка/выход из _middle_loop (среднего цикла) */
NOP ; NOP ; NOP; NOP ;;

.align_code 4;
J6 = J6 - 1; NOP ; NOP ; NOP ;;
/* декрементирование        счетчика для _outer_loop (внешнего цикла)
*/ If NJEQ, JUMP _outer_loop; NOP ; NOP ; NOP ;;
/* Конец внешнего цикла */
```

Листинг 38 - Использование буфера целевых адресов BTB (без прогнозирования переходов)

```
LC0 = 100;; /* Инициализация счетчика цикла */
.align_code 4;
_loop:
NOP;NOP;NOP;NOP;;
NOP;NOP;NOP;NOP;;
```

```
NOP;NOP;NOP;NOP;;
NOP;NOP;NOP;NOP;;
NOP;NOP;NOP;NOP;;
NOP;NOP;NOP;NOP;;
If NLC0E, JUMP _loop (NP); NOP;; /* Конец цикла */
```

Цикл на Листинге 38 исполняется за 1+99(7+5)+1*7= 1196 тактов. Первые 99 итераций цикла приводят к 5 тактам потерь из-за переходов на начало цикла. Последняя итерация не приводит к потерям из-за NP.

Листинг 39 - Использование буфера целевых адресов BTB (с прогнозированием переходов)

```
LC0 = 100;; /* Инициализация счетчика цикла */ lign_code 4;
_loop: nop;nop;nop;nop;; nop;nop;nop; nop;; nop;nop;nop;nop;; nop;nop;nop; nop;;
nop;nop;nop; nop;; nop;nop;nop;nop;;
If NLCOE, JUMP _loop ; nop;; /* Конец цикла */
```

В первый раз, когда исполняется программа, приведенная в Листинге 39, или тогда, когда в BTB еще не помещен адрес _loop, число тактов исполнения равно 1+1*(7+4)+99*7 + 5=710. Поскольку BTB не загружен, первая итерация цикла даст 4 такта потерь. Затем остальные 99 итераций исполняются за 7 тактов без потерь. Выход из цикла даст 5 тактов потерь. Когда программа, приведенная в Листинге 39, исполняется во второй раз, в BTB изначально загружен адрес _loop. Число тактов теперь равно 1+100*7 + 5=706.

8.4 Сводка команд устройства управления

В листинге 40 показан синтаксис команд устройства управления. Соглашения по обозначениям имён регистров, необязательных элементов или вариантов, используемые в этих листингах, подробно освещены в разделе «Регистры регистровых файлов». Вкратце эти соглашения таковы:

- {} - необязательные элементы указаны в фигурных скобках; фигурные скобки не являются частью синтаксиса команды.
- | - вертикальная черта разделяет варианты значений элементов; вертикальные черты не являются частью синтаксиса команды.
- Label - шаблон задаваемой пользователем программной метки (выделяется курсивом). При использовании Label в программе в качестве ссылки преобразуется ассемблером либо в 15- или 32-битный PC-относительный адрес, либо в 32-битный абсолютный адрес. Замена на 15-битный адрес (если это удаётся) производится в случае, когда метка Label и команда перехода не неё содержатся в одной и той же программной секции (.section). Замена на 32-битный адрес производится в случае, когда метка и команда перехода содержатся в разных секциях. Дополнительная информация об относительных и абсолютных адресах, а также о переходах содержится в разделе «Отработка переходов».

PC-относительная адресация и прогнозирование для переходов и вызовов используется по умолчанию. Для использования абсолютной адресации используйте опцию (ABS). Для указания того, что переход не является прогнозируемым, используйте опцию (NP).

Для условных команд вычислительного блока Condition является одним из следующих предикатов: AEQ, ALT, ALE, MEQ, MLT, MLE, SEQ, SLT, SF0 или SF1. Чтобы указать для предиката, к какому конкретному вычислительному блоку он относится, снабдите его префиксом X, Y или XY, как в XAEQ. Отсутствие префикса

интерпретируется как XY. Чтобы получить отрицание предиката, снабдите его префиксом N, как в NXAEQ.

Предикаты AEQ, ALT и ALE суть ALU-предикаты «равно нулю», «меньше нуля» и «меньше или равно нулю». Предикаты MEQ, MLT и MLE суть предикаты умножителя «равно нулю», «меньше нуля» и «меньше или равно нулю». Предикаты SEQ и SLT суть предикаты сдвигового устройства «равно нулю» и «меньше нуля». Предикаты SF0 и SF1 суть предикаты статических флагов 0 и 1 вычислительного блока.

Для условных команд IALU Condition является одним из следующих предикатов: EQ, LT, LE, ISF0 или ISF1. Чтобы указать для предиката, к какому конкретному IALU он относится, снабдите его префиксом J или K; ISF0 и ISF1 могут содержать предикаты любого IALU. Чтобы получить отрицание предиката, снабдите его префиксом N, как в NJEQ.

Предикаты JEQ, JLT и JLE суть J-IALU-предикаты «равно нулю», «меньше нуля» и «меньше или равно нулю» и аналогично для K-IALU. Предикаты ISF0 и ISF1 суть предикаты глобальных статических флагов IALU.

Для условных команд устройства управления Condition может быть любым предикатом вычислительного блока, IALU-предикатом или одним из предикатов устройства управления: BM, LC0E, LC1E, FLAG0_IN, FLAG1_IN, FLAG2_IN или FLAG3_IN. Чтобы получить отрицание предиката, снабдите его префиксом N, как в NBM.

Предикат BM есть предикат «ведущий на шине». Предикаты LC0E и LC1E являются предикатами «счетчик цикла 0 равен нулю» и «счетчик цикла 1 равен нулю». Предикаты от FLAG0_IN до FLAG3_IN суть предикаты равенства единице состояния входных дискретных сигналов.

Пример команды устройства управления, использующей предикат вычислительного блока:

IF NXMLT, JUMP label_10;;

Эта команда выполняет переход по адресу, который соответствует метке label_10 в случае, если вычисленное значение предиката nxmlt (не, вычислительный блок X, умножитель меньше, чем ноль) равно TRUE.

Команды, которые следуют за условной командой или за переходом/вызовом в строке, могут иметь ключевое слово DO|ELSE или не иметь префикса вообще. Отсутствие префикса указывает на то, что исполняется команда вне зависимости от значения предиката. Ключевое слово DO относится только к условной команде, в то время как ключевое слово else относится только к условному переходу.

Команды NOP, IDLE, BTBINV, BTBEN, BTBDIS, BTBLOCK, BTBELOCK, TRAP И EMUTRAP не могут быть условными. К примеру, следующая командная строка являются недопустимой:

if aeq; do idle;; /* НЕДОПУСТИМО! */

Листинг 40 - Команды устройства управления

```
{IF Condition,} JUMP| CALL <Label> {(NP)} { (ABS) } ;
{IF Condition,} CJMP |CJMP_CALL { (NP) } {(ABS)} ;
{IF Condition,} RETI|RTI { (NP) } {(ABS)} ;
{IF Condition,} RDS ;
IF Condition;
DO, команда; DO, команда; DO, команда;;
/* Такой синтаксис позволяет подчинить одному предикату до трёх условных команд.
Пропуск DO перед командой делает её безусловной. */
IF Condition, JUMP|CALL|CJMP|CJMP_CALL ; ELSE,
команда; ELSE, команда; ELSE, команда;;
/* Такой синтаксис позволяет подчинить одному предикату до трёх условных команд.
Пропуск ELSE перед командой делает её безусловной.*/
```

{X|Y|XY}SF1|SF0 = Выч.предикат ;
{X|Y|XY}SF1|SF0 += AND|OR|XOR Выч.предикат ;
ISF1|ISF0 = IALU-предикат Выч.предикат | УУ-предикат ;
ISF1|ISF0 += AND|OR|XOR IALU-предикат Выч.предикат | УУ-предикат ;
IDLE ;
IRQLOCK:
IRQELOCK;
BTBEN ;
BTBDIS ;
BTBLOCK ;
BTBELOCK ;
BTBINV ;
TRAP (<Imm5>) ;;
EMUTRAP ;; NOP;

9 Память и шины

Ядро процессора содержит большую внутреннюю память (встроенную SRAM) и обеспечивает доступ к внешней памяти через внешний порт процессора. В этой главе описывается внутренняя память процессора и показывается, какта может быть использована в программах. Информацию о подсоединении внешней памяти и временных параметрах доступа к ней см. в главах «Кластерная шина» и «Интерфейс SDRAM» Спецификации процессора.

В DSP-процессоре использована так называемая супергарвардская архитектура. В этой архитектуре имеются шины программ и данных, но поддерживается единое адресное пространство для хранения программ и данных. В состав архитектуры процессора входит четыре шины (см. Рисунок 130), но по-прежнему поддерживается единое адресное пространство для хранения программ и данных. По J- и K-шинам передаются только данные, I-шина обслуживает команды, а S-шина обеспечивает доступ со стороны внешних ведущих (хоста или других процессоров).

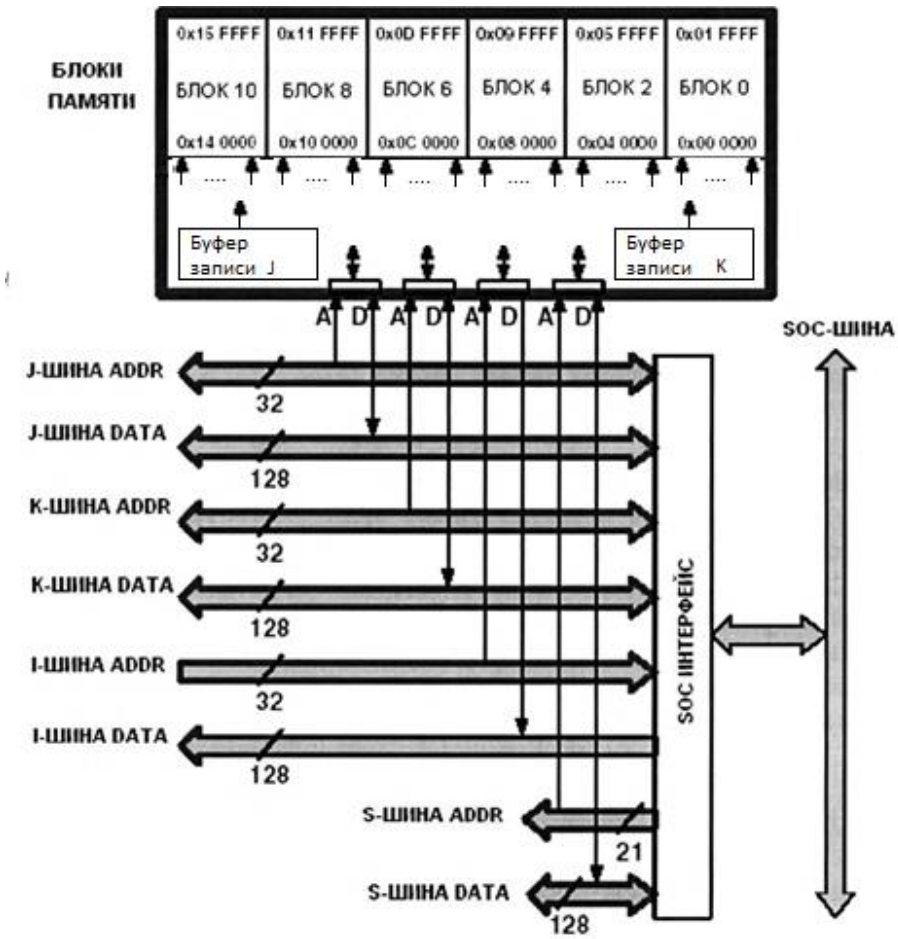


Рисунок 130 – Память и шины процессора

Внутренняя память процессора имеет 24 Мбит встроенной памяти типа SRAM, разбитой на 6 блоков по 4 Мбит (128 К 32-битных слов). Каждый блок может хранить либо команды, либо данные, либо и то, и другое, так что прикладные программы могут распределять память, исходя из собственных нужд. Помещение

команд и данных в различные блоки памяти, позволяет, однако, процессору осуществлять доступ к данным, выполняя одновременно выборку команд.

К каждому из 6-ти блоков внутренней памяти подключены четыре шины J, K, I, S, а также две шины записи JW и KW буферов записи. Каждый блок имеет внутреннюю логику арбитража доступа, что позволяет внутренней памяти выполнять четыре операции обмена с памятью за такт.

Из рисунка видно, что память может выполнять 6 операций за такт: четыре операции чтения и две операции записи. Однако, эти две операции записи на самом деле являются отложенными операциями, т.е. невозможно каждый последовательный такт иметь 6 операций доступа (если в каком-то такте мы имеем 6 операций доступа, значит в другом мы имели максимум 2).

Архитектура внутренних шин процессора дает возможность ядру и системе ввода/вывода осуществлять доступ к двенадцати 32-битным словам данных и четырем 32-битным командам на каждом такте. Гибкая структура памяти процессора обеспечивает:

- доступ ввода/вывода через SOC-шину и доступа ядра процессора к разным блокам памяти в одном и том же такте;
- доступ ядра процессора параллельно к трем блокам памяти – к одному блоку для выборки команд и к двум блокам для считывания/записи данных;
- программируемое разбиение памяти на программную память и память данных;
- программный доступ ко всей памяти как к 32-битным и 64-битным словам;
- программный доступ ко всей памяти как к 128-битным словам с выравниванием на 16-битную или 32-битную границу с использованием буфера выравнивания данных (DAB);
- внешний доступ ко всей памяти как к 128-битным словам.

Поскольку шины памяти (J-, K-, I- и S-шина) привязаны к функциональным блокам (J- IALU, K-IALU, устройство управления и SOC-интерфейс), а не к блокам памяти, конфликт по доступу к памяти происходит, когда два (или более) функциональных блока пытаются обратиться к одному и тому же внутреннему блоку памяти в одном и том же такте. Такой конфликт, если он возникает, приводит к дополнительным тактам потерь по конфликтующему доступу. Дополнительную информацию см. в разделе «Арбитрация шин при обращении к памяти» и «Конвейер памяти».

При двойном обращении к данным в течение одного такта ядро процессора использует независимые шины J и K для одновременного доступа к данным в двух блоках памяти. Хотя двойное обращение к данным обеспечивает большую пропускную способность, важно иметь в виду некоторые ограничения на то, как оно может быть использовано в программах.

Ограничения на двойное обращение к данным в течение одного такта следующие:

- два элемента данных должны принадлежать двум различным блокам памяти;
- если ядро пытается обратиться к двум словам из одного и того же блока памяти одной командой, будет порожден один такт приостановки. Дополнительную информацию о том, как шины получают доступ к блокам памяти, см. в разделе «Конвейер памяти»;
- данные должны принадлежать блокам памяти, отличным от блока, из которого осуществляется выборка команды. Если происходит обращение к блоку памяти за данными из которого в данном такте осуществляется выборка команд, то это приведет к приостановке конвейера команд.

Эффективное использование памяти зависит от того, как программа и данные в ней расположены, а также от того, как программа осуществляет доступ к данным. Дополнительную информацию см. в разделе «Работа с памятью. Рекомендации по программированию». Как показано на рисунке 130 процессор имеет четыре внутренних шины, подсоединенных к внутренней памяти. Обращения к памяти со стороны процессорного ядра (целочисленных ALU или устройства управления) используют J-шину, K-шину или I-шину, в то время как периферийные устройства процессора (внешний порт и DMA) используют S-шину для доступа к памяти. Используя S-шину SOC-интерфейс может обеспечить пересылку данных между внутренней памятью и внекристальными устройствами (к примеру, портами линков, внешней памятью и другими процессорами) без того, чтобы препятствовать доступу к памяти со стороны процессорного ядра.

Процессор использует систему приоритетов при арбитрировании двух или более шин, выполняющих одновременный доступ к одному и тому же блоку внутренней памяти. Дополнительную информацию см. в разделе «Арбитрация шин при обращении к памяти».

Если внутренняя память процессора подразделяется на *блоки* внутренней встроенной SRAM, то пространства внешней памяти подразделяются на *банки* SRAM и SDRAM, а управление выбором между ними осуществляется соответствующими линиями. Дополнительную информацию см. в главах «Кластерная шина» и «Интерфейс SDRAM» Спецификации процессора.

9.1 Физическая структура блока памяти

Внутренняя память состоит из шести блоков, каждый из которых представляет собой встроенную статическую память (SRAM) объемом 4 Мбит (128 К слов). Поскольку память поддерживает однотактные операции с данными шириной 128 бит, то в действительности внутренняя организация памяти есть 32К x 128 бит. Память работает на частоте ядра. Каждый блок способен предоставлять доступ объемом до 128 бит на каждом CCLK-такте без исключения. Для увеличения скорости работы памяти, а также для снижения энергопотребления, модуль 32К физически состоит из 8-ми модулей по 4К квадрослов, хотя для пользователя это практически не имеет значения. В одном такте активен только один модуль из 8-ми, что позволяет сэкономить потребление и увеличить скорость, т.к. модуль 4К более быстродействующий чем модуль 32К. Платой за это является некоторое увеличение площади. Отметим также что базовый модуль памяти является однопортовым, т.е. невозможно выполнение двух операций с одним модулем памяти в одном такте.

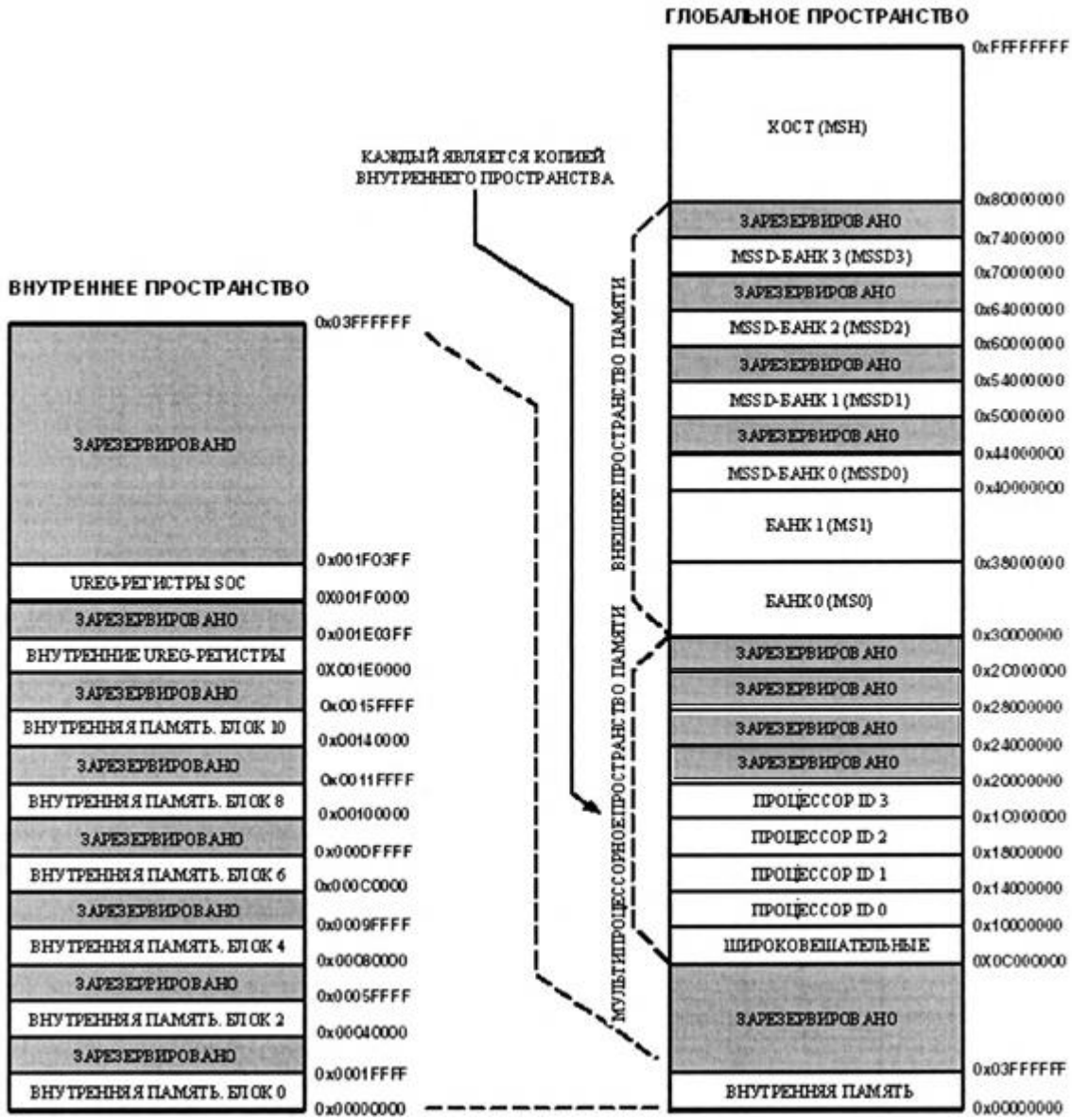


Рисунок 131 – Карта памяти процессора семейства

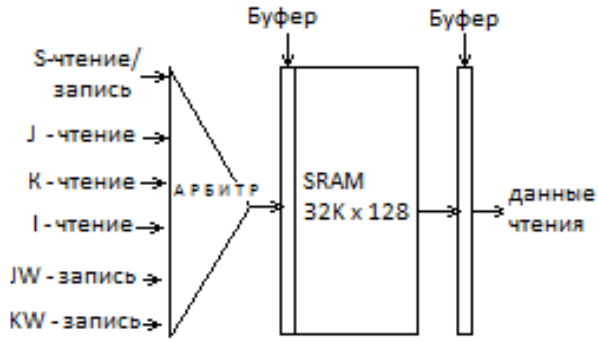


Рисунок 132 – Физическая структура блока памяти (версия 1)

Как видно из рисунка 132, каждый блок памяти обслуживает в каждом такте запросы от 6-ти возможных источников. Только один запрос получает доступ к накопителю памяти. Данная структура модуля памяти использовалась в первой версии

процессора. Во второй версии процессора структура модуля памяти была усложнена. Поскольку физически модуль состоит из 8-ми блоков 4К квадрослов, то блоки были поделены на две группы: в одной группе хранятся четные квадрослова, а в другой нечетные. Структура модуля памяти приведена на рисунке 133.

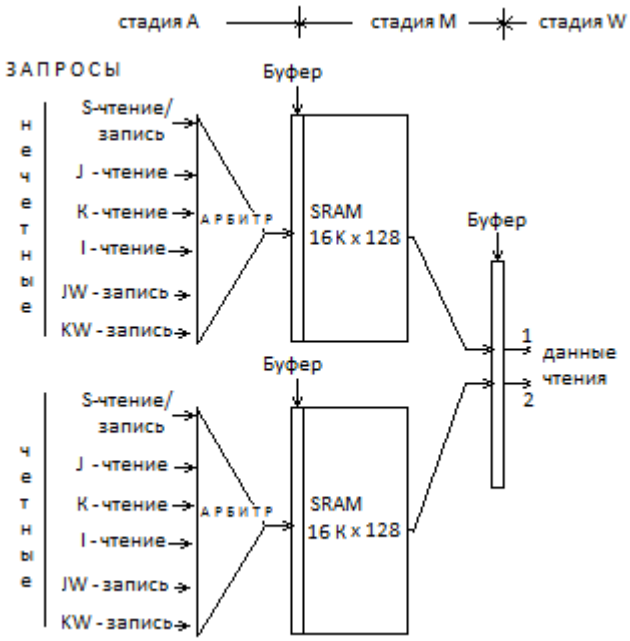


Рисунок 133 – Физическая структура блока памяти (версия 2)

При такой организации модуля памяти для каждой из групп имеется свой арбитр. В этом случае если к модулю памяти имеется два запроса, и они адресуют разные по четности квадрослова, то запросы могут быть обслужены в течение одного такта. Запросы попадающее в один тип квадрослов, обслуживаются поочередно. Таким образом, если имеется, например, запрос на чтение по шине K, то анализируется бит 2 адреса и, если его значение равно нулю, запрос посылается к группе четных квадрослов, а, если бит 2 равен единице, к группе нечетных квадрослов. И в первой, и во второй версии остается проблема арбитража нескольких запросов к одному банку памяти.

9.1.1 *Обращение к данным по чтению*

Транзакция чтения может быть инициирована четырьмя источниками: J-IALU, K-IALU, SOC-интерфейсом и устройством управления. Модули IALU формируют запрос на чтение на стадии A конвейера обработки команд. Устройство управления формирует запрос на чтение на стадии IA конвейера выборки команд. SOC-интерфейс формирует запрос на чтение, когда в его буфер загружается соответствующий запрос от внешнего ведущего устройства. Запросы поступают на арбитр блока памяти и одному из них в текущем такте предоставляется доступ к памяти. Память является синхронной статической и это означает, что в конце стадии конвейера по положительному фронту клока ядра запрос (адрес) защелкивается в накопителе памяти. В следующем такте данные считываются из накопителя и опять же положительному фронту клока ядра защелкиваются в выходном буфере модуля памяти. Далее данные передаются шине, инициировавшей запрос чтения.

9.1.2 *Обращение к данным по записи*

Транзакция записи может быть инициирована тремя источниками: J-буфером записи, K-буфером записи и SOC-интерфейсом. Буферы записи J и K формируют запросы после того как в них произведена запись с соответствующих шин. SOC-интерфейс формирует запрос на запись, когда в его буфер загружается соответствующий запрос от внешнего ведущего устройства. Запросы поступают на арбитр блока памяти и одному из запросов в текущем такте предоставляется доступ к памяти. Память является синхронной статической и это означает, что в конце стадии конвейера по положительному фронту клока ядра запрос (адрес и данные) защелкивается в накопителе памяти. В следующем такте данные записываются в накопитель. Почему запись в память с шин J и K производится через буфера записи, а не напрямую? Дело в том, что запрос на чтение внутренней памяти и запрос на запись внутренней памяти формируются на разных стадиях конвейера: чтение на A, а запись на W. Для операции записи – это очень удобно, т.к. данные в этот момент всегда готовы и нет тактов ожидания. Однако, из-за запросов на разных стадиях получается, что возможны одновременные запросы к памяти по чтению и записи команд, которые находятся в разных строках команд.

Например, код

```
[J0+0] = XR0;; // stage W
NOP;; //stage M
XR1 = [J1+0];; // stage A
```

вызовет конфликт одновременного доступа к банку памяти если регистры J0 и J1 адресуют один банк памяти. При этом команды находятся в разных линиях, и пользователь не ожидает никаких потерь производительности. В связи с этим запись всегда производится в буфер записи. Каждая из шин имеет свой буфер записи. Размер буфера равен четырем квадрословам. Буфер записи конкретной шины один для всех модулей внутренней памяти. После записи данных в буфер записи, команда записи считается выполненной и процессор о ней забывает. Буфер записи определяет, в какой блок памяти необходимо выполнить запись и посылает соответствующий запрос. Запрос записи имеет самый низкий приоритет, поэтому он ожидает удобного момента, когда к требуемому блоку памяти нет других запросов. Запрос записи становится высокоприоритетным, когда буфер заполняется и конвейер вынужден остановиться, когда в буфере нет места, чтобы поместить в него новые данные.

9.1.3 *Арбитраж шин при обращении к памяти*

При этом шина S выполняет операции чтения и записи, а шина I – только операции чтения. От шин J и K к блоку памяти поступают запросы на чтение, а от буферов записи JW и KW запросы на запись. Арбитр блока памяти имеет фиксированную схему приоритетов доступа. Если в текущем такте есть все шесть запросов к одному модулю памяти, то приоритет предоставления следующий (от самого высокого к самому низкому):

- S-шина (чтение и запись) – самый высокий приоритет;
- J-шина (чтение);
- K-шина (чтение);
- JW-шина (запись);
- KW-шина (запись);
- I-шина (чтение) – самый низкий приоритет.

Наличие одновременных запросов со стороны S-шины и со стороны J- или K-шин к одному банку вызывает приостановку конвейера исполнения команд на один такт. Наличие одновременных запросов со стороны S-шины и со стороны I-шины к

одному банку вызывает приостановку конвейера выборки команд на один такт. Запросы со стороны SOC-интерфейса не могут формироваться каждый такт клока ядра, т.к. они поступают в буфер интерфейса с частотой SOC-шины, т.е. частота этих запросов в 2 раза ниже.

В случае отсутствия запросов со стороны SOC-интерфейса, даже при наличии одновременно трех запросов на чтение со стороны J-, K-, I-шин, конвейеры исполнения и выборки команд не останавливаются, а продолжают движение. В этом случае арбитр выбирает наиболее высокоприоритетный запрос чтения (например, J), а оставшиеся два (K и I) помечает как задержанные (delayed) и все три запроса передает в буфер памяти. В этом случае на стадии M память в первом такте обслуживает запрос J, во втором такте – запрос K, в третьем такте – запрос I. В этом случае два такта блок памяти будет недоступен для новых внешних запросов, в том числе и для запросов со стороны SOC-интерфейса. Можно сказать, что запросы K- и I-шин, «проскочив» на стадию M, получают более высокий приоритет чем запрос S-шины. Таким образом, для случая задержанных запросов последовательность приоритетов будет следующей:

- K_delay (чтение) – самый высокий приоритет;
- I_delay (чтение);
- S-шина (чтение и запись);
- J-шина (чтение);
- K-шина (чтение);
- JW-шина (запись);
- KW-шина (запись);
- I-шина (чтение) – самый низкий приоритет.

Различные комбинации запросов будут вызывать различное число тактов приостановки конвейера. В таблице 51 приведены значения приостановок для различных ситуаций. В таблице символом «+» обозначено наличие запроса. Символом «-» обозначено отсутствие запроса. Пустая клетка означает любое значение. Одновременное наличие запросов K_delay и I_delay означает, что сначала будет обслужен K_delay запрос, а затем I_delay.

Таблица 51 – Такты приостановки из-за конфликта доступа к ресурсу памяти

K_dly	I_dly	S	J	K	I	Количество тактов приостановки
+		+				1 такт для S-шины
	+	+				1 такт для S-шины
-	-	+	+			1 такт для конвейера обработки команд
-	-	+		+		1 такт для конвейера обработки команд
+						1 такт для конвейера обработки команд
	+	-	+			1 такт для конвейера обработки команд
	+	-		+		1 такт для конвейера обработки команд
	+					1 такт для конвейера выборки команд

Из описанной схемы приоритетов видно, что если, например, имеются запросы от буферов записи (JW, KW) и есть более высокоприоритетный запрос, то запросы записи ничего не предпринимают и просто ожидают своей очереди. Если, например, имеются запросы на чтение от J- и K-шин, и есть более высокоприоритетный запрос, то запросы записи ничего не предпринимают и ожидают своей очереди. Однако, когда нет более высокоприоритетных запросов, то они передаются на следующую стадию вместе. Поскольку оба этих запроса принадлежат одной линии команд, то лучше всего их обрабатывать если не в одном такте, то хотя бы в двух последовательных тактах. Это вызывает наименьший простой конвейера исполнения.

Отдельно остановимся на обслуживании запроса от I-шины. Его обслуживание происходит немного интереснее. Видим, что его приоритет самый низкий и при наличии

плотного потока запросов от конвейера исполнения или от S-шины, у I-шины нет шансов получить обслуживание. Поэтому I-запрос ведет себя иначе. Когда нет более высокоприоритетных запросов, запрос I-шины передается на стадию M и обслуживается. Когда есть более высокоприоритетный запрос, I-запрос все равно проходит на стадию M вместе с высокоприоритетным запросом. I-запрос проходит на следующую стадию, продвигая конвейер выборки, но ждет там пока обслужится высокоприоритетный запрос, и после него обслуживается сам, задерживая при этом всю очередь. Переместившись таким образом на стадию M, запрос получат статус I_delay и обслуживается уже согласно схемы приоритетов. Такой способ предпринимается даже когда оба J- и K-запроса передаются на стадию M- и I-запрос будет уже третьим. Исключение составляет только наличие запроса по чтению со стороны S-шины. В этом случае I-запрос не перемещается на следующую стадию вместе с S-запросом, а приостанавливается на текущей стадии. Связано это с возможным конфликтом использования общего буфера.

Используя подобную тактику, конвейер выборки имеет возможность продвигаться даже при интенсивных обращениях в общий модуль памяти. Учитывая, что каждый такт адрес команд меняет одно квадрослово на противоположное (четное на нечетное и наоборот), число вероятных конфликтов доступа к общему ресурсу сокращается.

9.2 Конвейер памяти

В главе «Устройство управления» приведено подробное описание конвейера процессора. Отметим, что конвейер процессора состоит из конвейера выборки команд и конвейера исполнения команд. Продвижение на конвейере выборки команд не зависит напрямую от продвижения конвейера исполнения команд. Взаимосвязь между конвейерами осуществляется через буфер выравнивания команд IAB. Конвейер памяти является составной частью как конвейера выборки команд, так и конвейера исполнения команд. На рисунке 134 приведен конвейер команд процессора в привязке к конвейеру доступа к памяти. Стадии конвейера памяти, в составе конвейера процессора, выделены прямоугольником.

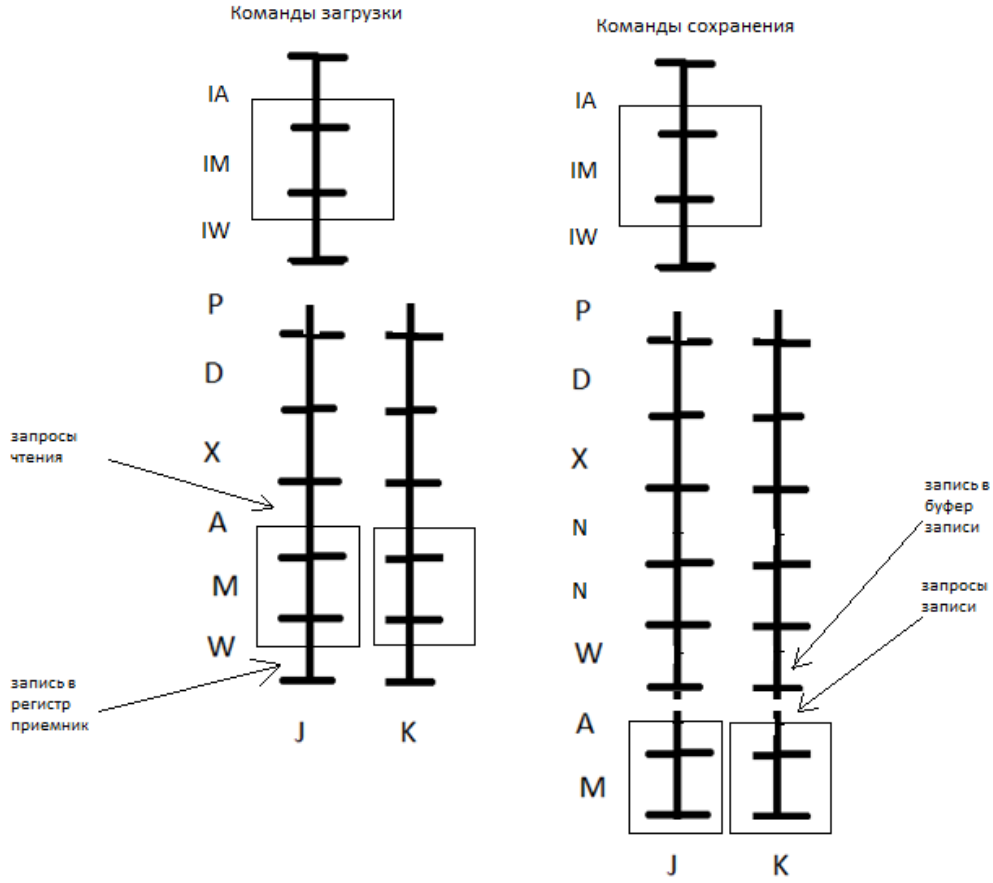


Рисунок 134 – Конвейер команд в привязке к конвейеру доступа к памяти.

Стадии конвейера памяти показаны на рисунке 134. Конвейер памяти является составной частью конвейера выборки команд. Стадия адреса IA начинается с передачи адреса квадрослова команд в модуль памяти, дешифрации адреса, выбора блока памяти к которому выполняется запрос и заканчивается арбитражем доступа к накопителю памяти. На стадии IM выполняется чтение квадрослова из накопителя в выходной буфер блока памяти. На стадии IW выполняется пересылка команд в буфер IAB.

Конвейер исполнения команд задействует конвейер памяти только когда команды загрузки данных из внутренней памяти либо команды сохранения данных во внутренней памяти выполняются на линиях конвейера J и (или) K. Все другие команды процессора никоим образом не используют конвейер памяти и не оказывают друг на друга никакого взаимного влияния. Конвейер команд загрузки вычисляет адрес доступа на стадии X. Стадия адреса A начинается с передачи адреса чтения данных в модуль памяти, дешифрации адреса, выбора блока памяти к которому выполняется запрос и заканчивается арбитражем доступа к накопителю памяти. На стадии M выполняется чтение данных из накопителя в выходной буфер блока памяти. На стадии W выполняется пересылка данных из буфера в регистр-приемник ядра.

Конвейер команд сохранения данных в памяти вычисляет адрес записи на стадии X. Адрес передается без изменений на стадию W, где команда завершает выполнение посредством передачи адреса и данных для записи в буфер записи. На этом команда завершается. Буфер записи самостоятельно посылает запрос на запись

в модуль памяти и завершает операцию записи. Данные в буфере записи могут хранится сколь угодно долго пока буфер записи не получит доступа к блоку памяти.

Рассмотрим пример (таблица 51) использования конвейера памяти в случае одновременного доступа к одному модулю памяти со стороны 6-ти источников. В данном примере предположим, что выборка команд происходит из того же блока памяти, на который указывают регистры J0 и K0. В этом случае стадии конвейера IA, IM, IW физически идентичны стадиям конвейера A, M, W, т.к. расположены в одном блоке памяти. Также предположим, что в такте 10 к этому же модулю памяти имеется запрос чтения со стороны S-шины.

Итого в такте 10 будет максимально возможное число запросов к одному банку памяти:

- запрос S-шины;
- запрос от команды #4 по шине J (стадия A, чтение);
- запрос от команды #4 по шине K (стадия A, чтение);
- запрос чтения команды по шине I (стадия IA, адрес 24);
- запрос записи данных от буфера записи JW;
- запрос записи данных от буфера записи KW.

В такте 9 выполнилась команда записи и в такте 10 мы получаем одновременно два запроса от буферов записи JW и KW. Конвейер выборки команд и конвейер исполнения команд приостанавливаются на один такт. В такте 11 мы видим тоже состояние этих конвейеров, как и в такте 10. В такте 11 в памяти обслуживается запрос S-шины. Действия, которые выполняет память выделены красным цветом. В такте 11 имеется уже 5 запросов к блоку памяти. В блок памяти передается три запроса чтения: J, K, I. Конвейер выборки и конвейер исполнения продвигаются на один такт. Кстати, обратим внимание на тот факт, что в такте 10, когда стадии IA, P, D, X, A конвейера вынуждены были остановиться, стадии M и W продолжили свое движение. Это позволяет командам #2 и #3 завершиться, несмотря на то что все другие команды притормозили.

В такте 12 блок памяти принял сразу три запроса на чтение. Запросы будут обслужены последовательно: в такте 12 – запрос #4-J, в такте 13 – запрос #4-K, в такте 14 – запрос $24. При этом обратим внимание, что в такте 14 конвейер исполнения уже продолжил движение, т.к. все его запросы были к этому такту обслужены памятью.

Таблица 52 – Конфликт доступа к одному блоку памяти

Код программы (начиная с адреса 0)

```
#1: [J0+=5] = XR0; [K0+=4] = YR0;; /* J0 и K0 указывают на один банк памяти*/
#2: nop;;
#3: nop;;
#4: XR1 = [J0+0]; YR1 = [K0+0];;
#5: nop;;
#6: nop;;
#7: nop;;
#8: nop;;
#9: nop; nop;;
#10: nop;;
```

Такт	IA	IM	IW	P	D	X	A	M	W	JW	KW
1	$0										
2	$4	$0									
3	$8	$4	$0								
4	$12	$8	$4	#1							

Такт	IA	IM	IW	P	D	X	A	M	W	JW	KW
5	$16	$12	$8	#2	#1						
6		$16	$12	#3	#2	#1					
7	$20		$16	#4	#3	#2	#1				
8		$20		#5	#4	#3	#2	#1			
9			$20	#6	#5	#4	#3	#2	#1		
10	$24			#7	#6	#5	#4	#3	#2	W-J	W-K
также имеем запрос чтения к банку памяти со стороны S-шины											
11	$24	S-bus		#7	#6	#5	#4	S-bus	#3	W-J	W-K
12		#4-J		#8	#7	#6	#5	#4-J		W-J	W-K
13		#4-K		#8	#7	#6	#5	#4-K		W-J	W-K
14		$24		#9	#8	#7	#6	#5 $24	#4	W-J	W-K
15	$28	W-J		#10	#9	#8	#7	#6 W-J	#5		W-K
16		$28 W-K		#11	#10	#9	#8	#7	#6		
17			$28	#12	#11	#10	#9	#8 W-K	#7		

В такте 14 мы получаем ситуацию, когда к блоку памяти остались только запросы от буферов записи. Поэтому в 14-м такте право доступа к памяти получает буфер записи JW. В 15-м такте мы получаем запрос к блоку памяти для чтения нового квадрослова команд и запрос записи от буфера KW. Преимущество при доступе имеет буфер записи, однако, если адрес записи соответствует четному квадрослову, то в этом же такте может получить обслуживание и запрос чтения команды, т.к. адрес $28 указывает на нечетное квадрослово. Обращаем ваше внимание, что если в линии команд #4 пользователь предполагает, что получит такт приостановки из-за конфликта доступа и получает такой такт, то для линии команд #0 такого конфликта доступа не произошло. При благоприятном стечении обстоятельств две команды записи в одной линии команд могут записать данные в один и тот же блок памяти без тактов приостановки

9.2.1 *Обзор возможных потерь времени при обращении к памяти*

Такты приостановки возникают при одновременном доступе к одному ресурсу разными модулями процессора. В процессоре второй версии таким ресурсом является банк памяти одной группы квадрослов (четной или нечетной). В первой версии таким ресурсов был весь банк памяти. Однако, не всякая приостановка вносит потери в производительность. Например, приостановка записи в буфере записи не влияет на продвижение конвейера и может не оказывать влияние на производительность процессора. Также приостановка конвейера выборки команд может не сказываться на производительности, если в буфере IAB имеются команды для передачи на конвейер исполнения. Потери всегда возникают в случае, когда в одной линии команд имеются две команды чтения и они обращаются к одному ресурсу памяти. Потери всегда возникают, когда после записи в память сразу же идет команда чтения этого же адреса памяти. В этом случае потери могут достигать 3-х тактов.

Как известно конвейер обработки может останавливаться, из-за конфликтов доступа, на любой стадии. Если на какой-то стадии возникает конфликт доступа, все предыдущие стадии конвейера приостанавливаются, а все последующие продолжают движение. Конвейер может остановиться и на стадии W. Это может быть вызвано,

например, ожиданием загрузки данных из внешней памяти или SOC-шины. Если команда остановилась на стадии W, то это означает что на стадии M может остановиться команда, которая выполняет чтение из накопителя памяти. Однако, приостановка команды чтения на стадии M не вызывает недоступности данного блока памяти для чтения команд или для доступа S-шины. Прочитанное значение перемещается в буфер и конвейер выборки команд либо S-шина могут производить операции с памятью. Также блок памяти доступен для выполнения операций записи со стороны буферов записи. Пример подобной ситуации приведен в таблице ниже.

Таблица 53 – Останов конвейера и доступ к памяти

Код программы (начиная с адреса 0)

```
#1: [J0+0] = XR0; [K0+1] = YR0;; /* J0 и K0 указывают на один банк памяти, четное квадрослово*/
#2: J8 = SYSCON;;
#3: nop;;
#4: XR1 = [J0+2]; YR1 = [K0+3];;
#5: nop;;
#6: jump lab_50;; /* промах в BTB в данном примере */
Lab_50: /* четное квадрослово*/
#50: nop;;
#51: nop;;
#52: nop;;
```

Такт	**IA**	**IM**	**IW**	**P**	**D**	**X**	**A**	**M**	**W**	**JW**	**KW**
1	$0										
2	$4	$0									
3	$8	$4	$0								
4	$12	$8	$4	#1							
5	$16	$12	$8	#2	#1						
6		$16	$12	#3	#2	#1					
7	$20		$16	#4	#3	#2	#1				
8		$20		#5	#4	#3	#2	#1			
9			$20	#6	#5	#4	#3	#2	#1		
10	$24			#7	#6	#5	#4	#3	#2	W-J	W-K
Команда #2 помещает запрос на чтение в буфер SOC-интерфейса. Конвейер исполнения останавливается в ожидании прочитанных данных из SOC-шины. Одновременно выполняется прогнозируемый переход по адресу 50. Конвейер выборки не останавливается											
11	$50	#4-J				#6	#5	#4-J	#3	W-J	W-K
12	$52	#4-K				#6	#5	#4-K	#3	W-J	W-K
13	$52	$50				#6	#5	$50	#3	W-J	W-K
14	$56	$52	$50			#6	#5	W-J	#3		W-K
15	$56	W-K	$52	#50		#6	#5	W-K	#3		
16	$60	$56		#50		#6	#5		#3		
17	$64	$60	$56	#50		#6	#5		#3		
18		$64	$60	#50		#6	#5		#3		
В 18-м такте происходит загрузка прочитанного значения из регистра SOC-шины											
19			$64	#51	#50		#6	#5	#4		
20				#52	#51	#50		#6	#5		

В описанном примере в такте 14 выполняется одновременное чтение команды по адресу 52 и запись W-J, т.к. запросы попадают в разные квадрослова. В такте 15 запрос чтения команды приостанавливается, т.к. запись W-K и чтение #56 попадают в четное квадрослово. Запись выполняется первой. Из примера видно, что несмотря на то, что конвейер обработки остановился в ожидании данных с SOC-шины, конвейер выборки команд за это время загрузил весь буфер IAB, буферы записи отослали все свои запросы в память и в дополнение к этому отложенный запрос чтения K был обработан во время приостановки конвейера и не внес дополнительного такта приостановки.

9.2.2 *Структура конвейера при доступе к памяти со стороны SOC-шины*

Внешние ведущие устройства (процессоры кластера, хост-процессор, контроллер DMA) могут осуществлять доступ к внутренней памяти процессора. Рассмотрим конвейер такого типа доступа на примере транзакции чтения внутренней памяти контроллером DMA. Диаграмма чтения приведена в таблице 54. В данном примере контроллер DMA запускает первую транзакцию #1 чтения, но прочитанные данные будут загружаться в передатчик одного из линк-портов. Вторая транзакция #2 чтения будет загружена в контроллер DMA (случай пересылки «внутренняя память – внешняя память»).

Таблица 54 – Конвейер доступа к памяти со стороны SOC-шины

Такт CCLK	**Такт SCLK**	**W IFIFO**	**R IFIFO**	**A**	**M**	**W OBUF**	**R OBUF**	**A SOC**	**M SOC**
1	1	#1							
2									
3	2	#2	#1						
4									
5	3		#2	#1					
6					#1				
7	4			#2		#1			
8					#2		#1		
9	5					#2		#1	
10							#2		
11	6							#2	#1
12									

В примере показаны два запроса чтения, которые формирует контроллер DMA при доступе к внутренней памяти. Контроллер работает на частоте SCLK, синхронной по отношению к частоте CCLK, но в 2 раза меньшей. В первом такте SCLK запрос от DMA попадает во входное IFIFO SOC-интерфейса (стадия W_IFIFO). Во втором такте SCLK запрос читается из IFIFO и загружается в промежуточный буфер, который тактируется CCLK (стадия R_IFIFO). Далее работает конвейер памяти на частоте CCLK. В 7-м такте CCLK прочитанные из памяти данные записываются в выходной буфер OBUF SOC-интерфейса (стадия W_OBUF). Из-за синхронности клоков, чтение буфера OBUF в промежуточный буфер SOC-шины возможен в 4-м такте SCLK (стадия R_OBUF). Дальше в 5-м такте SCLK выполняется арбитраж доступа к SOC-шине (OBUF имеет максимальный приоритет на стадии A_SOC), и на 6-м такте SCLK происходит запись данных в буферный регистр одного из передатчиков линк-портов.

Следом за транзакцией #1 была запущена транзакция #2. Второе чтение должно было вернуть данные во внутренний буфер контроллера DMA. Передача данных в DMA будет выполнена в 6-м такте, т.к. в этом случае OBUF не использует SOC-шину, а производит запись напрямую.

Как видно из примера, при максимальном потоке от SOC-шины, внутренняя память процессора загружена на 50 %.

9.3 Работа с памятью. Рекомендации по программированию

Синхронная статическая память SRAM не требует никаких специальных процедур обслуживания. Она сохраняет свое состояние при остановке синхронизации. В течение одного такта ядра можно выполнить доступ чтения данных либо записи данных. Основные рекомендации по программированию могут быть нацелены на минимизацию конфликтов доступа к памяти с целью достижения максимальной производительности. Учитывая, что вся внутренняя память разбита на 6 банков к которым возможен параллельный доступ, основной рекомендацией является размещение кода программы и используемых данных в разных банках памяти. Также, принимая во внимание, что ширина шины данных равна 128 бит, наиболее эффективное использование памяти будет при использовании всей ширины шины данных. Шина 128 бит очень удобна при сохранении и восстановлении контекста процессора во время обработки прерывания. Для ускорения сохранения и восстановления контекста в процессоре могут использоваться два указателя стеков, расположенных в разных банках памяти. Это позволяет за один такт ядра производить сохранение и загрузку до 8-ми регистров. Дополнительную эффективность работы с памятью добавляет разбиение каждого банка памяти на два подбанка с четными и нечетными квадрословами. В этом случае можно работать не с двумя, а с одним стеком. Как известно, компилятор всегда держит указатель стека выравненным на границе квадрослова. Если в модуле J указатель стека равен X, а в модуле K его значение равно X+4, то можно работать с одним стеком со скоростью обмена 8 регистров за такт. Это же обстоятельство можно учитывать и при работе с локальными переменными, расположенными на стеке, т.к. их смещение относительно указателя стека известно и может использоваться для параллельной выборки.

Разбиение банка памяти по квадрословам также положительно сказывается для случая, когда программа и данные размещены в одном блоке памяти. Как известно, команды выбираются квадрословами, и в этом случае уменьшается число конфликтов доступа.

10 Набор команд

В этой главе подробно описан набор команд процессора. Справки по командам сгруппированы в соответствии с устройствами, на которых эти команды исполняются:

- «Команды ALU»;
- «Команды CLU»;
- «Команды умножителя»;
- «Команды сдвигового устройства»;
- «Команды (целочисленные) IALU»;
- «Команды (загрузка/сохранение/пересылка) IALU»;
- «Команды устройства управления».

Более подробную информацию об этих архитектурных элементах процессора можно найти в разделах «Арифметико-логическое устройство (ALU)», «Коммуникационно-логическое устройство (CLU)», «Умножитель», «Сдвиговое устройство», «Целочисленное ALU (IALU)» и «Устройство управления». Более подробную информацию о регистрах (и правилах их именования), используемых в командах, можно найти в разделе «Регистры вычислительных блоков». Команды каждой группы подробно описаны в настоящей главе. Сводки всех команд можно найти в приложении «Краткий справочник» (A-1). Полную информацию о декодировании команд можно найти в приложении «Декодирование команд» (C-1).

10.1 Команды ALU

Это устройство выполняет все арифметические операции (сложение/вычитание) процессора над данными в формате с фиксированной и плавающей точкой, а также логические операции процессора над данными в формате с фиксированной точкой. ALU также выполняет операции преобразования данных, такие как операции расширения/сжатия над данными в форматах с фиксированной точкой. Описание операций ALU, флагов состояния ALU, ALU-предикатов, а также примеры можно найти в разделе «Арифметико-логическое устройство (ALU)».

Соглашения по обозначениям имён регистров, необязательных элементов или вариантов, используемые в справках на команды, подробно освещены в разделе «Регистры регистровых файлов».

10.1.1 *Сложение/вычитание*

Синтаксис

{X|Y|XY}{S|B}*Rs* = *Rm* + |- *Rn* {({S|SU}{NF})} ;

{X|Y|XY}{L|S|B}*Rsd* = *Rmd* +|- *Rnd* {({S|SU}{NF})} ;

Действие

Эти команды складывают или вычитают операнды в регистрах *Rm* и *Rn.* Результат помещается в регистр *Rs.* Префиксы L, S и B указывают тип операнда, а *d* указывает на двойной размер регистра (см. раздел «Регистры регистровых файлов»).

MAX_SN, MIN_SN и MAX_UN обозначают соответственно максимальное знаковое, минимальное знаковое и максимальное беззнаковое числа, представимые в выходном формате. Например, если выходной формат – это 16-битные короткие слова, то MAX_SN=0x7FFF, MIN_SN=0x8000 и MAX_UN=0xFFFF.

При включённом насыщении в операции сложения:

- Знаковое насыщение - опция (S):
  Rm+Rn превосходит MAX_SN, то *Rs* = MAX_SN о Если *Rm+Rn* менее MIN_SN, то Rs = MIN_SN

- Беззнаковое насыщение - опция (SU):
  Rm+Rn превосходитMAX_UN, то *Rs* = MAX_UN
  При включённом насыщении в операции вычитания:
- Знаковое насыщение - опция (S):
  Rm-Rn превосходит MAX_SN, то *Rs* = MAX_SN о Если *Rm-Rn* менее MIN_SN, то Rs = MIN_SN
- Беззнаковое насыщение - опция (SU):
  Rm-Rn менее 0, то *Rs* = 0

Флаги состояния

- AZ Установлен, если все биты результата равны 0
- AN Равен MSB результата
- AV (AOS) Знаковое переполнение
- AC Равен биту переноса из старшего разряда; может использоваться как признак беззнакового переполнения (инвертирован в случае с вычитанием)

Опции

- ( ) Без насыщения
- (S) С насыщением, знаковое (SU) С насыщением, беззнаковое
- (NF) Не обновлять флаги (состояния)

Примеры

YBR9 = R2 + R8 (S);; YSR2 = R1 - R0 (SU);;

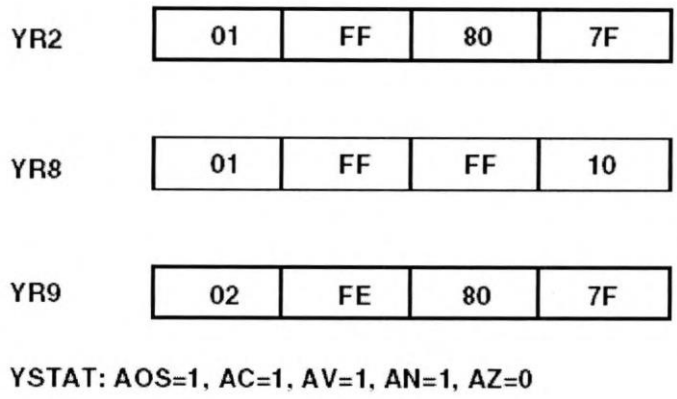


Рисунок 135 – Квадросложение со знаковым насыщением однобайтных операндов в одиночных регистрах вычислительного блока Y

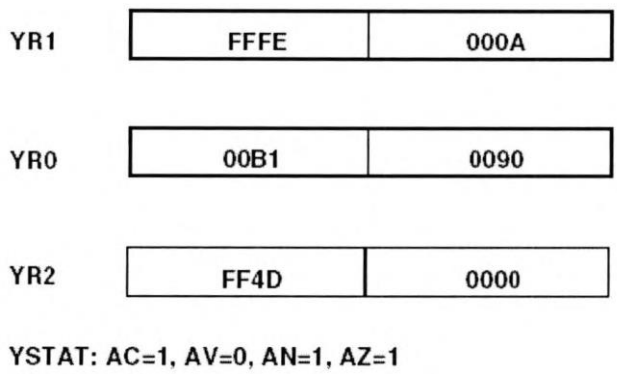


Рисунок 136 – Парное вычитание с беззнаковым насыщением коротких операндов в одиночных регистрах вычислительного блока Y

10.1.2 *Сложение/вычитание с переносом/займом*

Синтаксис

{X|Y|XY} Rs = Rm + CI {-1) ;

{X|Y|XY}L Rsd = Rmd + CI {-1} ;

{X|Y|XY} Rs = Rm + Rn + CI {({S|SU}{NF})} ;

{X|Y|XY} Rs = Rm - Rn + CI -1 {({S|SU}{NF})} ;

{X|Y|XY}L Rsd = Rmd + Rnd + CI { ({S | SU) {NF}) } ;

{X|Y|XY}L Rsd = Rmd - Rnd + CI -1 {({S|SU}{NF})} ;

Действие

Эти команды складывают операнды в регистрах *Rm* и *Rn,* добавляя бит переноса, или вычитают их друг из друга, вычитая также бит займа. Результат помещается в регистр *Rs.* Перенос (CI) имеет то же значение, что и флаг AC регистров X/YSTAT. Операнд *Rn* может быть опущен, в этом случае происходит сложение с переносом или вычитание с займом регистра *Rm* и CI. Результат помещается в регистр *Rs.* Префикс L указывает на операнды с длинными данными, и суффикс *d* указывает на двойной регистр.

При включённом насыщении в операции сложения с переносом:

- Знаковое насыщение - опция (S):
  Rm+Rn+CI > MAX_SN, то Rs = MAX_SN о Если Rm+Rn+CI < MIN_SN, то Rs = MIN_SN
- Беззнаковое насыщение - опция (SU):
  Rm+Rn+CI > MAX_UN, то Rs = MAX_UN

При включённом насыщении в операции вычитания с займом:

- Знаковое насыщение - опция (S):
  Rm-Rn+CI-1 >MAX_SN, то Rs = MAX_SN о Если Rm-Rn+CI-1 <MIN_SN, то Rs = MIN_SN
- Беззнаковое насыщение - опция (SU):
  Если Rm-Rn+CI-1 < 0, то Rs = 0

Флаги состояния

- AZ Установлен, если все биты результата равны 0
- AN Равен MSB результата
- AV (AOS) Знаковое переполнение
- AC Равен биту переноса из старшего разряда; может использоваться как признак беззнакового переполнения (инвертирован в случае с вычитанием)

Опции

- ( ) Без насыщения
- (S) С насыщением, знаковое
- (SU) С насыщением, беззнаковое (NF)Не обновлять флаги (состояния)

Примеры

XR2 = 0x00000001;; XR0 = 0xFFFFFFFF;;
XR1 = R0 + R0;; /* *порождает перенос* */ XR3 = R2 + R2 + CI;;
Результатом этих команд будет XR3 = 0x00000003 и XSTAT = 0. R9 = R3 - R10 + CI-1;;
Если R3 = 0x17 и R10=0x04, а *бит переноса от предыдущей операции* ALU
равен 1, *то* R9 = 0x13, *результирующий бит переноса* AC=1.

10.1.3 *Сложение/вычитание с делением пополам*

Синтаксис

{X|Y|XY}{ S|B} Rs = (Rm +|- Rn)/2 {({T}{U}{ NF})} ;

{X|Y|XY}{ L|S|B}Rsd = (Rmd + |- Rnd) /2 { ({T}{U}{NF}) } ;

Действие

Эти команды складывают или вычитают операнды в регистрах Rm и Rn и делят результат пополам. Так как бит переноса, получаемый в результате сложения, сдвигается вправо в позицию MSB выходного значения, то результат не может иметь переполнения. Возможно округление к ближайшему чётному (округление без систематического смещения) или усечение. Если полученный результат отрицателен, а представление беззнаковое, то Rs равен нулю и устанавливается бит переполнения. Префиксы L, S и B указывают тип операнда, а d указывает на двойной размер регистра (см. раздел «Регистры регистровых файлов»).

Флаги состояния

- AZ Установлен, если все биты результата равны 0
- AN Равен MSB результата
- AV (AOS) Для сложения – сброшен (AOS не изменяется); для вычитания с включёнными опциями (U) и (TU) – установлен в случае отрицательного результата, во всех остальных случаях – сброшен
- AC Сброшен

Опции

- ( ) - Знаковое, округление к ближайшему чётному
- (T) - Знаковое, с усечением
- (U) - Беззнаковое, округление к ближайшему чётному
- (TU) - Беззнаковое, с усечением
- (NF) - Не обновлять флаги (состояния)

Бит переноса в результате сложения сдвигается вправо в MSB согласно следующим правилам:

- знаковые входные данные – опции ( ) и (T): если AV= 1, то вдвинуть cn в MSB, иначе – сдвинуть результат арифметически вправо;
- беззнаковые входные данные – опции (U) и (TU): для сложения, вдвинуть cn в MSB; для вычитания – сдвинуть 0 в MSB.

Примеры

R9 = (R3 + R5)/2 ;; округление к ближайшему чётному
Если R3=0x1 *и* R5=0x2,
то R9 = 0x02.
Если R3=0x3 и R5=0x2,
то R9 = 0x02.
Если R3=0x7FFFFFFF и R5 = 0x1,
то бит переноса вдвигается в R9 = 0x40000000.
Если R3=0x80000001 и R5=0x1,
то арифметический сдвиг вправо даёт результат R9 = 0xC0000001.

10.1.4 *Абсолютное значение/ абсолютное значение суммы или разности*

Синтаксис

{X|Y|XУ}{S|B} Rs = ABS Rm {(NF)} ;

{X|Y|XY}{L|S|B} Rsd = ABS Rmd {(NF)} ;

{X|Y|XY}{S|B} Rs = ABS (Rm + Rn) {({X}{NF})} ;

{X|Y|XY}{L|S|B} Rsd = ABS (Rmd + Rnd) { ({X} {NF}) } ;

{X|Y|XY}{S|B} Rs = ABS (Rm - Rn) {({X|U}{NF})} ;

{X|Y|XY}{L|S|B} Rsd = ABS (Rmd - Rnd) {({X|U} {NF})} ;

Действие

Эти команды выполняют операцию сложения или вычитания над знаковыми операндами в регистрах *Rm* и *Rn,* затем абсолютное значение нормализованного результата помещается в регистр *Rs.* Если операнд *Rn* опущен, то выполняется операция взятия абсолютного значения от *Rm.*

Префиксы L, S и B указывают тип операнда, а *d* указывает на двойной размер регистра.

В случае сложения опция X предоставляет возможность получения результата в расширенном диапазоне. Обычный диапазон значений результатов этой операции без опции X - от нуля до максимального положительного числа, или 0x0 - 0x7F...F.

Результат, не входящий в этот диапазон, устанавливается равным максимальному положительному числу (насыщение). С включённой опцией X результатами операции будут беззнаковые числа в диапазоне 0x0-0xF...F.

В случае вычитания опция X предоставляет возможность получения результата в расширенном диапазоне. Обычный диапазон значений результатов этой операции без опции X - от нуля до максимального положительного числа, или 0x0 - 0x7F...F. Обычный диапазон может быть выбран как для знаковых ( ), так и для беззнаковых (U) входных данных. Результат, не входящий в этот диапазон, устанавливается равным максимальному положительному числу (насыщение). С включённой опцией X результатами операции будут беззнаковые числа в диапазоне 0x0-0xF...F, а входные данные предполагаются знаковыми.

Флаги состояния

- AZ Установлен, если все биты результата равны 0
- AN Равен MSB результата до операции ABS
- AV (AOS) Для ABS: установлен, если значение в *Rm* равно наименьшему отрицательному числу, иначе – сброшен.

  Для ABS (со сложением): установлен, если результат больше максимального знакового числа (0x7F...F) при не используемой опции X; если оба операнда равны 0x80 . .0 при используемой опции X.

  Для ABS (с вычитанием): установлен, если результат больше максимального знакового числа при не используемой опции X; иначе – сброшен.

- AC Сброшен

Опции

- ( ) Знаковые выходные данные, диапазон результатов [0x0, 0x7F...F]
- (X) Беззнаковые выходные данные, диапазон результатов [0x0, 0xF... F]
- (U) Беззнаковые входные и выходные данные
- (NF) Не обновлять флаги (состояния)

Примеры

R5 = ABS R4;;
Если R4 = 0x80000000,
то R5 = 0x7FFFFFFF, *и биты* AV, AN *установлены.*
Если R4 = 0x7FFFFFFF,
то R5 = 0x7 FFFFFFF.
Если R4 = 0xA0000000,

то R5 = 0x10000000,» *бит* AN *установлен.*
XR3 = ABS (R0 + R1) ;;
Если XR0 = 0x80000001 *и* XR1=0x80000001,
то XR3 = 0x7FFFFFFF, *и биты регистра* XSTAT *равны.* AOS=1, AC=0, AV=1, AN=0, AZ=0.
XR3 = ABS (R0 + R1) (X);;
Если XR0 = 0x80000001 mXR1=0x80000001,
то XR3 = 0xFFFFFFFE, *и биты регистра* XSTAT *равны.* AOS=0, AC=0, AV=0, AN=0, AZ=0.
R3 = ABS (R0 - R1) ;;
Если R0 = 0x6 и R1=0xA,
то R3 = 0x4.
Если R0 = 0xA и R1=0x6,
то R3 = 0x4.
Если R0 = 0x7FFFFFF8 *и* Rl = 0xFFFFFFF0,
то R3 = 0x7 FFFFFFF,
а если включена опция X, *то* R3 = 0x80000008.

10.1.5 *Дополнение*

Синтаксис

{X|Y|XY}{S|B) Rs = - Rm { (NF) } ;

{X|Y|XY}{L|S|B} Rsd = - Rmd { (NF) } ;

Действие

Эта команда возвращает дополнение до двух операнда в регистре *Rm.* Результат помещается в регистр *Rs.* Если входное значение равно минимальному отрицательному числу, представимому в указанном в команде формате, результатом операции будет максимальное положительное число.

Префиксы L, S и B указывают тип операнда, а *d* указывает на двойной размер регистра.

Флаги состояния

- AZ – Установлен, если все биты результата равны 0
- AN – Равен MSB результата
- AV(AOS) – Установлен для минимального отрицательного числа (0x80...0); в остальных случаях
- AC – Сброшен

Опции

(NF) – Не обновлять флаги (состояния)

Пример

XBR6 = -R3;;
Если XR3 = 0x808150FF,
то XR6 = 0x7F7FB001, *и биты регистра* XSTAT *равны.* AOS=l, AC=0, AV=1, AN=1, AZ=0.

10.1.6 *Максимум/минимум*

Синтаксис

{X|Y|XY} {S|B}Rs = MAX | MIN *(Rm, Rn)* { ((U} { Z) {NF})} ;

{X|Y|XY} {L|S|B}Rsd = MAX | MIN *(Rmd, Rnd)* { ({U} { Z }{NF}) } ;

Действие

Эти команды возвращают максимум (больший) или минимум (меньший) из двух операндов в регистрах Rm и Rn. Результат помещается в регистр Rs. Сравнение производится покомпонентно: однобайтные значения – с однобайтными значениями, короткие – с короткими, обычные – с обычными или длинные – с длинными, в зависимости от указанного размера данных, а максимальное или минимальное значение от каждого сравнения помещается в соответствующий компонент (байтный/короткий/обычный/длинный) регистра результата.

Если в команде указана опция Возврат к нулю (Z), то значение в регистре результата Rs будетравно операнду Rm только если Rm>=Rn для максимума или Rm<=Rn для минимума: в противном случае Rs= 0.

Префиксы L, S и B указывают тип операнда, а *d* указывает на двойной размер регистра.

Также результатом выполнения команды дополнительно может быть номер максимума или минимума во входном массиве. Подробнее об этой опции см. раздел «Ссылка на дополнение МАКС/МИН».

Флаги состояния

- AZ – Установлен, если все биты результата равны 0
- AN – Равен MSB результата
- AV(AOS) – Сброшен (не изменяется)
- AC – Сброшен

Опции

- ( ) – Знаковое
- (U) – Беззнаковое
- (Z) – Опция знаковый. Возврат к нулю
- (UZ) – Опция беззнаковый Возврат к нулю
- (NF) – Не обновлять флаги (состояния)

Примеры

SR9:8 = MAX (R3:2, R1:0) ;; *см. рисунок 137*

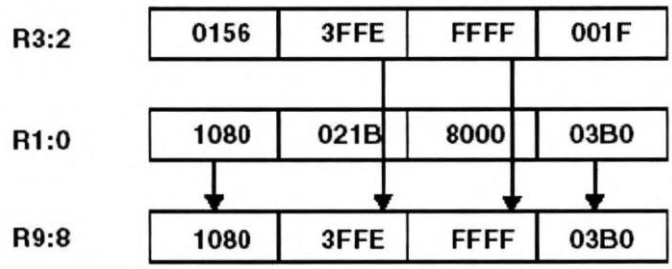


Рисунок 137 – Пример для четырёх коротких слов

SR9:8 = MAX (R3:2, R1:0) (Z);; СМ. рисунок 138

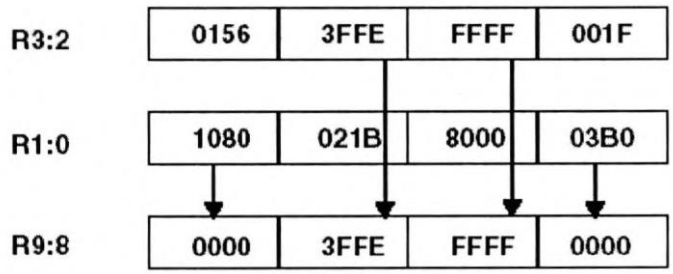


Рисунок 138 – Пример для четырёх коротких слов

SR9:8 = MIN (R3:2, R1:0) ;; см. рисунок 139

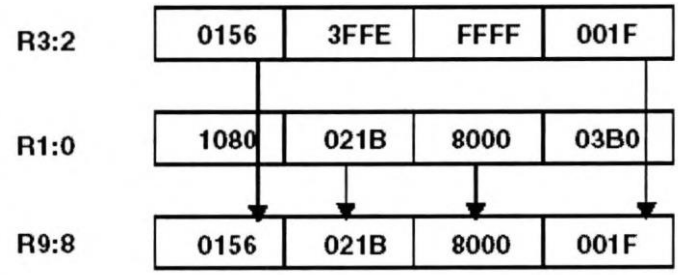


Рисунок 139 – Пример для четырёх коротких слов

SR9:8 = MIN (R3:2, R1:0) (Z);; См. рисунок 140.

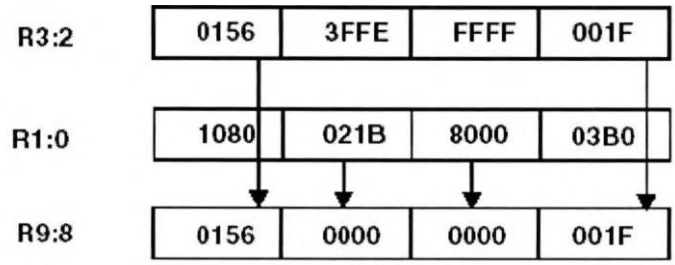


Рисунок 140 – Пример для четырёх коротких слов

10.1.7 *Максимум/минимум по Витерби*

Ситаксис

{X|Y|XY}S|B Rsd = VMAX|VMIN *(Rmd, Rnd)* {(NF)} ;

Действие

Эти команды возвращают максимум (больший) по Витерби или минимум (меньший) по Витерби из двух операндов в регистрах Rm и Rn. Результат помещается в регистр Rs. Сравнение производится покомпонентно: однобайтные значения – с однобайтными значениями, короткие – с короткими, в зависимости от указанного размера данных, а максимальное или минимальное значение от каждого сравнения помещается в соответствующую компоненту (байтную/короткую) регистра результата. Биты, соответствующие выбранным значениям максимума или минимума, помещаются в старшие 4 или 8 битов регистра PR1: 0 после того, как предыдущее содержимое PR1: 0 было сдвинуто вправо на то же количество битов.

Алгоритм вычисления максимума по Витерби:

for i = 0 to n—1 *(n равно 8 для восьми однобайтных слов данных и 4 для четырёх коротких)*

if *Rm*(i) >= *Rn*(i) then

Rs(i) = *Rm*(i)

PR1:0 = {1, PR1:0 [63 :1]}

else

Rs(i) = *Rn*(i)

PR1:0 = {0, PR1:0[63:1]}

end *(if)*

end *(for)*

Алгоритм вычисления минимума по Витерби:

for i = 0 to n-1 *(n равно 8 для восьми однобайтных слов данных и 4 четырёх коротких)*
if *Rm(i)* < *Rn*(i) then
Rs(i) = *Rm*(i)
PR1:0 = {1, PR1:0[63:1]}
else
Rs(i) = *Rn*(i)
PR1:0 = {0, PR1:0[63:1]}
end *(if)*
end (for)

Префиксы S и B указывают тип операнда, а *d* указывает на двойной размер регистра. Сравнение всегда производится над числами в знаковом формате.

Флаги состояния

- AZ Установлен, если все биты результата равны 0
- AN Равен MSB результата
- AV (AOS) Сброшен (не изменяется)
- AC Сброшен

Опции

- (NF) Не обновлять флаги (состояния)

Примеры

Начальное значение PR1: 0 = 0xABC01281 987FED35.
SR9:8 = VMAX (R3:2, R1: 0) ; ; *см. рисунок 141*
Начальное значение PR1: 0 = 0xABC01281 987FED35.
SR9: 8 = VMAX (R1:0, R3:2*);; см. рисунок 142*

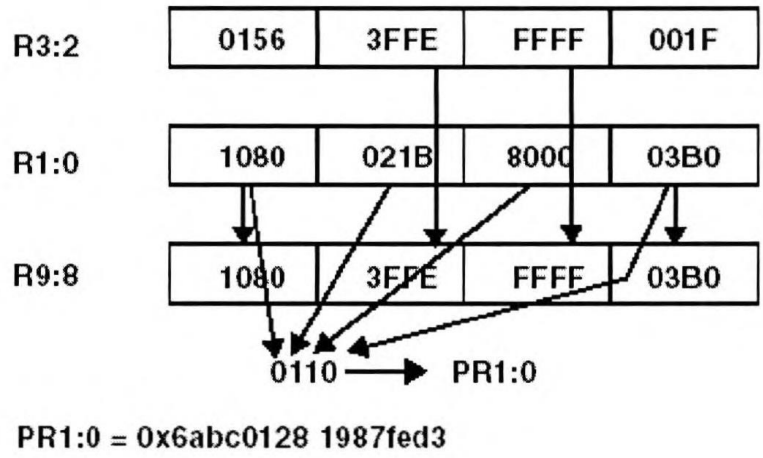


Рисунок 141 – Пример для четырёх коротких слов данных

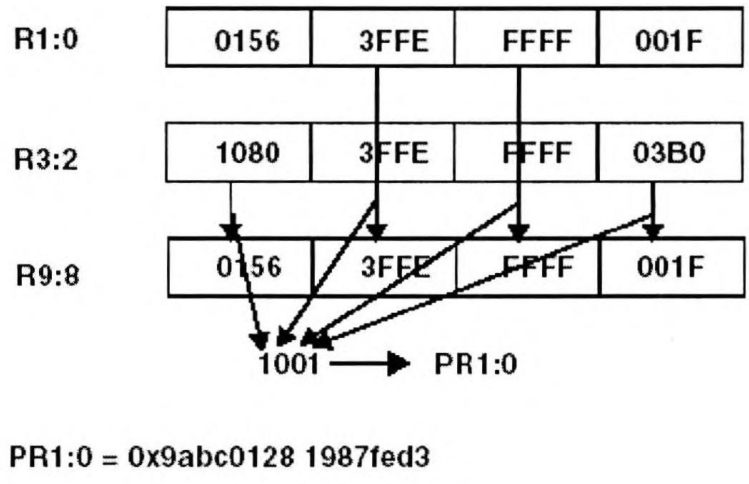


Рисунок 142 – Пример для четырёх коротких слов данных

10.1.8 *Инкремент/декремент*

Синтаксис

{X|Y|XY}{S|B}*Rs* = INC|DEC *Rm* {({S|SU}{NF})} ;

{X|Y|XY}{L|S|B}*Rsd* = INC|DEC *Rmd* {({S|SU}{NF})} ;

Действие

Эти команды прибавляют единицу к операнду в регистре *Rm* или вычитают единицу из операнда в регистре *Rm* Результат помещается в регистр *Rs.*

Префиксы L, S и B указывают тип операнда, а d указывает на двойной размер регистра см. раздел «Регистры регистровых файлов».

Знаковое насыщение - опция (S):

- Если (*Rm*+1) >MAX_SN, то *Rs*=MAX_SN;
- Если (*Rm*-1) <MIN_SN, то *Rs* MIN_SN

Беззнаковое насыщение - опция (SU):

- Если *(Rm-1)* <0, то *Rs* = 0
- Если (*Rm*+1) >MAX_U, то *Rs* = MAX_U

MAX_SN и MIN_SN обозначают соответственно максимальное и минимальное знаковые числа, представимые в выходном формате, а MAX_UN – максимальное беззнаковое число. Например, если выходной формат – это 16-битные короткие слова, то MAX_SN=0x7FFF, MIN_SN=0x8000.

Флаги состояния

AZ Установлен, если все биты результата равны 0;

AN Равен MSB результата;

AV (AOS) Знаковое переполнение;

AC Равен биту переноса из старшего разряда; может использоваться как признак беззнакового переполнения (инвертирован в случае с декрементом).

Опции

( ) Без насыщения

(S) С насыщением, знаковое

(SU) С насыщением, беззнаковое

(NF) Не обновлять флаги (состояния)

Примеры

R6 = INC R3 ;;

Если R3 = 0x00...01D,

то R6 = 0x00...01E.

R6 = DEC R3 ;;
Если R3 = 0x00...01D,
то R6 = 0x00...01C.

10.1.9 *Сравнение*

Синтаксис

{X|Y|XY}{S|B}COMP(*Rm, Rn)* {(U)} ;

{X|Y|XY}{L|S|B}COMP *(Rnd, Rnd)* {(U)} ;

Действие

Эта команда производит сравнение операнда в регистре *Rm с* операндом в регистре *Rn.* Команда устанавливает флаг AZ, если два операнда равны, флаг AN – если операнд в регистре *Rm* меньше операнда в регистре *Rn.* Сравнение производится покомпонентно: однобайтные значения - с однобайтными значениями, короткие - с короткими, обычные - с обычными, длинные с длинными, в зависимости от указанного размера данных. Заметим, что также как и для всех команд вычислительных блоков с множественными входными данными (например, команда BCOMP *(Rm, Rn*); однобайтное квадросравнение), флаги статуса завершения данной команды вычисляются с помощью битового ИЛИ соответствующих флагов каждой операции.

Префиксы L, S и B указывают тип операнда, а *d* указывает на двойной размер регистра см. раздел «Регистры регистровых файлов».

Флаги состояния

AZ Установлен, если значения в *Rm* и *Rn* одинаковы;
AN Установлен, если значение в *Rm* меньше значения в *Rn;*
AV(AOS) Сброшен (не изменяется);
AC Сброшен.

Опции

( ) Знаковое
(U) Беззнаковое

Примеры

COMP(R3,R5);;
Если R3=0xFFFFFFFF и R5=0x0,
то AN=1 и AZ=0. COMP(R3,R5) (U);;
Если R3=0xFFFFFFFF и R5=0x0,
mo AN=0 и AZ=0.
R3 = 0x00000000;; R5 = 0x7F7F0000;;
XBCOMP(R3,R5);;
IF XALT, JUMP my_label ;;
/* Переход имеет место, если значение любого байта в R3 меньше значения соответствующего байта в R5. */
XBCOMP(R5, R3) ;;
IF NXALE, JUMP my_label ;;
/* Переход имеет место, если значение всех байтов в R3 меньше значения соответствующих байтов в R5. */

10.1.10 *Отсечение по границе*

Синтаксис

{X|Y|XY}{ S|B} *Rs* = CLIP *Rm BY Rn* {(NF)} ;

{X|Y|XY}{L|S|B} *Rsd* = CLIP *Rmd* BY *Rnd* {(NF)} ;

Действие

Эта команда возвращает значение знакового операнда в регистре *Rm,* если его абсолютное значение меньше абсолютного значения операнда в регистре *Rn.* В противном случае возвращает | *Rn* |, если значение в *Rm* положительно, и -| *Rn* | если значение в *Rm* отрицательно. Результат операции помещается в регистр *Rs.*

Префиксы L, S и B указывают тип операнда, а *d* указывает на двойной размер регистра см. раздел «Регистры регистровых файлов».

Флаги состояния

AZ Установлен, если все биты результата равны 0;
AN Равен MSB результата;
AV (AOS) Сброшен (не изменяется);
AC Сброшен.

Опции

(NF) Не обновлять флаги (состояния)

Пример

XR3:2 = CLIP R9:8 BY R1: 0 ;; см. *рисунок 143*

XR9:8	F0000000	0000077B
XR1:0	00FFABCD	FFFA0000
XR3:2	FF005433	0000077B

XSTAT: AC=0, AV=0, AN=1, AZ=0

Рисунок 143 – Двойное обычное слово в вычислительном блоке X

10.1.11 *Продольная сумма*

Синтаксис

{X|Y|XY} *Rs* = SUM *SRm*|*BRm* {({U}{NF})} ;

{X|Y|XY} *Rs* = SUM *SRmd*|*BRmd* {({U}{NF})} ;

Действие

Эта команда складывает однобайтные/короткие слова в регистре *Bm* и сохраняет результат в регистре *Rs.* Если однобайтные/короткие слова знаковые, то они расширяются знаком перед операцией сложения. Результат операции всегда выровнен по правому краю - например, двоичная точка результата всегда находится справа от LSB.

Префиксы B и S указывают на однобайтные или короткие операнды соответственно, а *d* указывает на двойной размер [регистра - см. раздел «Регистры регистровых файлов».

Флаги состояния

AZ Установлен, если все биты результата равны 0;
AN Равен MSB результата;
AV(AOS) Сброшен (не изменяется);
AC Сброшен.

Опции

( ) Знаковое целое
(U) Беззнаковое целое

(NF) Не обновлять флаги (состояния

Примеры

XR4 = SUM SR3:2 ;; см. рисунок 144

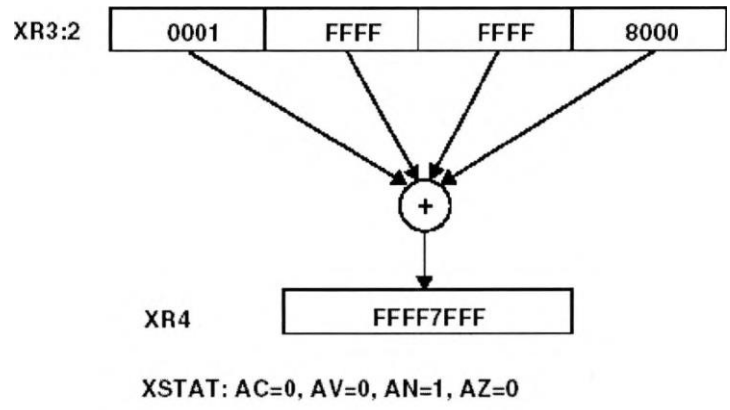


Рисунок 144 – Знаковая операция SUM

XR4 = SUM SR3:2 (U);; см. рисунок 145

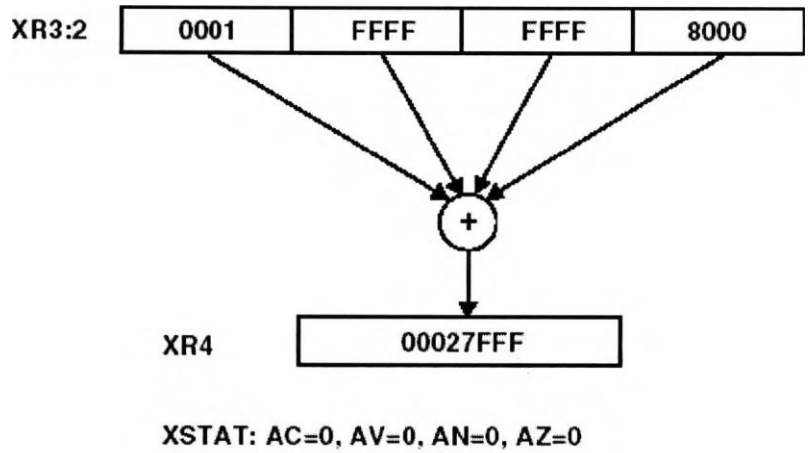


Рисунок 145 – Беззнаковая операция SUM

10.1.12 *Подсчёт единиц*

Синтаксис

{X|Y|XY}*Rs* = ONES *Rm**Rmd* {(NF)} ;

Действие

Эта команда подсчитывает количество битовых единиц в операнде в регистре *Rm.*

Результат помещается в регистр *Rs.*

Суффикс *d* указывает на двойной размер регистра - см. раздел «Регистры регистровых файлов».

Флаги состояния

AZ Установлен, если все биты результата равны 0;

AN Сброшен;

AV(AOS) Сброшен (не изменяется);

AC Сброшен.

Опции

(NF) Не обновлять флаги (состояния)

Пример

R6 = ONES R3;;

Если R3 = b#00...011101, moR6 = 0x00...04.

10.1.13 *Загрузка/пересылка регистра PR (побочного результата)*

Синтаксис

{X|Y|XY}PR1:0 = *Rmd* {(NF)} ;

{X|Y|XY}*Rsd* = PR1:0 {(NF)} ;

Действие

Эти команды загружают в регистр PR1: 0 операнд из регистра Rmd или загружают в регистр Rsd операнд из PR1: 0.

Суффикс d указывает на двойной размер регистра - см. раздел «Регистры регистровых файлов».

Флаги состояния

AZ Установлен, если все биты результата равны 0;

AN Равен MSB результата;

AV (AOS) Сброшен (не изменяется);

AC Сброшен.

Опции

(NF) Не обновлять флаги (состояния)

10.1.14 *Инкремент указателя однобитного FIFO*

Синтаксис

{X|Y|XY}*Rs* = BFOINC *Rmd* {(NF)} ;

Действие

Эта команда извлекает по семь наименее значащих разрядов из двух операндов, располагающихся соответственно в младшей и старшей частях двойного регистра *Rmd,* складывает их, делит сумму на 64 и помещает остаток от деления в шесть наименее значащих разрядов регистра результата, обозначенного здесь *Rs* (рисунок 146). Более подробную информацию можно найти в разделе «Операции с битовыми потоками».

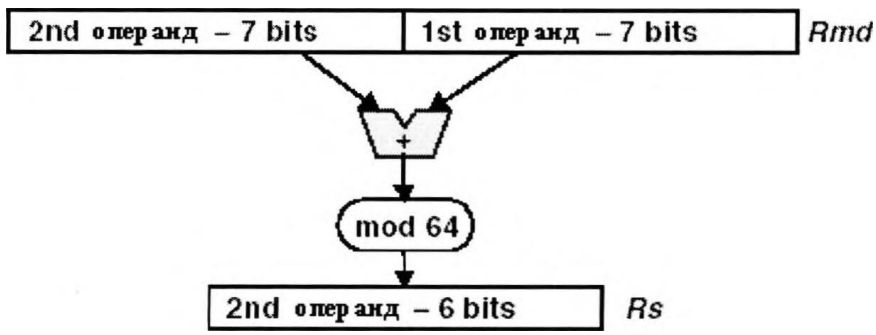


Рисунок 146 – Остаток от деления в операции BFOINC

Флаги состояния

AZ Установлен, если все биты результата равны 0;

AN Установлен, если результат сложения превысил 63;

AV (AOS) Сброшен (не изменяется);

AC Сброшен.

Опции

(NF) Не обновлять флаги (состояния)

Пример

{X|Y|XY}*Rs* = BFOINC *Rmd* ;
Если Rmd = 0x0...030...020,
то Rs = 0x0...010 *и флаг* AN *установлен.*

10.1.15 *Абсолютное значение с побочным накоплением*

Синтаксис

{X|Y|XY}PR0|PR1 += ABS (*SRmd* - *SRnd)* {({TJ}{NF})} ;

{X|Y|XY}PR0|PR1 += ABS *(BRmd- BRnd)* {({U}{NF})} ;

Действие

Эта команда вычитает однобайтные/короткие операнды в регистровой паре *Rnd* из соответствующих им операндов в регистровой паре *Rmd,* проводит операцию взятия абсолютного значения над результатами соответствующих размеров; продольно складывает полученные положительные значения и прибавляет эту величину к содержимому одного и регистров PR. Результат последней операции вновь сохраняется в регистре PR. Значения в регистрах PR всегда выровнены по правому краю - например, двоичная точка всегда находится справа от LSB. Значения в регистрах PR предполагаются знаковыми, если не указана опция (U) . Насыщение всегда включено.

Префиксы S и B указывают тип операнда, а суффикс *d* указывает на двойной размер регистра - см. раздел «Регистры регистровых файлов».

Флаги состояния

Последняя операция сложения со значением в PR влияет на следующие флаги:

AZ Установлен, если все биты результата равны 0;

AN Равен MSB результата;

AV (AOS) Установлен в случае знакового переполнения результата; иначе – сброшен;

AC Сброшен.

Опции

- ( ) Знаковые входные данные
- (U) Беззнаковые входные данные
- (NF) Не обновлять флаги (состояния)

Примеры

XPR0 += ABS (SR3:2 - SR1:0 *) ; ;* см. рисунок 147.

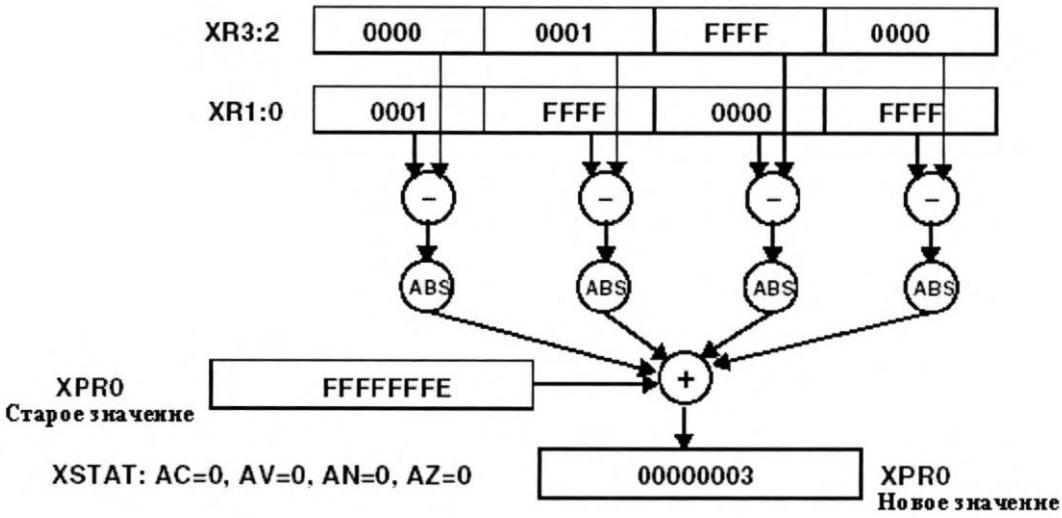


Рисунок 147 – Операция ABS с побочным накоплением

10.1.16 *Продольная сумма с побочным накоплением*

Синтаксис

{X|Y|XY}PR0|PR1 += SUM S*Rm* {({U}{NF})} ;

{X|Y|XY}PR0|PR1 += SUM S*Rmd* {({U}{NF})} ;

{X|Y|XY}PR0|PR1 += SUM B*Rm* {({U}{NF})} ;

{X|Y|XY}PR0|PR1 += SUM B *Rmd* {({U}{NF})} ;

Эта команда проводит операция сложения над однобайтными или короткими операндами в регистре Rm и прибавляет полученную величину к содержимому одного из регистров PR. Если однобайтные/короткие операнды знаковые, то они расширяются знаком перед операцией сложения. Конечный результат вновь помещается в регистр PR. Значения в регистрах PR всегда выровнены по правому краю, например, двоичная точка всегда находится справа от LSB. Значения в регистрах PR могут быть как знаковыми, так и беззнаковыми. Операция сложения со значением в PR всегда происходит с насыщением. Префиксы S и B указывают тип операнда, а суффикс d указывает на двойной размер регистра (см. раздел «Регистры регистровых файлов»).

Флаги состояния

Последняя операция сложения со значением в PR влияет на следующие флаги:

AZ Установлен, если все биты результата равны 0;

AN Равен MSB результата;

AV(AOS) Знаковое переполнение без опции (U), беззнаковое переполнение с опцией (U); иначе AV сброшен;

AC Сброшен.

Опции

( ) Знаковые входные данные и результат

(U) Беззнаковые входные данные и результат

(NF) Не обновлять флаги (состояния)

Пример

PR0 += sum SR3:2;; см. Рисунок 148.

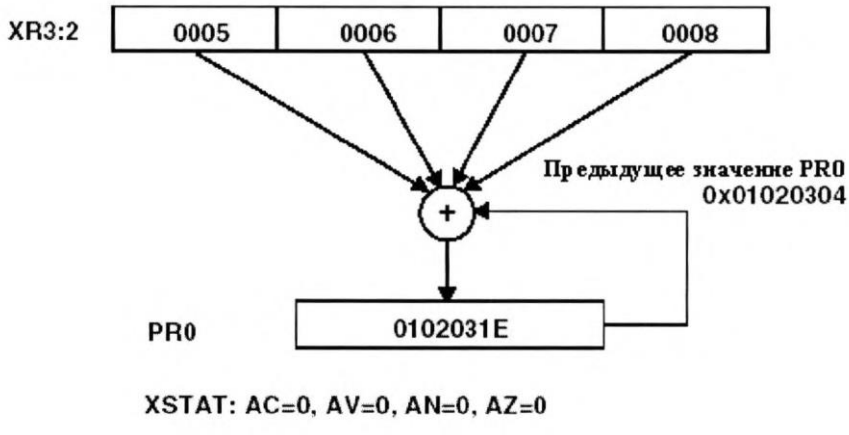


Рисунок 148 – Операция SUM с побочным накоплением

10.1.17 *Сложение/вычитание (двойная операция)*

Синтаксис

{X|Y|XY}{S|B}Rs = Rm + Rn, Ra = Rm - Rn { (NF) } ; (двойная операция)

{X|Y|XY}{L|S|B}Rsd = Rmd + Rnd, Rad = Rmd - Rnd {(NF)} ; (двойная операция)

Действие

Эта команда одновременно складывает и вычитает операнды в регистрах Rm и Rn. Результаты помещаются в регистры Rs и Ra. Насыщение всегда включено и результат знаковый.

Префиксы L, S и B указывают тип операнда, а d указывает на двойной размер регистра см. раздел «Регистры регистровых файлов».

Флаги состояния

AZ Установлен, если все биты результата равны 0;

AN Равен результату операции битовое ИЛИ над MSB двух результатов;

AV (AOS) Знаковое переполнение в одном из результатов;

AC Сброшен.

Опции

(NF) Не обновлять флаги (состояния)

Пример

R9 = R4 + R8, R2 = R4 - R8 ;;
Если R4 = 8 и R8 = 2,
то R9 = 10 и R2 = 6.

10.1.18 *Передать*

Синтаксис

{X|Y|XY}*Rs* = PASS *Rm* ;

{X|Y|XY}*LRsd* = PASS *Rmd* ;

Действие

Эта команда передаёт через ALU операнд из регистра *Rm* в регистр *Rs.*

Префикс L указывает тип операнда, а *d* указывает на двойной размер регистра (см. раздел «Регистры регистровых файлов»).

Флаги состояния

AZ Установлен, если все биты результата равны 0;

AN Равен MSB Rm;

AV (AOS) Сброшен;
AC Сброшен.

Пример

XR6 = PASS R3;;
Если R3 = 0x80000000,
то R6 = 0x80000000 *и значения битов регистра* XSTAT *таковы:* AC=0, AV=0, AN=1, AZ=0.

10.1.19 *Логическое И/И-НЕ/ИЛИ/ИСКЛ-ИЛИ/НЕ*

Синтаксис

{X|Y|XY}*Rs* = *Rm* AND|AND NOT|OR|XOR *Rn* {(NF)} ;

{X|Y|XY}LRsd = *Rmd* AND|AND NOT|OR|XOR *Rnd* { (NF) }

{X|Y|XY}*Rs* = NOT *Rm* {(NF)} ;

{X|Y|XY}*LRsd* = NOT *Rmd* {(NF)} ;

Действие

Эти команды проводят побитовые логические операции И (AND), И-НЕ (AND NOT), ИЛИ (OR) или ИСКЛ-ИЛИ (XOR) над операндами в регистрах Rm и Rn. Команда NOT проводит операцию логического дополнения над операндом в регистре Rm. Результат помещается в регистр Rs.

Префикс L указывает тип операнда, а d указывает на двойной размер регистра (см. раздел «Регистры регистровых файлов»).

Флаги состояния

AZ Установлен, если все биты результата равны 0;
AN Равен MSB Rm;
AV (AOS) Сброшен;
AC Сброшен.

Опции

(NF) Не обновлять флаги (состояния)

Примеры

R5 = R4 AND R8;;
Если R4 = b#...1001 и R8 = b#. . .1100,*то* R5 = b# . . . 1000.
R5 = R4 AND NOT R8;;
Если R4 = b#...1001 и R8 = b#. . .1100, *то* R5 = b#. . . 0001.
R5 = R3 OR R4;;
Если R3 = b#...1001 и R4 = b#. . .1100, *то* R5 = b# . . . 1101.
R5 = R2 XOR *R7;;*
Если R2 = b#...1001 и R7 = b#. . .1100, *то* R5 = b# . . . 0101.
R5 = NOT R3;;
Если R3 = b#0 ... 011101, = b#. . .1100, *то* R5 = b#1... 100010.

Префикс b# обозначает двоичную запись.

10.1.20 *Расширение*

Синтаксис

{X|Y|XY}*Rsd* = EXPAND S*Rm* {+|- *SRn*} {({I|IU}{NF})} ;

{X|Y|XY}*Rsq* = EXPAND S*Rmd* {+|- S*Rnd*}{({I|IU}{NF})) ;

{X|Y|XY}S*Rsd* = EXPAND B*Rm* {+|- B*Rn*}{({I|IU}{NF})} ;

{X|Y|XY}*SRsq* = EXPAND B*Rmd* {+|- B*Rnd*} {({I|IU}{NF})} ;

Действие

Эти команды складывают или вычитают операнды в регистрах *Rm* и *Rn,* затем осуществляет приведение результатов, расширяя 8-битные значения до 16-битных или 16- битные значения до 32-битных. Операнд *Rn* может быть опущен, что приводит к простому расширению в *Rm.* Если тип данных дробный, в конец результат добавляются битовые ноли. Если тип данных целочисленный, в начало результата добавляются битовые ноль (или битовые единицы для расширяемого знаком отрицательного числа).

Эти команды расширяют результат следующим образом:

- *Rsd* = EXPAND*SRm* расширяет два коротких слова до двух обычных;
- *Rsq* = EXPAND*SRmd* расширяет четыре коротких слова до четырёх обычных;
- *SRsd* = EXPAND*BRm* расширяет четыре однобайтных слова до четырёх коротких;
- *SRsq* = EXPAND*BRd* расширяет восемь однобайтных слов до восьми коротких.

Результат с фиксированной точкой помещается в регистр Rs.

Префикс S обозначает короткий тип операндов, префикс B обозначает однобайтные операнды. Суффиксы d и q обозначают размер регистров (см. раздел «Регистры регистровых файлов»).

Флаги состояния

AZ Установлен, если все биты результата равны 0;

AN Равен MSB результата;

AV(AOS) Установлен, если произошло переполнение во время операции с дробными числами, или в случае опции IU результат отрицательный;

AC Сброшен.

Опции

( ) Знаковые входные данные и результат

(U) Беззнаковые входные данные и результат

(NF) Не обновлять флаги (состояния)

Примеры

{X|Y|XY}Rsd = EXPAND SRm; см. Рисунок 149

{X|Y|XY}Rsq = EXPAND SRmd;

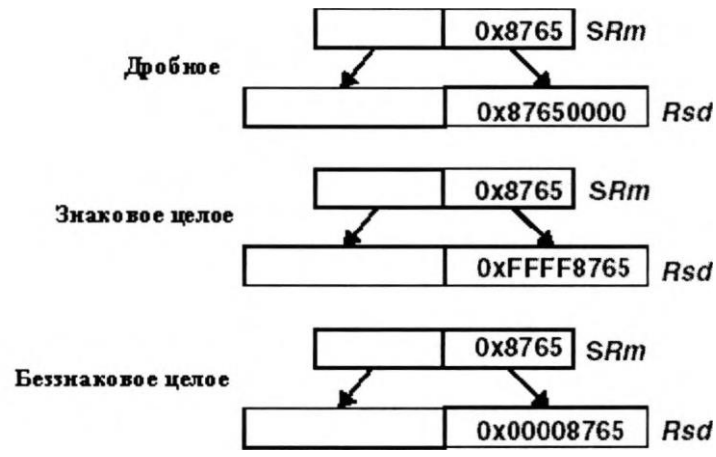


Рисунок 149 – Дробная, знаковая целая и беззнаковая целая операция EXPAND

{X|Y|XY}SRsd = EXPAND BRm; см. рисунок 150

{X|Y|XY}SRsq = EXPAND BRmd;

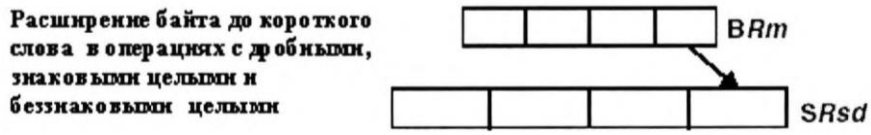


Рисунок 150 – Расширение однобайтных слов до коротких слов

{X|Y|XY}Rsd = EXPAND SRm + SRn; см. рисунок 151
{X|Y|XY}Rsq = EXPAND SRmd + SRnd;

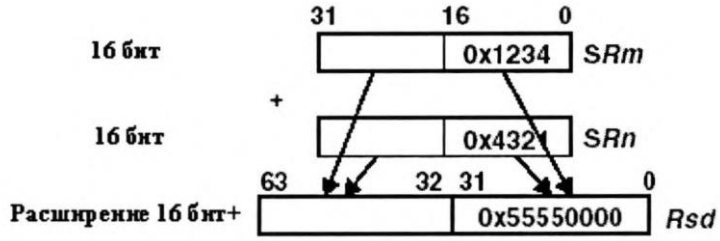


Рисунок 151 – Расширение коротких слов до обычных слов

{X|Y|XY}SRsd = EXPAND BRm + BRm;
{X|Y|XY}SRsq = EXPAND BRmd + BRmd; см. рисунок 152

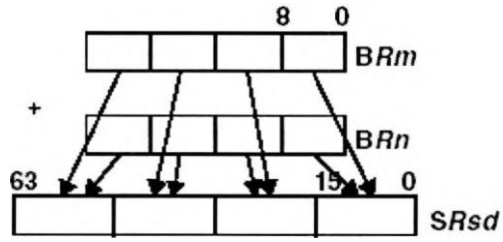


Рисунок 152 – Сложение и расширение однобайтных слов до коротких слов

{X|Y|XY}Rsd = EXPAND SRm - SRn; см. рисунок 153
{X|Y|XY}Rsq = EXPAND SRmd - SRnd;

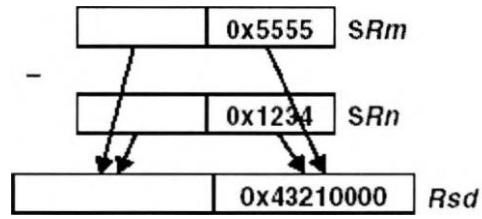


Рисунок 153 – Вычитание и расширение коротких слов до обычных слов

{X|Y|XY}SRsd = EXPAND BRm - BRm; см. рисунок 154
{X|Y|XY}SRsq = EXPAND BRmd - BRmd;

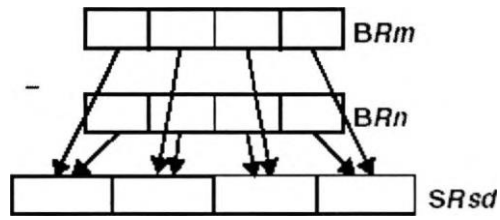


Рисунок 154 – Вычитание и расширение однобайтных слов до коротких слов

10.1.21 *Сжатие*

Синтаксис

{X|Y|XY}*SRs* = COMPACT *Rmd* {+|- *Rnd*} {({T|I|IS|ISU}{NF})} ;

{X|Y|XY}*BRs* = COMPACT S*Rmd* {+|- S*Rnd*} {({T|I|IS|ISU}{NF})} ;

{X|Y|XY}*Rs* = COMPACT L*Rmd* {({U|IS|ISU}{NF})} ;

{X|Y|XY}LRsd = COMPACT *QRmq* {({U|IS|ISU}{NF})} ;

Действие

Эти команды выполняют пооперандное сложение и вычитание регистров *Rmd* и *Rnd,* промежуточные результаты сжимаются до данных аналогичного типа, но меньшей точности. Окончательные результаты помещаются в регистр *Rs.* Операнд *Rnd* может быть опущен, тогда действие команды сводится к сжатию *Rmd.*

Сжатие может быть выполнено одним из следующих способов:

- два обычных слова – в два коротких слова (SRs = COMPACT Rmd);
- четыре коротких слова – в четыре однобайтных слова (BRs = COMPACT SRmd);
- одно длинное слово – в одно обычное слово (Rs = COMPACT LRmd);
- одно квадрослово – в одно длинное слово (LRs = COMPACT QRmq*).*

Опции (I), (IS) и (ISU) позволяют использовать два типа данных – дробные (по умолчанию) и целочисленные.

Сжатие дробных операндов производится пересылкой старшей части результата в регистр приёмника с округлением к ближайшему чётному (по умолчанию), либо усечением младшей половины – опция (T). Переполнение может возникнуть только при округлении максимального положительного числа, в этом случае происходит насыщение результата (в предположении, что данные знаковые).

Сжатие целых операндов производится пересылкой младшей части результата в регистр приёмника. При сжатии целых операндов режим насыщения может быть выбран с помощью опции. Когда режим насыщения выключен (по умолчанию), старшие биты входных знаковых операндов используются только для отслеживания переполнения. Флаг переполнения устанавливается, если хотя бы один из старших битов отличен от MSB результата (бит знака). С включённым режимом насыщения на знаковых входных данных опция (IS) – в случае переполнения (определяется так же, как и с опцией (I)) результат устанавливается равным 0x7FF...F для положительного входного операнда и 0x800...0 - для отрицательного. В случае беззнакового насыщения (включена опция (ISU)) признаком переполнения служит тот факт, что не все старшие биты нулевые, а значение насыщенного результата всегда равно 0xFF...F.

Префиксы обозначают типы операнда, включая: короткие слова (S), однобайтные слова (B), длинные слова (L), квадрослова (Q). Суффиксы d и q обозначают размер регистров (см. раздел «Регистры регистровых файлов»).

Флаги состояния

AZ Установлен, если все биты результата равны 0;

AN Равен MSB результата;

AV(AOS) Установлен, если результат больше максимального сжатого числа (или, в случае опции (ISU), если результат отрицательный); в противном случае сброшен;

AC Сброшен.

Опции

( ) Дробное с округлением;

(T) Дробное с усечением;
(U) Беззнаковое;
(I) Целое знаковое, без насыщения;
(IS) Целое знаковое, с насыщением;
(ISU) Целое беззнаковое, с насыщением;
(NF) Не обновлять флаги (состояния).

Примеры

{X|Y|XY} SRs = COMPACT Rmd; см. рисунок 155.

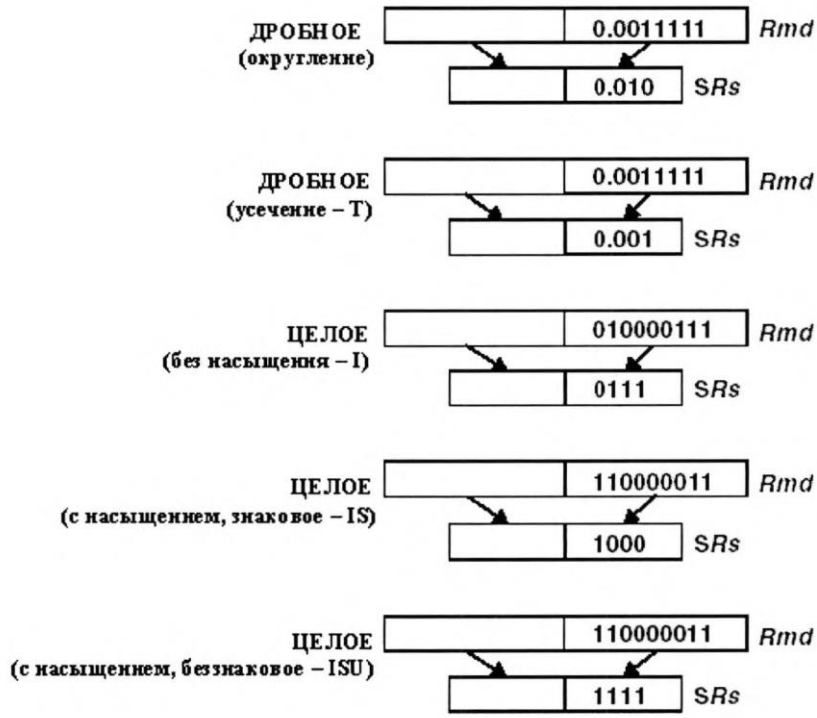


Рисунок 155 – Опции команды COMPACT

{x|y|xy}BRs = compact SRmd; см. рисунок 156.

Рисунок 156 – Сжатие коротких слов в однобайтные слова

{X|Y|XY}SRs = COMPACT Rmd + Rnd; см. рисунок 157.
{X|Y|XY}BRsq = COMPACT SRmd + SRnd;

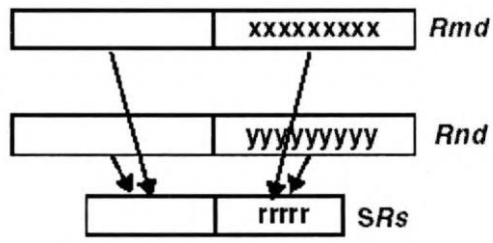


Рисунок 157 – Сложение и сжатие обычных слов в короткие слова

{X|Y|XY} BRs = COMPACT SRmd + SRnd; см. рисунок 158.

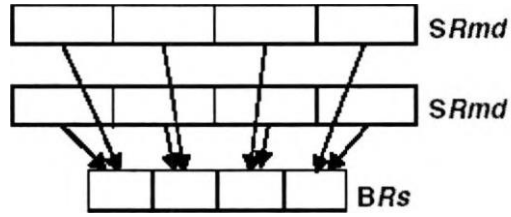


Рисунок 158 – Сложение и сжатие коротких слов в однобайтные слова

10.1.22 *Слияние*

Синтаксис

{X|Y|XY}*BRsd* = MERGE *Rm, Rn* {(NF)};

{X|Y|XY}*BRsq* = MERGE *Rmd, Rnd* {(NF)} ;

{X|Y|XY}*SRsd* = MERGE *Rm, Rn* {(NF)} ;

{X|Y|XY}*SRsq* = MERGE *Rmd, Rnd* {(NF)} ;

Действие

Эта команда объединяет (переставляет) операнды в регистрах *Rm* и *Rs.* Результат помещается в регистр *Rs* (рисунок 159).

Префиксы B и S означают однобайтные и короткие типы операндов соответственно, суффиксы d и q означают размер регистров (см. раздел «Регистры регистровых файлов»).

Флаги состояния

AZ Установлен, если все биты результата равны 0;
AN Равен MSB результата;
AV (AOS) Сброшен (не изменяется);
AC Сброшен.

Опции

(NF) Не обновлять флаги (состояния)

Пример

XBR3:2 = MERGE R0, R1;;

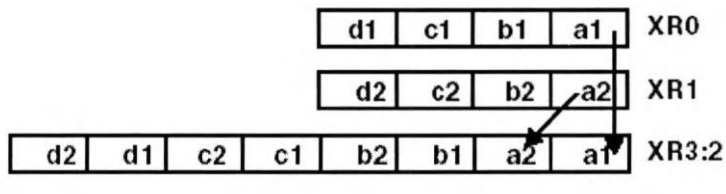


Рисунок 159 – Выполнение команды слияния

Команда MERGE может быть использована для перестановки коротких и однобайтных операндов путём повторного слияния результатов.

10.1.23 *Переставить (однобайтные слова)*

Синтаксис

{X|Y|XY}*Rsd* = PERMUTE *(Rmd, Rn)* {(NF)} ;

Действие

Результат, *Rsd,* составлен из байтов первого операнда *Rmd,* в порядке, указанном в управляющем слове в *Rn.*

Rn разбивается на 8 полубайтов (4-битных полей), *Rsd* разбивается на 8 байтов. Каждый полубайт в *Rn* управляет соответствующим по номеру байтом в *Rsd.* Например, полубайт 2 в Rn (биты 11:8) соответствует байту 2 в *Rsd* (биты 23:16). Значение каждого полубайта управляющего слова указывает, какой байт *Rmd* будет записан в байт в *Rsd,* соответствующий полубайту.

Расшифровка значений полубайтов в Rn:

b#0000: Выбрать байт 0 - биты 7:0 младшего слова
b#0001: Выбрать байт 1 - биты 15:8 младшего слова
b#0010: Выбрать байт 2 - биты 23:16 младшего слова
b#0011: Выбрать байт 3 - биты 31:24 младшего слова
b#0100: Выбрать байт 4 - биты 7:0 старшего слова
b#0101: Выбрать байт 5 - биты 15:8 старшего слова
b#0110: Выбрать байт 6 - биты 23:16 старшего слова
b#0111: Выбрать байт 7 - биты 31:24 старшего слова
b#1XXX: Зарезервировано

Различные поля в Rn могут указывать на один и тот же байт в *Rmd.* В этом случае один байт *Rmd* будет дважды вставлен в *Rsd.*

Флаги состояния

Не влияет на флаги состояния.

Опции

(NF) Не обновлять флаги (состояния)

Пример

R1:0 = 0x01234567 89ABCDEF;; R4 = 0x03172454;;
R7:6 = PERMUTE (R1:0, R4);;
Результат в R7:6 равен 0xEF8 9CD01 AB674 5 67.

10.1.24 *Переставить (короткие слова)*

Синтаксис

{X|Y|XY} *Rsq* = PERMUTE (*Rmd, -Rmd, Rn)* {(NF)} ;

Действие

Результат, Rsq, составлен из коротких слов первого операнда Rmd и дополнений к коротким словам: -Rmd. Порядок коротких слов указывается в управляющем слове в Rn.

Rn разбивается на 8 полубайтов (4-битных полей), а Rsq разбивается на 8 коротких слов. Каждый полубайт в Rn управляет соответствующим коротким словом в Rsq (например, полубайт 2 в Rn, биты 11:8, соответствует короткому слову 2 в Rsq (биты 47:32)). Значение каждого полубайта управляющего слова указывает, какой байт Rmd будет записан в байт в Rsq, соответствующий полубайту.

Расшифровка значений полубайтов в Rn:

b#0000: Выбрать короткое слово 0 - биты 15:0 Rmd
b#0001: Выбрать короткое слово 1 - биты 31:16 Rmd
b#0010: Выбрать короткое слово 2 - биты 47:32 Rmd
b#0011: Выбрать короткое слово 3 - биты 63:48 Rmd

b#0100: Выбрать короткое слово 0 - дополнение битов 15:0 Rmd
b#0101: Выбрать короткое слово 1 - дополнение битов 31:16 Rmd
b#0110: Выбрать короткое слово 2 - дополнение битов 47:32 Rmd
b#0111: Выбрать короткое слово 3 - дополнение битов 63:48 Rmd

b#1XXX: Зарезервировано

Различные полубайты в *Rn* могут указывать на одно и то же короткое слово в *Rm.* В этом случае одно короткое слово *Rmd* будет дважды вставлено в *Rsq.*

Эта команда не может быть выполняться параллельно с некоторыми командами умножителя, так как команды ALU с квадрословными результатами используют ресурсы шины умножителя.

Флаги состояния

AV Переполнение (AV) вычисляется для каждого компонента, установлено, если одно из коротких слов в Rmd, выбранное хотя бы одним из полей Rn, равно минимальному отрицательному числу (0x8000);

AOS Установлен, если происходит переполнение и очищается только явной записью в регистр X/YSTAT.

Опции

(NF) Не обновлять флаги (состояния)

Пример

R1:0 = 0x01234567 89ABCDEF;; /* -R1:0 = 0xFEDDBA99 76553211*/ R4 = 0x03147623;;
R11:8 = PERMUTE (R1:0, -R1:0, R4);;
Результат в R11: *8 равен* 0xCDEF0123 89AB3211 FEDDBA99 45670123.

10.1.25 *Сдвиг вправо на заданное число бит с упаковкой в меньший формат представления*

Синтаксис

{X|Y|XY} {B|S}Rsd = COMPACT {S}Rm by <Imm> {({U}{R})};

Действие

Команда выполняет следующие действия:

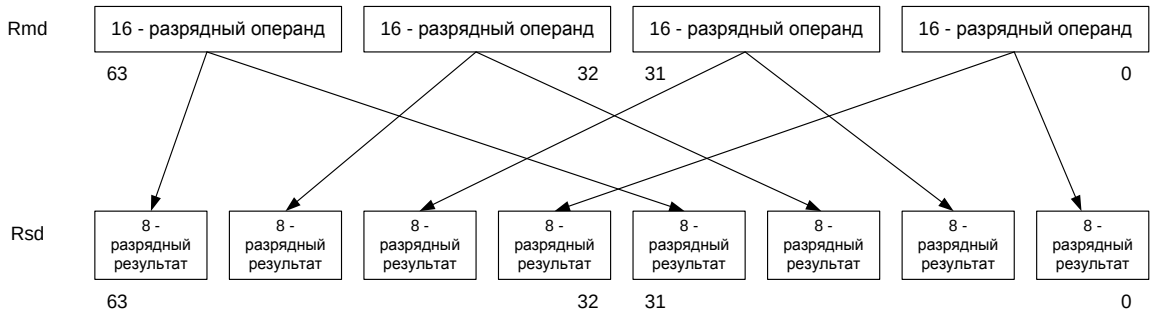
- сдвиг операнда вправо на заданное число Imm (значение в поле команды);
- округление результата (если используется опция R);
- если результат отрицательный, происходит его обнуление;
- выполнение упаковки с насыщением результата в меньший формат представления.

Команда может использоваться для преобразования формата данных из 16-ти разрядного представления в 8-ми разрядное, а также из 32-ти разрядного в 16-ти разрядное. При преобразовании 16 бит -> 8 бит регистр Rmd содержит четыре 16-ти разрядных числа, в результате преобразования получается четыре байта. При преобразовании 32 бит -> 16 бит регистр Rmd содержит два 32-разрядных слова, в результате преобразования получается два 16-ти разрядных слова. Сдвиг исходного операнда может быть логическим (расширение нулем), или арифметическим (расширение знаком).

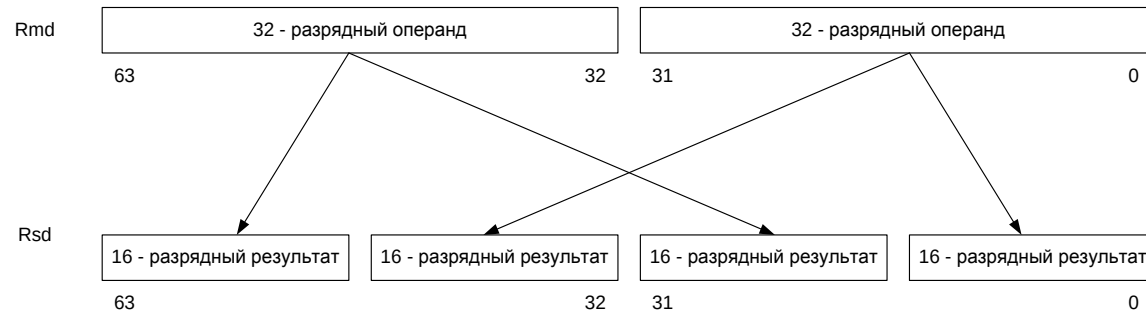
В данной команде регистр-приемник является двойным. Однако, результат работы операции всегда помещается в 32-разрядное слово. Это слово попадает в оба регистра, составляющих Rsd.

Значение константы Imm должно лежать в пределах от 0 до 31 для преобразования 16-ти разрядных слов, и от 0 до 63 для преобразования 32-разрядных слов.

Выполнение операции над 16-ти разрядными операндами:
{X|Y|XY} {B|S}Rsd = COMPACT {S}Rm by <Imm5> {({U}{R})};



Выполнение операции над 32-х разрядными операндами:
{X|Y|XY} {B|S}Rsd = COMPACT {S}Rm by <Imm7> {({U}{R})};



Флаги состояния

Бит 30 кода исполнительной команды должен быть всегда установлен, что означает выполнение команды с опцией (NF), вне зависимости от того присутствует опция в командной строке или нет. Это означает отсутствие обновления флагов состояния. При условном исполнении, опция (NF) должна присутствовать в коде условной команды.

Опции

(R) Округление результата после сдвига до ближайшего четного;
(U) Беззнаковое.

Ограничения:

Инструкция всегда занимает два слота.

10.1.26 *Знаковое расширение с вычитанием*

Синтаксис

{X|Y|XY} {B|S}Rsd = EXPAND {B|S}Rm - {B|S}Rn (ZF);

{X|Y|XY} {B|S|L}Rsq = EXPAND {B|S}Rm - {B|S}Rn (ZF);

Действие

Команда выполняет следующие действия:

- преобразование операндов в расширенное представление (расширение нулем);
- вычитание операндов в расширенном представлении;
- результат записывается в регистр – приемник.

Операнды всегда рассматриваются как беззнаковые целые 8-ми или 16-ти разрядные числа. Результат операции всегда целый знаковый. 8-ми разрядные операнды расширяются до 16-ти разрядных, 16-ти разрядные – до 32-разрядных.

{X|Y|XY} Rsd = EXPAND SRm - SRn (ZF); Расширяет два 16-разрядных числа в два 32-разрядных числа.

{X|Y|XY} Rsq = EXPAND SRmd - SRnd (ZF); Расширяет четыре 16-разрядных числа в четыре 32-разрядных числа.

{X|Y|XY} SRsd = EXPAND BRm - BRn (ZF); Расширяет четыре байта в четыре 16-разрядных числа.

{X|Y|XY} SRsq = EXPAND BRmd - BRnd (ZF); Расширяет восемь байт в восемь 16-разрядных чисел.

Префикс S означает короткий (16 бит) тип операндов, префикс B –байтовые операнды. Суффиксы d и q обозначают размер регистров.

Флаги состояния

Бит 30 кода исполнительной команды должен быть всегда установлен, что означает выполнение команды с опцией (ZF), вне зависимости от того присутствует опция в командной строке или нет. Это означает отсутствие обновления флагов состояния. При условном исполнении, опция (NF) должна присутствовать в коде условной команды.

Опции

(ZF) Заполнение нулями

Ограничения:

Инструкция всегда занимает два слота.

10.1.27 *Отсечение по границе до положительных значений*

Синтаксис

{X|Y|XY} {S|B}Rs = CLIPZ Rm by Rn;

{X|Y|XY} {L|S|B}Rsd = CLIPZ Rmd by Rnd;

Действие

Команда действует схожим образом с командой clip, с тем отличием, что отрицательный результат принудительно устанавливается равным 0. Команда clip позволяет усечь исходное число до интервала [K,-K]. Команда clipz позволяет выполнить аналогичное действие для интервала [K,0]. Если абсолютное значение знакового операнда в регистре Rm меньше абсолютного значения операнда в регистре Rn, команда возвращает Rm, если значение Rm неотрицательно, и 0, если значение Rm отрицательно. В противном случае возвращает |Rn|, если значение Rm неотрицательно, и 0, если значение Rm отрицательно. Результат операции помещается в регистр Rs.

Префиксы L, S и B указывают тип операнда. Префикс L означает длинный (64 бита) тип операнда, префикс S означает короткий (16 бит) тип операндов, префикс B – байтовые операнды. Суффикс d обозначает двойной размер регистров.

Ограничения:

Инструкция всегда занимает два слота.

10.1.28 *Сложение/вычитание с делением пополам с округлением к бесконечности*

Синтаксис

{X|Y|XY} {B|S} Rs = ROUNDUP (Rm +|- Rn)/2 {(U|NF)}

{X|Y|XY} {L|S|B} Rsd = ROUNDUP (Rmd +|- Rnd)/2 {(U|NF)}

Действие

Команда действует схожим образом с обычной командой сложения и деления пополам, за исключением того, что округление производится не к ближайшему

четному, а к + бесконечности. При выполнении команды к промежуточному результату прибавляется 1, а затем выполняется сдвиг вправо (деление на 2).

Флаги состояния

Бит 30 кода исполнительной команды должен быть всегда установлен, что означает выполнение команды с опцией (NF), вне зависимости от того присутствует опция в командной строке или нет. Это означает отсутствие обновления флагов состояния. При условном исполнении, опция (NF) должна присутствовать в коде условной команды.

Опции

( ) Знаковая операция, округление к + бесконечности;

(U) Беззнаковая операция, округление к + бесконечности;

(NF) Не обновлять флаги (состояния).

Ограничения:

Инструкция всегда занимает два слота.

10.1.29 *Поэлементное векторное сравнение*

Синтаксис

{X|Y|XY} {S|B} Rs = SETMASK Rm {EQ}|{GE}|{GT} Rn {(U|NF)};

{X|Y|XY} {S|B|L} Rsd = SETMASK Rm {EQ}|{GE}|{GT} Rn {(U|NF)};

Действие

Команда векторного сравнения выполняет поэлементное сравнение векторов в регистрах Rm, Rn и формируют битовые маски на основе результата. Размер элементов определяется префиксом *L*|*S*|*B* регистра – приемника.

Параметр *EQ*: Если значение элемента в регистре Rm *равно* значению элемента в регистре Rn, результатом операции будет число той же разрядности, что и исходные операнды, состоящее из всех 1. В противном случае значение числа будет равно 0.

Параметр *GE*: Если значение элемента в регистре Rm *больше либо равно*, чем значение элемента в регистре Rn, результатом операции будет число той же разрядности, что и исходные операнды, состоящее из всех 1. В противном случае значение числа будет равно 0.

Параметр *GT*: Если значение элемента в регистре Rm *больше*, чем значение элемента в регистре Rn, результатом операции будет число той же разрядности, что и исходные операнды, состоящее из всех 1. В противном случае значение числа будет равно 0.

Иными словами, команды векторного сравнения формируют битовые маски для дальнейшей векторной обработки данных.

Флаги состояния

Бит 30 кода исполнительной команды должен быть всегда установлен, что означает выполнение команды с опцией (NF), вне зависимости от того присутствует опция в командной строке или нет. Это означает отсутствие обновления флагов состояния. При условном исполнении, опция (NF) должна присутствовать в коде условной команды.

Опции

(U) Операнды обрабатываются, как беззнаковые;

(NF) Не обновлять флаги (состояния).

Ограничения:

Инструкция всегда занимает два слота.

10.2 Команды ALU плавающей арифметики

ALU плавающей арифметики является составной частью блока АЛУ вычислительного модуля. Это устройство выполняет все арифметические операции (сложение/вычитание) процессора над данными в формате с плавающей точкой, а также выполняет различные операции преобразования данных. Особенностью данного ALU является формат обрабатываемых данных. Поддерживаются три формата данных с плавающей запятой:

- 32-битный формат одинарной точности согласно стандарта IEEE;
- 40-битный формат расширенной точности;
- 64-битный формат двойной точности согласно стандарта IEEE (версия 2 процессора).

Остановимся более подробно на поддержке операций двойной точности. Поддержка данного формата данных реализована только во второй версии процессора. Формат расширенной и двойной точности использует 64-битные регистры. Коды команд и мнемоника команд расширенной и двойной точности одинаковы. Выбор варианта используемого формата определяется специальным битом регистра SQCTL устройства управления. Поддержка расширенного (40 бит) формата сохраняется только для обеспечения программной совместимости. Данный формат сохраняет размер экспоненты одинарной точности (8 бит), но имеет увеличенный размер мантиссы (31 бит). Использование формата двойной точности согласно стандарта IEEE, обеспечивает то же быстродействие, тот же объем памяти для хранения данных, при значительно увеличенном диапазоне представления данных. В описании команд, везде, где имеется ссылка на команды расширенной точности, нужно иметь в виду, что под этими командами подразумевается либо 40-разрядный формат расширенной точности, либо 64-разрядный формат стандартной двойной точности. Форматы данных подробно описаны в главе 2 «Регистры вычислительных блоков».

Под денормализованными числами понимаются числа в формате с плавающей запятой у которых значение экспоненты равно нулю, а значение мантиссы не нулевое. В отступление от стандарта IEEE, процессор всегда рассматривает такие операнды как ноль. Если в результате операции получается денормализованное число, процессор преобразует его в ноль с установкой соответствующего флага.

В описании команд часто упоминается смещение экспоненты для чисел одинарной и расширенной (40 бит) точности. Необходимо иметь в виду, что подобное смещение для чисел одинарной точности равно 127, а для двойной точности 1023. Также имеется упоминание на число NORM.MAX – максимальное нормализованное число. Оно представляет собой число, у которого все биты мантисы равны 1, а экспонента равна максимальному значению минус 1 (0xFE для одинарной и 40-бит точности, 0x7FE для двойной)

10.2.1 *Сложение/Вычитание (плавающая арифметика)*

Синтаксис

{X|Y|XY}F*Rs* = *Rm* +|- *Rn* {({T}{NF})} ;

{X|Y|XY}F*Rsd* = *Rmd* + |- *Rnd* {({T}{NF})} ;

Действие

Эти команды складывают или вычитают операнды в формате с плавающей точкой в регистрах *Rm* и *Rn.* Нормализованный результат помещается в регистр *FRs.* Суффикс *d* обозначает расширенный размер операнда - см. раздел «Регистры регистровых файлов».

Округление производится к ближайшему (IEEE) или с усечением, выбор между режимами определяется опцией (T). В случае переполнения в округлённом

результате возвращается ±бесконечность (округление к ближайшему) или +NORM.MAX (округление к нулю). Денормализованный округлённый результат сбрасывается к ±нулю. Денормализованные входные данные сбрасываются к ±нулю. NAN на входе приводит к результату со всеми битовыми разрядами, равными 1.

Флаги состояния

AZ Установлен, если округлённый результат денормализован или равен ±нулю;

AUS Установлен, если округлённый результат денормализован; иначе не сбрасывается;

AN Установлен, если результат отрицателен;

AV Установлен, если округление вызвало переполнение;

AVS Установлен, если округление вызвало переполнение; иначе не сбрасывается;

AC Сброшен;

AI Установлен, если какой-либо входной операнд есть NAN, либо если входные операнды - бесконечности разных (для сложения) или одного (для вычитания) знаков;

AIS Установлен, если какой-либо входной операнд есть NAN, либо если входные операнды - бесконечности разных (для сложения) или одного (для вычитания) знаков; иначе не сбрасывается.

Опции

( ) Округление;

(T) Усечение;

(NF) Не обновлять флаги (состояния).

Примеры

```
XFR0 = (R1 + R2)/2 ;; /* Вычислительный блок X, с округлением (по умолчанию) */
FR0 = (R1 + R2)/2 (T) ;; /* SIMD операция с усечением */
XFR1:0 = (R3:2 + R5:4)/2 ;; /* Сложение с делением пополам операндов повышенной точности */

YFR0 = (R1 - R2)/2 (T) ;; /* Y Вычислительный блок Y, с усечением */ FR0 = (R1 - R2)/2 ;; /* SIMD операция с округлением (по умолчанию) */ XFR1:0 = (R3:2 - R5:4)/2;; /* Вычитание с делением пополам операндов повышенной точности */
```

10.2.2 *Сложение/вычитание с делением пополам (плавающая арифметика)*

Синтаксис

{X|Y|XY}F*Rs = (Rm* +|- *Rn)/2* {({T}{NF})} ;

{X|Y|XY}F*Rsd = (Rmd* +|- *Rnd*)/2 {({T}{NF})} ;

Действие

Эти команды складывают или вычитают операнды в формате с плавающей точкой в регистрах *Rm* и *Rn,* делят результат на два, декрементируя показатель результата до округления. Нормализованный результат помещается в регистр F*Rs.* Суффикс d обозначает расширенный размер операнда - см. раздел «Регистры регистровых файлов».

Округление производится к ближайшему (IEEE) или с усечением, выбор между режимами определяется опцией (T). В случае переполнения в округлённом результате возвращается ±бесконечность (округление к ближайшему) или +NORM.MAX (округление к нулю). Денормализованный округлённый результат сбрасывается к ±нулю. Денормализованные входные данные сбрасываются к ±нулю. NAN на входе приводит к результату со всеми битовыми разрядами, равными 1.

Флаги состояния

AZ Установлен, если округлённый результат денормализован или равен ±нулю;

AUS Установлен, если округлённый результат денормализован; иначе не сбрасывается;

AN Установлен, если результат отрицателен;

AV Установлен, если округление вызвало переполнение;

AVS Установлен, если округление вызвало переполнение; иначе не сбрасывается;

AC Сброшен;

AI Установлен, если какой-либо входной операнд есть NAN, либо если входные операнды - бесконечности разных (для сложения) или одного (для вычитания) знаков;

AIS Установлен, если какой-либо входной операнд есть NAN, либо если входные операнды - бесконечности разных (для сложения) или одного (для вычитания) знаков; иначе не сбрасывается.

Опции

( ) Округление;

(T) Усечение;

(NF) Не обновлять флаги (состояния).

Примеры

```
XFR0 = (R1 + R2)/2 ;; /* Вычислительный блок X, с округлением (по умолчанию) */
FR0 = (R1 + R2)/2 (T) ;; /* SIMD операция с усечением */
XFR1:0 = (R3:2 + R5:4)/2 ;; /* Сложение с делением пополам операндов повышенной точности */

YFR0 = (R1 - R2)/2 (T) ;; /* Y Вычислительный блок Y, с усечением */ FR0 = (R1 - R2)/2 ;; /* SIMD операция с округлением (по умолчанию) */ XFR1:0 = (R3:2 - R5:4)/2;; /* Вычитание с делением пополам операндов повышенной точности */
```

10.2.3 *Максимум/Минимум (плавающая арифметика)*

Синтаксис

{X|Y|XY}F*Rs* = MAX|MIN (Rm, *Rn)* {(NF)} ;

{X|Y|XY}F*Rsd* = MAX|MIN *(Rmd, Rnd)* { (NF)} ;

Действие

Эти команды возвращают максимум (больший) или минимум (меньший) из операндов в формате с плавающей точкой в регистрах Rm и Rn. Результат помещается в регистр FRs. Суффикс d обозначает расширенный размер операнда - см. раздел «Регистры регистровых файлов».

NAN на входе приводит к результату со всеми битовыми разрядами, равными 1. MAX от (+нуля и -нуля) возвращает +ноль. MIN от (+нуля и -нуля) возвращает -ноль. Денормализованные входные данные сбрасываются к ±нулю.

Флаги состояния

AZ Установлен, если результат равен ±нулю;

AUS Не сбрасывается;

AN Установлен, если результат отрицателен;

AV Сброшен;

AVS Не сбрасывается; сохраняет своё предыдущее значение;

AC Сброшен;

AI Установлен, если какой-либо входной операнд есть NAN;

AI S Установлен, если какой-либо входной операнд есть NAN; иначе не сбрасывается.

Опции

(N F) Не обновлять флаги (состояния)

Примеры

XFR0 = MAX(R1,R2);; /* вычислительный блок X */ YFR1:0 = MAX(R3:2, R5:4) ;; /* вычислительный блок Y, операнды повышенной точности, с округлением (по умолчанию) */
XFR0 = MIN(R1,R2) ;; /*вычислительный блок X */
YFR1:0 = MIN(R3:2, R5:4) /*вычислительный блок Y, операнды повышенной точности, с округлением (по умолчанию) */

10.2.4 *Абсолютное значение суммы или разности (плавающая арифметика)*

Синтаксис

{X|Y|XY}F *Rs* = ABS *Rm* {(NF)} ;

{X|Y|XY}F *Rsd* = ABS *Rmd* { (NF) } ;

{X|Y|XY}F *Rs* = ABS *(Rm* +|- *Rn)* {({T}{NF})} ;

{X|Y|XY}F *Rsd* = ABS *(Rmd* +|- *Rnd)* {({T}{NF})} ;

Действие

Эти команды выполняют операцию сложения или вычитания над операндами в формате с плавающей точкой в регистрах Rm и Rn, затем абсолютное значение нормализованного результата помещается в регистр FRs. Если операнд Rn опущен, то выполняется операция взятия абсолютного значения от Rm. Суффикс d обозначает расширенный размер операнда - см. раздел «Регистры регистровых файлов».

Округление производится либо к ближайшему (IEEE), либо с усечением, выбор между режимами определяется опцией (T). В случае переполнения при округлении возвращается +бесконечность (округление к ближайшему) или +NORM. MAX (округление к нулю). При округлении денормализованного результата происходит сброс к +нулю. Денормализованные входные данные сбрасываются к +нулю. NAN на входе приводит к результату со всеми битовыми разрядами, равными 1.

Флаги состояния

AZ Установлен, если операнд (округлённый результат сложения или вычитания) денормализован или равен ±нулю;

AUS Установлен, если операнд (округлённый результат сложения или вычитания) денормализован; иначе не сбрасывается;

AN Установлен, если результат сложения до проведения операции ABS отрицателен;

AVS Установлен, если операнд (округлённый результат сложения или вычитания) вызвал переполнение; иначе не сбрасывается;

AV Установлен, если операнд (округлённый результат сложения или вычитания) вызвал переполнение;

AC Сброшен;

AI Установлен, если какой-либо входной операнд есть NAN, либо если входные операнды - бесконечности разных (для сложения) или одного (для вычитания) знаков;

AIS Установлен, если какой-либо входной операнд есть NAN, либо если входные операнды - бесконечности разных (для сложения) или одного (для вычитания) знаков; иначе не сбрасывается.

Примеры

```
FR0 = ABS R1 ;;     /* SIMD операция ABS над XR1 и YR1 */
XFR1:0 = ABS R1:0 ;;        /* операнды повышенной точности */
XFR0 = ABS(R1 + R2) ;;   /* вычислительный блок X, с округлением (по
умолчанию) */
FR0 = ABS(R1 + R2) (T) ;;/* SIMD операция, с усечением */
XFR1:0 = ABS(R3:2 + R5:4) ;;      /* Сложение и взятие абсолютного значения
операндов повышенной точности */
XFR0 = ABS(R1 - R2) ;;   /* вычислительный блок X, с округлением (по
умолчанию) */
FR0 = ABS(R1 - R2)        (T)      ;;        /* SIMD операция, с усечением */
XFR1:0 = ABS(R3:2 — R5:4) ;;    /* Вычитание и взятие абсолютного значения
операндов повышенной точности */
```

10.2.5 *Смена знака (плавающая арифметика)*

Синтаксис

{X|Y|XY}F *Rs* = - *Rm* {(NF)} ;

{X|Y|XY}F *Rsd* = - *Rmd* {(NF)} ;

Действие

Эта команда инвертирует знаковый бит операнда в формате с плавающей точкой в регистре *Rm.* Инвертированный результат помещается в регистр *FRs.* Суффикс d обозначает расширенный размер операнда - см. раздел «Регистры регистровых файлов».

Денормализованные входные данные сбрасываются к ±нулю (знак инвертирован по сравнению со знаком *Rm).* NAN на входе приводит к результату со всеми битовыми разрядами, равными 1.

Флаги состояния

AZ Установлен, если результат равен ±нулю или денормализован;
AUS Не сбрасывается;
AN Установлен, если результат отрицателен;
AV Сброшен;
AVS Не сбрасывается;
AC Сброшен;
AI Установлен, если входной операнд NAN;
AIS Установлен, если входной операнд NAN; иначе не сбрасывается.

Опции

(NF) Не обновлять флаги (состояния)

Примеры

```
XFR0 = - R1 ;;/* смена знака, вычислительный блок X */
YFR1:0 = - R3:2 ;;     /* операнды повышенной точности */
```

10.2.6 *Сравнение (плавающая арифметика)*

Синтаксис

{X|Y|XY}FCOMP *(Rm, Rn)* ;

{X|Y|XY}FCOMP *(Rmd, Rnd) ;*

Действие

Эта команда сравнивает значение операнда в формате с плавающей точкой в регистре *Rm* со значением операнда в формате с плавающей точкой в регистре *Rn.*

Денормализованные входные данные сбрасываются к нулю. Суффикс d обозначает расширенный размер операнда - см. раздел «Регистры регистровых файлов».

Команда устанавливает флаг AZ если два операнда равны, флаг AN - если операнд в регистре *Rm* меньше операнда в регистре *Rn.*

Флаги состояния

AZ Установлен, если Rm=Rn и ни один из Rm или Rn не NAN;
AUS Не сбрасывается;
AN Установлен, если Rm<Rn и ни один из Rm или Rn не NAN;
AV Сброшен;
AVS Не сбрасывается;
AC Сброшен;
AI Установлен, если какой-либо входной операнд есть NAN;
AIS Установлен, если какой-либо входной операнд есть NAN; иначе не сбрасывается.

Обеспечение совместимости с IEEE: после выполнения FCOMP программист может использовать предикаты ALT (результат ALU меньше чем) или ALE (результат ALU меньше чем, либо равен) - см. раздел «ALU-предикаты». В обоих случаях значение предиката будет ложным, если хотя бы один операнд равен NAN. В некоторых случаях предикаты с отрицанием также могут давать значение «ложь», если хотя бы один операнд равен NAN. В таких случаях программист *не* должен использовать эти предикаты с отрицанием (NALT и NALE). Вместо этого следует поменять местами операнды и использовать предикаты без отрицания предикаты: ALT вместо NALE или ALE вместо NALT.

Примеры

```
FCOMP(R5,R8);;
IF ALT, JUMP my_label;;
```

Значение предиката истинно, если R5<R8. Если значения в регистрах равны, R5>R8 или хотя бы одно из них есть NAN, то значение предиката ложно.

При использовании предиката NALT в случае, если R5<R8, будет получено значение «ложь».

Во всех остальных случаях, в т.ч. с NAN на входе, значение предиката будет «истина». Того же результата, но со значением «ложь» в случае NAN на входе, можно добиться, поменяв местами операнды FCOMP и поменяв ALT на ALE (или наоборот):

```
FCOMP(R8,R5);;
IF ALE, JUMP my_other_label;;
```

10.2.7 *Преобразование из формата с плавающей точкой к формату с фиксированной точкой*

Синтаксис

{X|Y|XY}*Rs* = FIX *FRm*|*FRmd* {BY Rn} {({T}{NF})} ;

Действие

Эти команды прибавляют операнд в дополнительном представлении в регистре *Rn* к порядку операнда в формате с плавающей точкой в регистре *Rm,* а затем преобразуют результат в 32-битное целое число в формате с фиксированной точкой в дополнительном представлении. Если операнд *Rn* опущен, то производится простое преобразование формата операнда в *Rm.* Операнд в формате с плавающей точкой может быть округлён или усечён. Результата помещается в регистр *Rs.* Суффикс d обозначает расширенный размер операнда - см. раздел «Регистры регистровых файлов».

Округление производится к ближайшему (IEEE) или с усечением, выбор между режимами определяется опцией (T). NAN на входе приводит к результату со

всеми битовыми разрядами, равными 1. Если операция сложения с порядком приводит к числу, настолько малому, что оно непредставимо в плавающем формате, а также если входное значение равно нулю и денормализовано, то возвращается ноль. В результате переполнения всегда возвращается значение после знакового насыщения - 0x7FFFFFFF для положительного результата и 0x80000000 - для отрицательного.

Флаги состояния

AZ Установлен, если результат с фиксированной точкой равен 0;

AUS Установлен, если результат до округления не превосходит по модулю 0.5 и отличен от нуля; иначе не сбрасывается;

AN Установлен, если результат с фиксированной точкой отрицателен;

AV Установлен, если порядок результата с плавающей точкой больше или равен 158; за тем исключением, когда операнд отрицателен, сумма равна 158, а мантисса состоит только из нулей[1];

AVS Установлен, если порядок результата с плавающей точкой больше или равен 158; за тем исключением, когда операнд отрицателен, сумма равна 158, а мантисса состоит только из нулей; иначе не сбрасывается;

AC Сброшен;

AI Установлен, если входной операнд есть NAN;

AIS Установлен, если входной операнд есть NAN; иначе не сбрасывается.

Пример

```
YR0 = FIX YFR1 (T) ;; /* вычислительный блок X, с усечением */
XR0 = FIX XFR3:2 ;; /* операнды повышенной точности, с округлением (по
умолчанию) */
YR0 = FIX YFR1 BY R2 (T) ;; /* вычислительный блок Y, с усечением */
XR0 = FIX XFR3:2 BY R4 ;; /* операнды повышенной точности, с округлением (по
умолчанию) */
```

10.2.8 *Преобразование из формата с фиксированной точкой к формату с плавающей точкой*

Синтаксис

```
{X|Y|XY}FRs|FRsd = FLOAT Rm{BY Rn}{({T}{NF})};
```

Действие

Эти команды преобразуют операнд в формате с фиксированной точкой в регистре *Rm* к формату с плавающей точкой. Регистр *Rn,* если указан, обозначает масштабирующий множитель: значение в дополнительном представлении в формате с фиксированной точкой в регистре *Rn* добавляется к порядку результата в формате с плавающей точкой. Итоговый результат помещается в регистр *FRs.* Суффикс d обозначает расширенный размер операнда (см. подраздел «Регистры регистровых файлов»).

Округление производится к ближайшему (IEEE) или с усечением, выбор между режимами определяется опцией (T). Сдвиг порядка может привести к числу, непредставимому в формате с плавающей точкой - слишком большому (переполнение) ил слишком малому. В первом случае возвращается ±бесконечность (округление к ближайшему) или ±NORM.MAX (округление к нулю); во втором случае возвращается ±ноль.

[1] т.е. когда в результате выполнения команды будет получено минимальное отрицательное число, равное 0x80... 00.

Флаги состояния

AZ В случае без масштабирования, установлен, если округлённый результат равен ±нулю; в случае с масштабированием, установлен, если округлённый результат денормализован или равен ±нулю;

AUS В случае без масштабирования, не сбрасывается; в случае с масштабированием, установлен, если округлённый результат денормализован - иначе не сбрасывается;

AN Установлен, если результат отрицателен;

AV В случае без масштабирования, сброшен; в случае с масштабированием, установлен, если округлённый результат вызвал переполнение;

AVS В случае без масштабирования, не сбрасывается; в случае с масштабированием, установлен, если округление вызвало переполнение - иначе не сбрасывается;

AC Сброшен;

AI Сброшен;

AIS Не сбрасывается.

Опции

( ) Округление;

(T) Усечение;

(NF) Не обновлять флаги (состояния).

Пример

```
XFR0 = FLOAT  R1 (T) ;;          /* вычислительный блок X, с усечением */
YFR1:0 = FLOAT R1 ;;             /* операнды повышенной точности,
                                    с округлением (по умолчанию) */
XFR0 = FLOAT R1 BY R2 (T) ;; /* вычислительный блок X, с усечением */
YFR1:0 = FLOAT R1 BY R2;;    /* операнды повышенной точности, с
                                    округлением (по умолчанию) */
```

10.2.9 *Преобразование из формата с плавающей точкой одинарной точности к формату с плавающей точкой повышенной точности (64 или 40 бит)*

Синтаксис

{X|Y|XY}F *Rsd* = EXTD *Rm* {(NF)} ;

Действие

Эта команда расширяет операнд с плавающей точкой в регистре Rm до значения повышенной точности. Результат в формате с повышенной точностью помещается в регистр FRsd. Суффикс d обозначает расширенный размер операнда - см. раздел «Регистры регистровых файлов».

Денормализованные входные данные сбрасываются к ±нулю. NAN на входе приводит к результату со всеми битовыми разрядами, равными 1.

Флаги состояния

AZ Установлен, если результат равен ±нулю или денормализован;

AUS Не сбрасывается;

AN Установлен, если результат отрицателен;

AV Сброшен;

AVS Не сбрасывается;

AC Сброшен;

AI Установлен, если входной операнд NAN;

AIS Установлен, если входной операнд NAN; иначе не сбрасывается.

Опции

(NF) Не обновлять флаги (состояния)

Пример

```
XFR1:0 = EXTD R2 ;; /* вычислительный блок X, повышение точности до 64(40) бит */
```

10.2.10 *Преобразование из формата с плавающей точкой повышенной точности к формату с плавающей точкой одинарной точности*

Синтаксис

{X|Y|XY}F*Rs* = SNGL *Rmd* {({T}{NF})} ;

Действие

Эта команда переводит операнд в формате с плавающей точкой повышенной точности в (двойном) регистре *Rmd* к формату с плавающей точкой одинарной точности. Результат помещается в регистр *FRs.* Суффикс d обозначает расширенный размер операнда - см. раздел «Регистры регистровых файлов».

Округление производится к ближайшему (IEEE) или с усечением, выбор между режимами определяется опцией (T). NAN на входе приводит к результату в формате с плавающей точкой со всеми битовыми разрядами, равными 1. В случае переполнения возвращается значение ±NORM.MAX при включённой опции (T) и значение ±бесконечность в противном случае.

Флаги состояния

AZ Установлен, если результат равен ±нулю;

AUS Не сбрасывается;

AN Установлен, если результат отрицателен;

AV Установлен, если порядок равен 0xFE, и мантисса больше, чем 0x7FFFFF80 и опция (T) не включена;

AVS Установлен, если порядок равен 0xFE, мантисса больше, чем 0x7FFFFF80; иначе не сбрасывается;

AC Сброшен;

AI Установлен, если входной операнд есть NAN;

AIS Установлен, если входной операнд есть NAN; иначе не сбрасывается.

Опции

( ) Округление;

(T) Усечение;

(NF) Не обновлять флаги (состояния).

Пример

```
XFR0 = SNGL R3:2 (T) ;; /* вычислительный блок X, опция усечения */
```

10.2.11 *Отсечение по границе (плавающая арифметика)*

{X|Y|XY}F*Rs* = CLIP *Rm* BY *Rn* { (NF) } ;

{X|Y|XY}F*Rsd* = CLIP *Rmd* BY *Rnd* {(NF)} ;

Действие

Эта команда возвращает значение операнда в формате с плавающей точкой в регистре Rm, если его абсолютное значение меньше абсолютного значения операнда в формате с плавающей точкой в регистре *Rn.* В противном случае возвращает | *Rn* |, если значение в *Rm* положительно, и -| *Rn* |, если значение в *Rm* отрицательно. Результат операции помещается в регистр *FRs* (рисунок 160). Суффикс d обозначает расширенный размер операнда - см. раздел «Регистры регистровых файлов».

NAN на входе приводит к результату со всеми битовыми разрядами, равными 1. Денормализованные входные данные сбрасываются к ±нулю.

Флаги состояния

AZ Установлен, если результат равен ±нулю или денормализован;
AUS Не сбрасывается;
AN Установлен, если результат отрицателен;
AV Сброшен;
AVS Не сбрасывается;
AC Сброшен;
AI Установлен, если какой-либо входной операнд есть NAN;
AIS Установлен, если какой-либо входной операнд есть NAN; иначе не сбрасывается.

Опции

(NF) Не обновлять флаги (состояния).

Примеры

FR3 = CLIP R9 BY R0;

XSTAT: AC=0, AV=0, AN=0, AZ=0
YSTAT: AC=0, AV=0. AN=1, AZ=0

Рисунок 160 – Двойное обычное слово в вычислительных блоках X и Y

XFR0 = CLIP R1 BY R2 ;; /* вычислительный блок X */
YFR1:0 = CLIP R5:4 BY R3:2 ;; /* вычислительный блок Y, операнды повышенной точности */

10.2.12 *Копирование знака (плавающая арифметика)*

Синтаксис

{X|Y|XY}F*Rs* = *Rm* COPYSIGN *Rn* {(NF)} ;

{X|Y|XY}F*Rsd* = *Rmd* COPYSIGN *Rnd* {(NF)} ;

Действие

Эта команда копирует знак операнда в формате с плавающей точкой в регистре Rn в операнд с плавающей точкой из регистра Rm без изменения значений его порядка и мантиссы. Результат помещается в регистр FRs. Суффикс d обозначает расширенный размер операнда - см. раздел «Регистры регистровых файлов».

NAN на входе приводит к результату со всеми битовыми разрядами, равными 1. Денормализованные данные в Rm приводят к нулевому результату со знаком, скопированным из Rn.

Флаги состояния

AZ Установлен, если результат равен ±нулю или денормализован;
AUS Не сбрасывается;
AN Установлен, если результат отрицателен;

AV Сброшен;
AVS Не сбрасывается;
AC Сброшен;
AI Установлен, если какой-либо входной операнд есть NAN;
AIS Установлен, если какой-либо входной операнд есть NAN; иначе не сбрасывается.

Опции

Не обновлять флаги (состояния).

Пример

YFR0 = R1 COPYSIGN R2 ;; /* вычислительный блок Y */
XFR1:0 = R3:2 COPYSIGN R5:4 ;; /* вычислительный блок X, операнды повышенной точности */

10.2.13 *Масштабирование (плавающая арифметика)*

Синтаксис

{X|Y|XY}F *Rs* = SCALB *FRm* BY *Rn* { (NF) } ;

{X|Y|XY}F *Rsd* = SCALB F *Rmd* BY *Rn* {(NF)} ;

Действие

Эта команда масштабирует порядок операнда в формате с плавающей точкой в Rm путём его суммирования с целым числом в дополнительном представлении в формате с фиксированной точкой в регистре Rn. Масштабированный результат в формате с плавающей точкой помещается в регистр FRs. Суффикс d обозначает расширенный размер операнда - см. раздел «Регистры регистровых файлов». Если сумма порядка FRm и Rn равна нулю, результатом операции будет ноль.

В случае переполнения результатом будет ±бесконечность. Денормализованный результат сбрасывается к ±нулю. Денормализованные входные данные сбрасываются к ±нулю. NAN на входе приводит к результату со всеми битовыми разрядами, равными 1.

Флаги состояния

AZ Установлен, если округлённый результат денормализован или равен ±нулю;
AUS Установлен, если округлённый результат денормализован; иначе не сбрасывается;
AN Установлен, если результат отрицателен;
AV Установлен, если при округлении произошло переполнение;
AVS Установлен, если при округлении произошло переполнение; иначе не сбрасывается;
AC Сброшен;
AI Установлен, если входной операнд есть NAN;
AIS Установлен, если входной операнд есть NAN; иначе не сбрасывается.

Опции

(NF) Не обновлять флаги (состояния)

Пример

FR5 = SCALB R3 BY R2;;

Если R3 = 5 *(значение с плавающей точкой;* 5=(2^3x0.625)) и R2 = 2 *(значение с фиксированной точкой),*

то R5 = 20 *(значение с плавающей точкой;*20=($2^{(3+2)}$x0.625)).
YFR1:0 = SCALB FR3:R2 BY R5;; /* операнды повышенной точности*/

10.2.14 *Передать (плавающая арифметика)*

Синтаксис

{X|Y|XY}F*Rs* = PASS *Rm* ;

{X|Y|XY}F*Rsd* = PASS *Rmd* ;

Действие

Эта команда передаёт через ALU операнд в формате с плавающей точкой из регистра *Rm* в соответствующие поля операнда с плавающей точкой в регистре FRs. Суффикс d обозначает расширенный размер операнда - см. раздел «Регистры регистровых файлов».

Денормализованные входные данные сбрасываются к ±нулю. NAN на входе приводит к результату со всеми битовыми разрядами, равными 1.

Флаги состояния

AZ Установлен, если результат равен ±нулю;
AUS Не сбрасывается;
AN Установлен, если результат отрицателен;
AV Сброшен;
AVS Не сбрасывается; сохраняет значение от предыдущей операции;
AC Сброшен;
AI Установлен, если входной операнд есть NAN;
AIS Установлен, если входной операнд есть NAN; иначе не сбрасывается.

Примеры

```
FR0= PASS R1 ;;      /* SIMD операция передачи */
XFR1:0 = PASS R3:2 ;;        /* операнды повышенной точности */
```

10.2.15 *Обратное приближение (плавающая арифметика)*

Синтаксис

{X|Y|XY}F*Rs* = RECIPS *Rm* {(NF)} ;

{X|Y|XY}F*Rsd* = RECIPS *Rmd* {(NF)} ;

Действие

Эта команда вычисляет с 8-битной точностью приближение к 1 / *Rm,* числу, обратному к значению в регистре *Rm.* Суффикс d обозначает расширенный размер операнда (см. подраздел «Регистры регистровых файлов»).

Мантисса приближения определяется по таблице с использованием семи MSB (исключая скрытый бит) мантиссы операнда в *Rm* в качестве индекса. Несмещённый порядок приближения вычисляется как дополнительный код значения несмещённого порядка операнда в *Rm* минус один. Если буквой e обозначить несмещённый порядок *Rm,* то несмещённый порядок в *FRs* равен -e-1. Входное значение ±ноль приводит к результату ±бесконечность и установке флага переполнения. Если несмещённый порядок *Rm* больше +125 (для одинарной точности), операция возвращает ±ноль. NAN на входе приводит к результату со всеми битовыми разрядами, равными 1.

Флаги состояния

AZ Установлен, если результат равен ±нулю;
AN Установлен, если результат отрицателен;
AV Установлен, если входной операнд равен ±нулю;
AVS Установлен, если входной операнд равен ±нулю; иначе не сбрасывается[1];

AC Сброшен;
AI Установлен, если входной операнд NAN;
AIS Установлен, если входной операнд NAN; иначе не сбрасывается[2]

Опции

(NF) Не обновлять флаги состояния

Пример

/ * Этот код реализует сходящийся итеративный алгоритм деления чисел в формате с плавающей точкой. Результат получается с точностью до LSB */
.SECTION program; main:
YR0 = 10.0 ;; /* Произвольный числитель */ YR12 = 3.0 ;; /* Произвольный знаменатель */ YR11 = 2.0 ;;
R7 = R0 ;;
YFR0 = RECIPS R12 ;; /* Получить 8 бит приближения к 1/r12 */ YFR12 = R0 * R12 ;;
YFR7 = R0 * R7 ; YFR0=R11-R12 ;; YFR12 = R0* R12 ;;
YFR7 = R0 * R7 ; YFR0=R11-R12 ;;
/* Одинарная точность [получена]. Следующие три строки можно выбросить, если точность результата с +/-1 в LSB достаточна */
YFR12 = R0* R12 ;;
YFR7 = R0 * R7 ;; YFR0 = R11 - R12 ;;
/* Предыдущие три строки дают результат с точностью до LSB */ YFR0 = R0 * R7 ;; /* Выходное значение в YR0 */
endhere:
NOP ;;
IDLE ;;
JUMP endhere ;;

10.2.16 *Обратное приближение к квадратному корню (плавающая арифметика)*

Синтаксис

{X|Y|XY} F*Rs* = RSQRTS *Rm* { (NF) } ;

{X|Y|XY}F*Rsd* = RSQRTS *Rmd* {(NF)} ;

Действие

Эта команда вычисляет с 8-битной точностью приближение к $1/\sqrt{Rm}$, числу, обратному к корню квадратному из значения в регистре *Rm.* Суффикс d обозначает расширенный размер операнда (см. подраздел «Регистры регистровых файлов»).

Мантисса приближения определяется по таблице с использованием LSB несмещённого порядка операнда в *Rm,* конкатенированного с шестью MSB (исключая скрытый бит) мантиссы операнда в *Rm,* в качестве индекса. Несмещённый порядок приближения вычисляется как дополнительный код значения несмещённого порядка операнда в *Rm*, сдвинутый вправо на один бит, минус один. То есть, если буквой e обозначить несмещённый порядок *Rm,* несмещённый порядок в FRs равен –[e/2] – 1. Знак полученного приближения равен знаку входного операнда. Входное значение ±ноль приводит к результату ±бесконечность и установке флага переполнения. Входное значение +бесконечность приводит к результату +ноль. NAN или отрицательное ненулевое значение (включая -бесконечность) на входе приводят к результату со всеми битовыми разрядами, равными 1.

Флаги состояния

AZ Установлен, если результат равен нулю;
AUS Не сбрасывается;
AN Установлен, если входной операнд равен –нулю;
AV Установлен, если входной операнд равен ±нулю;
AVS Установлен, если входной операнд равен ±нулю; иначе не сбрасывается;

AC Сброшен;

AI Установлен, если входной операнд есть NAN или отрицательное ненулевое число;

AI S Установлен, если входной операнд есть NAN или отрицательное ненулевое число; иначе не сбрасывается.

Опции

(NF) Не обновлять флаги (состояния)

Пример

```
/* Этот код реализует итеративный алгоритм Ньютона-Рафсона для вычисления
обратного к квадратному корню в формате с плавающей точкой. Результат
получается с точностью до LSB */
XR0 = 9.0 ;; // произвольные входные данные ;;
XR8 = 3.0 ;;
XR1 = 0.5 ;;
XFR4 = RSQRTS R0 ;;
XFR12 = R4 * R4 ;;
XFR12 = R12 * R0 ;;
XFR4 = R1 * R4 ; XFR12 = R8 - R12 ;;
XFR4 = R4 * R12 ;;
XFR12 = R4 * R4 ;;
XFR12 = R12 * R0 ;;
XFR4 = R1 * R4 ; XFR12 = R8 - R12 ;;

/* Одинарная точность [получена]. Следующие четыре строки можно выбросить, если
точность результата с +/-1 в LSB достаточна */
XFR4 = R4 * R12 ;;
XFR12 = R4 * R4 ;;
XFR12 = R12 * R0 ;;
XFR4 = R1 * R4 ; XFR12 = R8 - R12 ;;
XFR4 = R4 * R12 ;; /* обратное к квадратному корню из xr0 хранится в r4*/
endhere:
NOP ;;
IDLE ;;
JUMP endhere ;;
```

10.2.17 *Мантисса (плавающая арифметика)*

Синтаксис

{X|Y|XY}*Rs* = MANT F*Rm*|F*Rmd* {(NF)} ;

Действие

Эта команда извлекает мантиссу (биты дробной части плюс скрытый бит, исключая бит знака) из операнда в формате с плавающей точкой в регистре *Rm.* Полученная беззнаковая величина сдвигается к левому краю поля фиксированной точки и помещается в регистр *Rs.* Суффикс d обозначает расширенный размер операнда (см. подраздел «Регистры регистровых файлов».

Отметим, что флаг AN устанавливается в соответствии со знаком входного операнда.

Опции округления игнорируются, округление как таковое не производится, так как все результаты точны по своей сути. Денормализованные входные данные сбрасываются к нулю. NAN или бесконечность на входе приводят к результату со всеми битовыми разрядами, равными 1.

Флаги состояния

AZ Установлен, если результат равен нулю;

AUS Не сбрасывается;
AN Установлен, если входной операнд (FRm) отрицателен;
AV Сброшен;
AVS Не сбрасывается;
AC1 Сброшен;
AI Установлен, если входной операнд есть NAN или бесконечность;
AIS Установлен, если входной операнд есть NAN или бесконечность; иначе не сбрасывается.

Опции

(NF) Не обновлять флаги (состояния)

Примеры

```
XR0 = MANT FR1 ;; /* вычислительный блок X */
YR0 = MANT FR3:2 ;; /* вычислительный блок Y, операнды повышенной точности */
```

10.2.18 *Логарифм (плавающая арифметика)*

Синтаксис

{X|Y|XY}Rs = LOGB FRm|FRmd { ({S} {NF}) } ;

Действие

Эта команда преобразует порядок операнда в формате с плавающей точкой в регистре Rm в несмещённое значение в формате с фиксированной точкой в дополнительном представлении. Результат помещается в регистр Rs в виде целого числа. Суффикс d обозначает расширенный размер операнда (см. подраздел «Регистры регистровых файлов»).

Устранение смещения порядка производится путём вычитания 127 из значения порядка операнда в формате с плавающей точкой одинарной точности в Rm. Без опции насыщения ±бесконечность на входе приводит к результату, равному +бесконечности в формате с плавающей точкой; в противном случае ±бесконечность на входе приводит к максимальному положительному результату (0x7FFFFFFF). Без опции насыщения ±ноль на входе приводит к результату, равному -бесконечности в формате с плавающей точкой; в противном случае ±ноль на входе приводит к минимальному отрицательному результату (0x80000000). Денормализованные входные данные сбрасываются к ±нулю. NAN или бесконечность на входе приводят к результату со всеми битовыми разрядами, равными 1.

Флаги состояния

AZ Установлен, если результат в формате с фиксированной точкой равен нулю;
AUS Не сбрасывается;
AN Установлен, если результат в формате с фиксированной точкой отрицателен;
AV Установлен, если входной операнд ±бесконечность или ±ноль
AVS Установлен, если входной операнд ±бесконечность или ±ноль; иначе не сбрасывается;
AC Сброшен;
AI Установлен, если входной операнд есть NAN;
AIS Установлен, если входной операнд есть NAN; иначе не сбрасывается.

Опции

( ) Насыщение выключено;
(S) Насыщение включено;
(NF) Не обновлять флаги (состояния).

Примеры

```
XR0 = LOGB XFR1 (S) ;; /* вычислительный блок X, опция насыщения включена */
YR0 = LOGB YFR3:2 ;; /* вычислительный блок Y, операнды повышенной точности */
```

10.2.19 *Сложение/Вычитание (двойная операция, плавающая арифметика)*

Синтаксис

{X|Y|XY} FRs = Rm + Rn, FRa = Rm - Rn { (NF) } ; (двойная операция)

{X|Y|XY} FRsd = Rmd + Rnd, FRad = Rmd - Rnd { (NF) } ; (двойная операция)

Действие

Эта команда одновременно складывает и вычитает операнды в формате с плавающей точкой в регистрах *Rm* и *Rn.* Результаты помещаются в регистры FRs и FRa. Суффикс d обозначает расширенный размер операнда (см. подраздел «Регистры регистровых файлов»).

Флаги состояния

AZ Установлен, если округлённый результат денормализован или равен ±нулю;

AUS Установлен, если округлённый результат денормализован; иначе не сбрасывается;

AN Установлен, если результат отрицателен;

AV Установлен, если при округлении произошло переполнение;

AVS Установлен, если при округлении произошло переполнение; иначе не сбрасывается;

AC Сброшен;

AI Установлен, если какой-либо входной операнд есть NAN или оба входных операнда суть бесконечности любого знака;

AIS Установлен, если какой-либо входной операнд есть NAN или оба входных операнда суть бесконечности любого знака; иначе не сбрасывается.

Опции

(NF) Не обновлять флаги (состояния)

Примеры

```
XFR0 = R1 + R2, XFR3 = R1 - R2 ;; /* вычислительный блок X, сложение и
вычитание */
XFR1:0 = R3:2 + R5:4, XFR7:6 = R3:2 - R5:4 ;; /* операнды повышенной точности */
```

10.2.20 *Векторные команды ALU плавающей арифметики*

ALU плавающей арифметики выполняет базовый набор команд над операндами различной точности. Особенность базового набора – выполнение операции над одной парой операндов с формированием одного результата операции. Данный тип команд соответствует типу SISD обработки. Возможна SIMD обработка над операндами плавающей арифметики, но она осуществляется как вариант выполнения одной и той же команды в двух независимых модулях обработки X и Y. Это позволяет выполнять операцию над двумя парами операндов с генерацией двух результатов. В процессоре выполнено расширение возможностей SIMD обработки для чисел формата плавающей запятой (ПЗ) одинарной точности. В каждый из X и Y модуль добавлена возможность выполнения операции над двумя парами операндов с генерацией двух результатов. В сумме (одновременно в модуле X и Y) это дает возможность обрабатывать 4 пары операндов и получать за такт 4 результата. Все эти возможности относятся к вычислениям с ПЗ стандартной одинарной точности. Буква “F” в названии регистра указывает на операцию над числами с ПЗ. Для идентификации выполнения операции в одном из модулей над двумя парами операндов с ПЗ одинарной точности будем записывать операнды через запятую. Введем новый синтаксис:

{X|Y|XY}SFRsd = Rmd операция Rnd (mX)(nX)(T)(NF);; (Вариант 2)

FRsd = RmH операция RnH, RmL операция RnL (Вариант 1)

Примечание – синтаксис вариант 1 считается устаревшим.

Как известно, синтаксис FRd подразумевает операцию над числами расширенной точности либо числами двойной точности. Дополнительный синтаксис FRd будет означать два числа ПЗ одинарной точности в паре регистров. В этом случае в одном вычислительном модуле становится возможным выполнение операции над двумя парами чисел ПЗ одинарной точности в одном такте. Новая операция не противоречит размеру используемых в вычислительном модуле шин. Перечень возможных операций соответствует списку обычных операций с ПЗ одинарной точности с несколькими ограничениями. В коде команды поля Rs, Rm, Rn содержат только одно число. Обычно это четный регистр пары регистров. Для поля регистра-приемника четность регистра Rs является обязательным условием. Для регистра-операнда допустимо переставлять номера регистров пары. Перед выполнением операции два операнда пары регистров будут переставлены местами.

Операция типа *FR1:0 = R3 op R5, R2 op R4* эквивалентна выполнению двух операций: *FR0 = R3 op R5, R1 = R2 op R4*. Младший (значение справа) регистр одной пары участвует в операции с младшим регистром другой пары и результат записывается в младший регистр приемника. Старший (значение слева) регистр пары участвует в операции со старшим регистром другой пары и результат записывается в старший регистр приемника. В данном примере, регистры операндов r0 и r2 младшие, а регистры r1 и r3 старшие.

Для некоторых команд существуют особенности выполнения.

Команда FR1:0 = float R1 by R2, float R0 by R2, будет использовать общий регистр шкалы R2 для обоих операндов.

Команда FR1:0 = fix R1 by R2, fix R0 by R2 будет использовать общий регистр шкалы R2 для обоих операндов.

Команда FR1:0 = scalb R1 by R2, scalb R0 by R2 будет использовать общий регистр шкалы R2 для обоих операндов.

Как известно операции АЛУ могут формировать признаки операций. Как быть в случае векторных операций? Каждая операция над своей парой операндов формирует признаки согласно описанию выполнения стандартной команды. Особенность векторной операции состоит в том, что мы имеет два набора признаков операции вместо одного. Если в коде команды имеется бит разрешения формирования признаков, то признаки от обеих операций образуют логическое ИЛИ и записываются в соответствующие биты регистра состояний.

Поскольку векторные операции всегда выполняются над парами регистров, то значение номера регистра в коде операнда должно быть четным. Регистр приемника всегда должен быть четным. Если какой-то из регистров источника имеет нечетный номер, то это приведет к перестановке операндов. При этом имеется следующая особенность.

Если номер регистра Rm нечетный, то операнды в паре регистров меняются местами перед выполнением операции. В синтаксисе Вариант 2 нечётный номер Rm задаётся с помощью опции mX.

Если номер регистра Rn нечетный, то операнды в паре регистров меняются местами перед выполнением операции, но для команд сложения и вычитания это делается особенным образом: R1.R0=>-R0.R1, т.е дополнительно выполняется изменение знака исходного операнда в младшем регистре. При работе с комплексными числами это позволяет выполнять операции с так называемыми сопряженными числами. При этом для получения нужного результата нужно использовать операции сложения либо вычитания, а в операциях одновременного сложения-вычитания правильно выбирать регистры приемников. В синтаксисе Вариант 2 нечётный номер Rnd задаётся с помощью опции nX.

Синтаксис Вариант 1 считается устаревшим. Он плох тем, что в нём нельзя выполнить команду с константой. В синтаксисе Вариант 2 эта возможность появилась. Вариант 2 приведен к стандартному виду, то есть аналогично тому, как префиксы B,S,L указывают на тип данных при обработке целых чисел, префикс SF указывает на операцию над вектором из чисел одинарной точности.

В синтаксисе Вариант 2 задание нечетного значения номера регистра Rmd или Rnd выполняется с помощью опций. Опция (mX) означает перестановку пары регистров Rmd, (nX) означает перестановку пары регистров Rnd. Использование этой опции означает, что младший бит номера регистра будет установлен в 1.

10.2.20.1 Особенности обозначений и терминов в описании команд

Регистровая пара (пара регистров) – обозначает два регистра образующих один операнд для выполнения операции. Допустимыми регистровыми парами могут быть: r1:0, r3:2, r5:4, r7:6, r9:8, r11:10, r13:12, r15:14, r17:16, r19:18, r21:20, r23:22, r25:24, r27:26, r29:28, r31:30. Для операндов-источников разрешена перестановка номеров регистров внутри допустимой пары.

Rsd, Rmd, Rnd – обозначают регистровые пары.

Rs, Rm, Rn – один из регистров (старший или младший, четный или нечетный) соответствующей регистровой пары.

Младший регистр пары – соответствует номеру регистра пары, стоящему справа. Например, для случая R5:4, регистр R4 – младший, а для случая R6:7, младшим будет регистр R7.

Старший регистр пары – соответствует номеру регистра пары, стоящему слева. Например, для случая R5:4, регистр R5 – старший, а для случая R6:7, старшим будет регистр R6.

Четный регистр пары – соответствует четному номеру регистра пары. Например, для случая R5:4, регистр R4 – четный. Для случая R6:7, четным будет регистр R6.

Нечетный регистр пары – соответствует нечетному номеру регистра пары. Например, для случая R5:4, регистр R5 – нечетный. Для случая R6:7, нечетным будет регистр R7.

Пара операндов – младший операнд регистра Rnd и младший операнд регистра Rmd, либо старший операнд регистра Rnd и старший операнд регистра Rmd. Над парой операндов выполняется операция и формируется результат. Для векторных операций имеются две пары операндов и два результата.

Флаги состояния – соответствуют в описании результату выполнения стандартной операции над одной парой операндов. Поскольку выполняются сразу две операции над двумя парами операндов и формируются два набора флагов состояния (для каждой пары операндов алгоритм формирования флагов одинаковый), то итоговый набор получается как логическое ИЛИ соответствующих флагов двух наборов. Например, для операции сравнения пары регистров R1:0 и R3:2 признак равенства нулю Z будет равен 1, если R0==R2 либо R1==R3. Но не обязательно чтобы обе пары были равны.

Перестановка регистров пары – для допустимых значений пары, означает замену номеров регистров местами. Например, R1:0 меняется на R0:1. В коде команды будет нечетное значение регистра пары. Перед выполнением операции, процессор поменяет местами операнды пары регистров R1:0. Такой нестандартный синтаксис вводится для упрощения записи. В противном случае необходимо было бы добавлять опцию X (eXchange) для каждого из двух операндов.

10.2.20.2 Сложение/вычитание

Синтаксис

{X|Y|XY}SFRsd = Rmd +|- Rnd {({mX}{nX} {T}{NF})} ; (Вариант 2)

{X|Y|XY}F*Rsd = RmH* + |- *RnH, RmL* + |- *RnL,* {({T}{NF})} ; (Вариант 1)

Действие

Эти команды складывают или вычитают операнды в формате с плавающей точкой в регистрах Rmd и Rnd. Нормализованный результат помещается в регистр Rsd. Операция выполняется независимо над каждой парой соответствующих регистров: младший регистр Rmd с младшим регистром Rnd и результат записывается в четный регистр Rsd, старший регистр Rmd со старшим регистром Rnd и результат записывается в нечетный регистр Rsd. Имеется особенность в выполнении операции в случае, когда значение Rn в поле команды нечетное – для старшего операнда пары выполняется смена знака на противоположный.

Округление производится к ближайшему (IEEE) или с усечением, выбор между режимами определяется опцией (T). В случае переполнения в округлённом результате возвращается ±бесконечность (округление к ближайшему) или +NORM.MAX (округление к нулю). Денормализованный округлённый результат сбрасывается к ±нулю. Денормализованные входные данные сбрасываются к ±нулю. NAN на входе приводит к результату со всеми битовыми разрядами, равными 1.

Флаги состояния

AZ Установлен, если округлённый результат денормализован или равен ±нулю;

AUS Установлен, если округлённый результат денормализован; иначе не сбрасывается;

AN Установлен, если результат отрицателен;

AV Установлен, если округление вызвало переполнение;

AVS Установлен, если округление вызвало переполнение; иначе не сбрасывается;

AC Сброшен;

AI Установлен, если какой-либо входной операнд есть NAN, либо если входные операнды - бесконечности разных (для сложения) или одного (для вычитания) знаков;

AIS Установлен, если какой-либо входной операнд есть NAN, либо если входные операнды - бесконечности разных (для сложения) или одного (для вычитания) знаков; иначе не сбрасывается.

Опции

( ) Округление;

(T) Усечение;

(NF) Не обновлять флаги (состояния);

(mX) Поменять местами регистры в паре Rmd;

(nX) Поменять местами регистры в паре Rnd.

Примеры

```
SFR1:0 = R3:2 + R5:4 (mX)   /* R1=R2+R5, R0 = R3+R4 */
```

10.2.20.3 Сложение/вычитание с делением пополам

Синтаксис

{X|Y|XY}SFRsd = (Rmd +|- Rnd)/2 {({mX}{nX} {T} {NF}) } ; (Вариант 2)

{X|Y|XY}F*Rsd = (RmH* +|- *RnH*)/2, *(RmL* +|- *RnL*)/2 {({T}{NF})} ; (Вариант 1)

Действие

Эти команды складывают или вычитают операнды в формате с плавающей точкой в регистрах Rm и Rn, делят результат на два, декрементируя показатель результата до округления. Нормализованный результат помещается в регистр *Rs.* Суффикс d обозначает выполнение операции сразу для двух пар операндов с формированием двух результатов.

Округление производится к ближайшему (IEEE) или с усечением, выбор между режимами определяется опцией (T). В случае переполнения в округлённом результате возвращается ±бесконечность (округление к ближайшему) или +NORM.MAX (округление к нулю). Денормализованный округлённый результат сбрасывается к ±нулю. Денормализованные входные данные сбрасываются к ±нулю. NAN на входе приводит к результату со всеми битовыми разрядами, равными 1.

Флаги состояния

AZ Установлен, если округлённый результат денормализован или равен ±нулю;

AUS Установлен, если округлённый результат денормализован; иначе не сбрасывается;

AN Установлен, если результат отрицателен;

AV Установлен, если округление вызвало переполнение;

AVS Установлен, если округление вызвало переполнение; иначе не сбрасывается;

AC Сброшен;

AI Установлен, если какой-либо входной операнд есть NAN, либо если входные операнды - бесконечности разных (для сложения) или одного (для вычитания) знаков;

AIS Установлен, если какой-либо входной операнд есть NAN, либо если входные операнды - бесконечности разных (для сложения) или одного (для вычитания) знаков; иначе не сбрасывается.

Опции

( ) Округление;

(T) Усечение;

(NF) Не обновлять флаги (состояния);

(mX) Поменять местами регистры в паре Rmd;

(nX) Поменять местами регистры в паре Rnd.

10.2.20.4 Максимум/Минимум

Синтаксис

{X|Y|XY}SFRsd = MAX | MIN (Rmd, Rnd) {({mX}{nX} {NF}) } ; (Вариант 2)

{X|Y|XY}F*Rsd* = MAX|MIN (*RmHm RnH),* MAX|MIN (*RmL, RnL)* { (NF)}; (Вариант 1)

Действие

Эти команды возвращают максимум (больший) или минимум (меньший) из операндов в формате с плавающей точкой в регистрах *Rm* и *Rn.* Результат помещается в регистр *Rs.* Суффикс d обозначает выполнение операции сразу для двух пар операндов с формированием двух результатов. *Для каждой пары операндов вычисляется свой максимум или минимум.*

NAN на входе приводит к результату со всеми битовыми разрядами, равными 1. MAX от (+нуля и -нуля) возвращает +ноль. MIN от (+нуля и -нуля) возвращает -ноль. Денормализованные входные данные сбрасываются к ±нулю.

Флаги состояния

AZ Установлен, если результат равен ±нулю;

AUS Не сбрасывается

AN Установлен, если результат отрицателен;

AV Сброшен;
AVS Не сбрасывается; сохраняет своё предыдущее значение;
AC Сброшен;
AI Установлен, если какой-либо входной операнд естьNAN;
AIS Установлен, если какой-либо входной операнд есть NAN; иначе не сбрасывается.

Опции

(NF) Не обновлять флаги (состояния)
(mX) Поменять местами регистры в паре Rmd
(nX) Поменять местами регистры в паре Rnd

10.2.20.5 Абсолютное значение/ Абсолютное значение суммы или разности

Синтаксис

{X|Y|XY}SFRsd = ABS Rmd {({mX}{nX} {NF}) };

{X|Y|XY}SFRsd = ABS (Rmd +|- Rnd) {({mX}{nX} {T} {NF}) } ; (Вариант 2)

{X|Y|XY}F*Rsd* = ABS (*RmH), ABS (RmL)* { (NF) } ;

{X|Y|XY}F*Rsd* = ABS *(RmH* +|- *RnH),* ABS *(RmL* +|- *RnL)* {({T}{NF})} ; (Вариант 1)

Действие

Эти команды выполняют операцию сложения или вычитания над операндами в формате с плавающей точкой в регистрах *Rm* и *Rn,* затем абсолютное значение нормализованного результата помещается в регистр *Rs.* Если операнд *Rn* отсутствует, то выполняется операция взятия абсолютного значения от *Rm.* Суффикс d обозначает выполнение операции сразу для двух пар операндов с формированием двух результатов.

Округление производится либо к ближайшему (IEEE), либо с усечением, выбор между режимами определяется опцией (T). В случае переполнения при округлении возвращается +бесконечность (округление к ближайшему) или +NORM. MAX (округление к нулю). При округлении денормализованного результата происходит сброс к +нулю. Денормализованные входные данные сбрасываются к +нулю. NAN на входе приводит к результату со всеми битовыми разрядами, равными 1.

Флаги состояния

AZ Установлен, если операнд (округлённый результат сложения или вычитания) денормализован или равен ±нулю;
AUS Установлен, если операнд (округлённый результат сложения или вычитания) денормализован; иначе не сбрасывается;
AN Установлен, если результат сложения до проведения операции ABS отрицателен;
AV Установлен, если операнд (округлённый результат сложения или вычитания) вызвал переполнение;
AVS Установлен, если операнд (округлённый результат сложения или вычитания) вызвал переполнение; иначе не сбрасывается;
AC Сброшен;
AI Установлен, если какой-либо входной операнд – это NAN, либо если входные операнды – это бесконечности разных (для сложения) или одного (для вычитания) знаков;
AIS Установлен, если какой-либо входной операнд есть NAN, либо если входные операнды - бесконечности разных (для сложения) или одного (для вычитания) знаков; иначе не сбрасывается.

Опции

( ) Округление;

(T) Усечение;
(NF) Не обновлять флаги (состояния);
(mX) Поменять местами регистры в паре Rmd;
(nX) Поменять местами регистры в паре Rnd.

10.2.20.6 Смена знака

Синтаксис

{X|Y|XY}SFRsd = - Rmd {({mX} {NF}) }; (Вариант 2)

{X|Y|XY}F*Rsd = -RmH, -RmL* {(NF)} ; (Вариант 1)

Действие

Эта команда инвертирует знаковый бит операнда в формате с плавающей точкой в регистре *Rm.* Инвертированный результат помещается в регистр *Rs.* Суффикс d обозначает выполнение операции сразу для двух операндов с формированием двух результатов.

Денормализованные входные данные сбрасываются к ±нулю (знак инвертирован по сравнению со знаком *Rm).* NAN на входе приводит к результату со всеми битовыми разрядами, равными 1.

Флаги состояния

AZ Установлен, если результат равен ±нулю или денормализован;
AUS Не сбрасывается;
AN Установлен, если результат отрицателен;
AV Сброшен;
AVS Не сбрасывается;
AC Сброшен;
AI Установлен, если входной операнд NAN;
AIS Установлен, если входной операнд NAN; иначе не сбрасывается.

Опции

(NF) Не обновлять флаги (состояния);
(mX) Поменять местами регистры в паре Rmd.

10.2.20.7 Сравнение

Синтаксис

{X|Y|XY}SFCOMP (Rmd, Rnd) {({mX}{nX}} ; (Вариант 2);

{X|Y|XY}FCOMP (RmH, RnH), FCOMP(RmL, RnL) ; (Вариант 1)

Действие

Эта команда сравнивает значения операндов в формате с плавающей точкой в регистре *Rmd* с соответствующими значениями операндов в формате с плавающей точкой в регистре *Rnd.* Денормализованные входные данные сбрасываются к нулю.

Команда устанавливает:

- флаг AZ, если два операнда любой из пар операндов равны;
- флаг AN, если операнд в регистре *Rmd* меньше соответствующего ему операнда в регистре *Rnd.*

Флаги состояния

AZ Установлен, если *Rm=Rn* и ни один из *Rm* или *Rn* не NAN;
AUS Не сбрасывается;
AN Установлен, если *Rm<Rn* и ни один из *Rm* или *Rn* не NAN;
AV Сброшен;
AVS Не сбрасывается;
AC Сброшен;

AI Установлен, если какой-либо входной операнд есть NAN;

AIS Установлен, если какой-либо входной операнд есть NAN; иначе не сбрасывается.

При выполнении VCOMP программист должен помнить, что флаги состояния образуются как логическое ИЛИ от двух операций сравнения. Вполне возможно, что лучшим решением будет использование стандартной команды FCOMP отдельно для каждой пары операндов, т.к. операция VCOMP имеет особенности.

Опции

(mX) Поменять местами регистры в паре Rmd;

(nX) Поменять местами регистры в паре Rnd.

10.2.20.8 Преобразование из формата с плавающей точкой к формату с фиксированной точкой

Синтаксис

{X|Y|XY}*Rsd* = FIX SF*Rmd* {BY *Rn*} {({mX}{T}{NF})}; (Вариант 2)

{X|Y|XY}*Rsd* = FIX F*RmH*{BY Rn}, FIX F*RmL* {BY Rn}, {({T}{NF})} ;(Вариант 1)

Действие

Если операнд *Rn* опущен, то производится простое преобразование формата двух операндов регистровой парыRmd в целочисленный формат и запись двух результатов в регистр Rsd. Если операнд Rn присутствует, то 32-разрядное целочисленное значение Rn используется одновременно для обоих операндов ПЗ регистра Rmd. В этом случае, операнд в дополнительном представлении в регистре Rn прибавляется к порядку каждого из операндов в формате с плавающей точкой в регистре Rmd, а затем полученные числа с ПЗ преобразуются в 32-битовые целые числа в формате с фиксированной точкой в дополнительном представлении. Операнд в формате с плавающей точкой может быть округлён или усечён. Результаты помещаются в регистр *Rsd.* Суффикс d обозначает выполнение операции сразу для двух операндов с формированием двух результатов.

Округление производится к ближайшему (IEEE) или с усечением, выбор между режимами определяется опцией (T). NAN на входе приводит к результату со всеми битовыми разрядами, равными 1. Если операция сложения с порядком приводит к числу, настолько малому, что оно непредставимо в плавающем формате, а также если входное значение равно нулю и денормализовано, то возвращается ноль. В результате переполнения всегда возвращается значение после знакового насыщения – 0x7FFFFFFF для положительного результата и 0x80000000 – для отрицательного.

Флаги состояния

AZ Установлен, если результат с фиксированной точкой равен 0;

AUS Установлен, если результат до округления не превосходит по модулю 0,5 и отличен от нуля; иначе не сбрасывается;

AN Установлен, если результат с фиксированной точкой отрицателен;

AV Установлен, если порядок результата с плавающей точкой больше или равен 158; за тем исключением, когда операнд отрицателен, сумма равна 158, а мантисса состоит только из нулей (т.е. когда в результате выполнения команды будет получено минимальное отрицательное число, равное 0x80... 00);

AVS Установлен, если порядок результата с плавающей точкой больше или равен 158; за тем исключением, когда операнд отрицателен, сумма равна 158, а мантисса состоит только из нулей; иначе не сбрасывается;

AC Сброшен;

AI Установлен, если входной операнд есть NAN;

AIS Установлен, если входной операнд есть NAN; иначе не сбрасывается.

Опции

( ) округление;
(T) усечение;
(NF) не обновлять флаги (состояния);
(mX) Поменять местами регистры в паре Rmd.

10.2.20.9 Преобразование из формата с фиксированной точкой к формату с плавающей точкой

Синтаксис

{X|Y|XY}SFRsd = FLOAT Rmd {BY Rn} {({mX}{T}{NF})}; (Вариант 2)

{X|Y|XY}FRsd = FLOAT RmH {BY Rn}, FLOAT RmL {BY Rn}, {({T}{NF})}; (Вариант 1)

Действие

Эта команда преобразует два числа в формате с фиксированной точкой в регистре Rmd к формату с плавающей точкой. Регистр Rn, если указан, обозначает масштабирующий множитель: значение в дополнительном представлении в формате с фиксированной точкой в регистре Rn добавляется к порядку результата в формате с плавающей точкой. Итоговый результат помещается в регистр Rsd. Отметим, что одно и тоже значение регистра Rn используется для обоих чисел регистра Rmd. Суффикс d обозначает выполнение операции сразу для двух операндов с формированием двух результатов.

Округление производится к ближайшему (IEEE) или с усечением, выбор между режимами определяется опцией (T). Сдвиг порядка может привести к числу, непредставимому в формате с плавающей точкой – слишком большому (переполнение) или слишком малому. В первом случае возвращается ±бесконечность (округление к ближайшему) или ±NORM.MAX (округление к нулю); во втором случае возвращается ±ноль.

Флаги состояния

AZ В случае без масштабирования, установлен, если округлённый результат равен ±нулю; в случае с масштабированием, установлен, если округлённый результат денормализован или равен ±нулю

AUS В случае без масштабирования, не сбрасывается; в случае с масштабированием, установлен, если округлённый результат денормализован - иначе не сбрасывается[2]

AN Установлен, если результат отрицателен

AV В случае без масштабирования, сброшен; в случае с масштабированием, установлен, если округлённый результат вызвал переполнение

AVS В случае без масштабирования, не сбрасывается; в случае с масштабированием, установлен, если округление вызвало переполнение – иначе не сбрасывается;

AC Сброшен;
AI Сброшен;
AIS Не сбрасывается.

Опции

( ) округление;
(T) усечение;
(NF) не обновлять флаги (состояния);
(mX) Поменять местами регистры в паре Rmd.

10.2.20.10 Отсечение по границе

Синтаксис

{X|Y|XY}SFRsd = CLIP Rmd BY Rnd {({mX}{nX} {NF}) }; (Вариант 2)

{X|Y|XY}F*Rsd* = CLIP *RmH* BY *RnH,* CLIP *RmL* BY *RnL* {(NF)} ; (Вариант 1)

Действие

Эта команда возвращает значение операнда в формате с плавающей точкой в регистре *Rm,* если его абсолютное значение меньше абсолютного значения операнда в формате с плавающей точкой в регистре *Rn.* В противном случае возвращает | *Rn* |, если значение в *Rm* положительно, и -| *Rn* | если значение в *Rm* отрицательно. Результат операции помещается в регистр *Rs*. Суффикс d обозначает выполнение операции сразу для двух пар операндов с формированием двух результатов.

NAN на входе приводит к результату со всеми битовыми разрядами, равными 1. Денормализованные входные данные сбрасываются к ±нулю.

Флаги состояния

AZ Установлен, если результат равен ±нулю или денормализован;
AUS Не сбрасывается;
AN Установлен, если результат отрицателен;
AV Сброшен;
AVS Не сбрасывается;
AC Сброшен;
AI Установлен, если какой-либо входной операнд есть NAN;
AIS Установлен, если какой-либо входной операнд есть NAN; иначе не сбрасывается.

Опции

(T) усечение;
(NF) не обновлять флаги (состояния);
(mX) Поменять местами регистры в паре Rmd;
(nX) Поменять местами регистры в паре Rnd.

10.2.20.11 Копирование знака

Синтаксис

{X|Y|XY}SFRsd = *Rmd* COPYSIGN *Rnd* {({mX}{nX} {NF}) }; (Вариант 2)

{X|Y|XY}F*Rsd = RmH* COPYSIGN *RnH, RmL* COPYSIGN *RnL,* {(NF)}; (Вариант 1)

Действие

Эта команда копирует знак операнда в формате с плавающей точкой в регистре *Rn* в операнд с плавающей точкой из регистра *Rm* без изменения значений его порядка и мантиссы. Результат помещается в регистр Rs. Суффикс d обозначает выполнение операции сразу для двух пар операндов с формированием двух результатов.

NAN на входе приводит к результату со всеми битовыми разрядами, равными 1. Денормализованные данные в *Rm* приводят к нулевому результату со знаком, скопированным из *Rn.*

Флаги состояния

AZ Установлен, если результат равен ±нулю или денормализован;
AUS Не сбрасывается;
AN Установлен, если результат отрицателен;
AV Сброшен;
AVS Не сбрасывается;

AC Сброшен;

AI Установлен, если какой-либо входной операнд есть NAN;

AIS Установлен, если какой-либо входной операнд есть NAN; иначе не сбрасывается.

Опции

(NF) не обновлять флаги (состояния);

(mX) Поменять местами регистры в паре Rmd;

(nX) Поменять местами регистры в паре Rnd.

10.2.20.12 Масштабирование

Синтаксис

{X|Y|XY}SFRsd = SCALB FRmd BY Rn {({mX} {NF}) }; (Вариант 2)

{X|Y|XY}F*Rsd* = SCALB *FRmH* BY *Rn,* SCALB *FRmL* BY *Rn* {(NF)}; (Вариант 1)

Действие

Эта команда масштабирует порядок операнда в формате с плавающей точкой в Rm путём его суммирования с целым числом в дополнительном представлении в формате с фиксированной точкой в регистре Rn. Масштабированный результат в формате с плавающей точкой помещается в регистр Rs. Если сумма порядка Rm и Rn равна нулю, результатом операции будет ноль. Для обоих операндов регистровой пары Rmd используется одно и тоже значение Rn. Суффикс d обозначает выполнение операции сразу для двух пар операндов с формированием двух результатов.

В случае переполнения результатом будет ±бесконечность. Денормализованный результат сбрасывается к ±нулю. Денормализованные входные данные сбрасываются к ±нулю. NAN на входе приводит к результату со всеми битовыми разрядами, равными 1.

Флаги состояния

AZ Установлен, если округлённый результат денормализован или равен ±нулю;

AUS Установлен, если округлённый результат денормализован; иначе не сбрасывается;

AN Установлен, если результат отрицателен;

AV Установлен, если при округлении произошло переполнение;

AVS Установлен, если при округлении произошло переполнение; иначе не сбрасывается;

AC Сброшен;

AI Установлен, если входной операнд есть NAN;

AIS Установлен, если входной операнд есть NAN; иначе не сбрасывается.

Опции

(NF) Не обновлять флаги (состояния);

(mX) Поменять местами регистры в паре Rmd.

10.2.20.13 Пересылка

Синтаксис

{X|Y|XY}SFRsd = PASS Rmd {({mX}}; (Вариант 2)

{X|Y|XY}F*Rsd* = PASS *RmH,* PASS *RmL* ; (Вариант 1)

Действие

Эта команда передаёт через ALU операнды в формате с плавающей точкой из регистра *Rmd* в соответствующие поля операнда с плавающей точкой в регистре Rsd. Суффикс d обозначает выполнение операции сразу для двух операндов с формированием двух результатов.

Денормализованные входные данные сбрасываются к ±нулю. NAN на входе приводит к результату со всеми битовыми разрядами, равными 1.

Флаги состояния

AZ Установлен, если результат равен ±нулю;
AUS Не сбрасывается;
AN Установлен, если результат отрицателен;
AV Сброшен;
AVS Не сбрасывается; сохраняет значение от предыдущей операции;
AC Сброшен;
AI Установлен, если входной операнд есть NAN;
AIS Установлен, если входной операнд есть NAN; иначе не сбрасывается.

Опции

(mX) Поменять местами регистры в паре Rmd.

10.2.20.14 Обратное приближение

Синтаксис

{X|Y|XY}SFRsd = RECIPS Rmd {({mX}{NF}) }; (Вариант 2)

{X|Y|XY}F*Rsd* = RECIPS *RmH,* RECIPS *RmL* {(NF)}; (Вариант 1)

Действие

Эта команда вычисляет с 8-битной точностью приближение к 1 / *Rm,* числу, обратному к значению в регистре *Rm.* Суффикс d обозначает выполнение операции сразу для двух операндов с формированием двух результатов.

Мантисса приближения определяется по таблице с использованием семи MSB (исключая скрытый бит) мантиссы операнда в *Rm* в качестве индекса. Несмещённый порядок приближения вычисляется как дополнительный код значения несмещённого порядка операнда в *Rm* минус один. Если буквой e обозначить несмещённый порядок *Rm,* то несмещённый порядок в *FRs* равен -e-1. Входное значение ±ноль приводит к результату ±бесконечность и установке флага переполнения. Если несмещённый порядок *Rm* больше +125(для одинарной точности), операция возвращает ±ноль. NAN на входе приводит к результату со всеми битовыми разрядами, равными 1.

Флаги состояния

AZ Установлен, если результат равен ±нулю;
AN Установлен, если результат отрицателен;
AV Установлен, если входной операнд равен ±нулю;
AVS Установлен, если входной операнд равен ±нулю; иначе не сбрасывается[1];
AC Сброшен;
AI Установлен, если входной операнд NAN;
AIS Установлен, если входной операнд NAN; иначе не сбрасывается[2]

Пример

(NF) Не обновлять флаги (состояния);
(mX) Поменять местами регистры в паре Rmd.

10.2.20.15 Обратное приближение к квадратному корню

Синтаксис

{X|Y|XY}*SFRsd* = RSQRTS *Rmd* {({mX}{NF}) }; (Вариант 2)

{X|Y|XY}F*Rsd* = RSQRTS *RmH,* RSQRTS *RmL* {(NF)}; (Вариант 1)

Действие

Эта команда вычисляет с 8-битной точностью приближение к 1/√Rm, числу, обратному к корню квадратному из значения в регистре *Rm.* Суффикс d обозначает выполнение операции сразу для двух операндов с формированием двух результатов.

Мантисса приближения определяется по таблице с использованием LSB несмещённого порядка операнда в *Rm,* конкатенированного с шестью MSB (исключая скрытый бит) мантиссы операнда в *Rm,* в качестве индекса. Несмещённый порядок приближения вычисляется как дополнительный код значения несмещённого порядка операнда в *Rm*, сдвинутый вправо на один бит, минус один. То есть, если буквой e обозначить несмещённый порядок *Rm,* несмещённый порядок в FRs равен –[e/2] – 1. Знак полученного приближения равен знаку входного операнда. Входное значение ±ноль приводит к результату ±бесконечность и установке флага переполнения. Входное значение +бесконечность приводит к результату +ноль. NAN или отрицательное ненулевое значение (включая -бесконечность) на входе приводят к результату со всеми битовыми разрядами, равными 1.

Флаги состояния

AZ Установлен, если результат равен нулю;

AUS Не сбрасывается;

AN Установлен, если входной операнд равен –нулю;

AV Установлен, если входной операнд равен ±нулю;

AVS Установлен, если входной операнд равен ±нулю; иначе не сбрасывается;

AC Сброшен;

AI Установлен, если входной операнд, есть NAN или отрицательное ненулевое число;

AIS Установлен, если входной операнд, есть NAN или отрицательное ненулевое число; иначе не сбрасывается.

Опции

(NF) Не обновлять флаги (состояния);

(mX) Поменять местами регистры в паре Rmd.

10.2.20.16 Мантисса

Синтаксис

{X|Y|XY}*Rsd* = MANT *SFRmd* {({mX}{NF}) }; (Вариант 2)

{X|Y|XY}*Rsd* = MANT *FRmH,* MANT *FRmL* {(NF)}; (Вариант 1)

Действие

Эта команда извлекает мантиссу (биты дробной части плюс скрытый бит, исключая бит знака) из операнда в формате с плавающей точкой в регистре Rm. Полученная беззнаковая величина сдвигается к левому краю поля фиксированной точки и помещается в регистр *Rs.* Суффикс d обозначает выполнение операции сразу для двух операндов с формированием двух результатов.

Отметим, что флаг AN устанавливается в соответствии со знаком входного операнда.

Опции округления игнорируются, округление как таковое не производится, так как все результаты точны по своей сути. Денормализованные входные данные сбрасываются к нулю. NAN или бесконечность на входе приводят к результату со всеми битовыми разрядами, равными 1.

Флаги состояния

AZ Установлен, если результат равен нулю;

AUS Не сбрасывается;

AN Установлен, если входной операнд (FRm) отрицателен;

AV Сброшен;

AVS Не сбрасывается;

AC Сброшен;
AI Установлен, если входной операнд есть NAN или бесконечность;
AIS Установлен, если входной операнд есть NAN или бесконечность; иначе не сбрасывается.

Опции

(NF) Не обновлять флаги (состояния);

(mX) Поменять местами регистры в паре Rmd.

10.2.20.17 Логарифм

Синтаксис

{X|Y|XY}*Rsd* = LOGB *SFRmd* {({mX} {S}{NF})}; (Вариант 2)

{X|Y|XY}*Rsd* = LOGB *FRmH*, LOGB *FRmL* { ({S} {NF}) } ; (Вариант 1)

Действие

Эта команда преобразует порядок операнда в формате с плавающей точкой в регистре *Rm* в несмещённое значение в формате с фиксированной точкой в дополнительном представлении. Результат помещается в регистр *Rs* в виде целого числа. Суффикс d обозначает выполнение операции сразу для двух операндов с формированием двух результатов.

Устранение смещения порядка производится путём вычитания 127 из значения порядка операнда в формате с плавающей точкой одинарной точности в *Rm.* Без опции насыщения ±бесконечность на входе приводит к результату, равному +бесконечности в формате с плавающей точкой; в противном случае ±бесконечность на входе приводит к максимальному положительному результату (0x7FFFFFFF). Без опции насыщения ±ноль на входе приводит к результату, равному -бесконечности в формате с плавающей точкой; в противном случае ±ноль на входе приводит к минимальному отрицательному результату (0x80000000). Денормализованные входные данные сбрасываются к ±нулю. NAN или бесконечность на входе приводят к результату со всеми битовыми разрядами, равными 1.

Флаги состояния

AZ Установлен, если результат в формате с фиксированной точкой равен нулю;
AUS Не сбрасывается;
AN Установлен, если результат в формате с фиксированной точкой отрицателен;
AV Установлен, если входной операнд ±бесконечность или ±ноль
AVS Установлен, если входной операнд ±бесконечность или ±ноль; иначе не сбрасывается;
AC Сброшен;
AI Установлен, если входной операнд есть NAN;
AIS Установлен, если входной операнд есть NAN; иначе не сбрасывается.

Опции

( ) Насыщение выключено;
(S) Насыщение включено;
(NF) Не обновлять флаги (состояния);
(mX) Поменять местами регистры в паре Rmd.

Примеры

YRS7:6 = LOGB SFR3:2 (S) (NF)

10.2.20.18 Сложение/Вычитание (двойная операция)

Синтаксис

{X|Y|XY}*SFRsd = Rmd + Rnd*, *SFRd = Rmd - Rnd* {({mX}{nX} {NF}) }; (Вариант 2)

{X|Y|XY} FRsd = RmH + RnH, RmL + RnL FRad = RmH + RnH, RmL + RnL {(NF) }; (Вариант 1)

Действие

Эта команда одновременно складывает и вычитает операнды в формате с плавающей точкой в регистрах *Rm* и *Rn.* Результаты помещаются в регистры Rs и Ra.

Суффикс d обозначает выполнение операции сразу для двух пар операндов с формированием четырех результатов.

Флаги состояния

AZ Установлен, если округлённый результат денормализован или равен ±нулю;

AUS Установлен, если округлённый результат денормализован; иначе не сбрасывается;

AN Установлен, если результат отрицателен;

AV Установлен, если при округлении произошло переполнение;

AVS Установлен, если при округлении произошло переполнение; иначе не сбрасывается;

AC Сброшен;

AI Установлен, если какой-либо входной операнд есть NAN или оба входных операнда суть бесконечности любого знака;

AIS Установлен, если какой-либо входной операнд есть NAN или оба входных операнда суть бесконечности любого знака; иначе не сбрасывается.

Опции

(NF) Не обновлять флаги (состояния);

(mX) Поменять местами регистры в паре Rmd;

(nX) Поменять местами регистры в паре Rnd.

10.3 Команды CLU

Коммуникационно-логическое устройство (CLU) выполняет все арифметические операции (сложение/вычитание) и логические операции, специфичные для телекоммуникационных алгоритмов. Описание операций CLU, флагов состояния CLU, CLU-предикатов, а также примеры можно найти в подразделе «Коммуникационно-логическое устройство (CLU)».

Соглашения по обозначениям имён регистров, необязательных элементов или вариантов, используемые в справках на команды, подробно освещены в разделе «Регистры регистровых файлов». Вкратце эти соглашения таковы:

{ } - необязательные элементы указаны в фигурных скобках; фигурные скобки не являются частью синтаксиса команды.

| - вертикальная черта разделяет варианты значений элементов; вертикальные черты не являются частью синтаксиса команды.

Rmd - имена регистров, выделенные курсивом, представляют собой подставляемые шаблоны для имён одиночных (*Rs, Rm* или *Rn),* двойных (*Rsd, Rmd* или *Rnd)* или квадро- *(Rsq, Rmq* или *Rnq*) регистров.

10.3.1 *Решётчатый максимум (CLU)*

Синтаксис

{X|Y|XY}{S} TRsd = TMAX (T*Rmd+ Rmq_h, TRnd + Rmq_l)* { (NF) } ;

{X|Y|XY}{S}T*Rsd* = TMAX*(TRmd - Rmq_h, TRnd - Rmq_l)* {(NF) } ;

{X|Y|XY}{*S*}*Rs* = TMAX(*TRm, TRn)* {(NF)} ;

/* здесь Rmq_h должен быть старшей частью квадрорегистра, а Rmq_l должен быть младшей частью ТОГО ЖЕ квадрорегистра */

Действие

Функция TMAX выполняется над аргументами в TRm и TRn или над результатами их суммы или разности. Более подробно о выполнении функции TMAX, см. подразделы «Функция TMAX» и «Решётчатые функции».

Примечание – В данной команде режим насыщения всегда включён. См. подраздел «Опция Насыщение».

Флаги состояния

TROV Флаг переполнения (TROV), пересчитывается на каждом такте;

TRSOV Устанавливается при наступлении переполнения и сбрасывается только загрузкой регистров X/YSTAT.

Опции

(NF) Не обновлять флаги (состояния).

Примеры

XYTR1:0 = TMAX(TR3:2 + R3:2, TR5:4 + R1:0) ;;
XYTR1:0 = TMAX(TR3:2 - R7:6, TR5:4 - R5:4) ;;
XYR8 = TMAX(TR4, TR0) ;;

10.3.2 *Максимум (CLU)*

Синтаксис

{X|Y|XY}{S}T*Rsd* = MAX(T*Rmd* + *Rmq_h,* T*Rnd* + *Rmq_l)* {(NF)} ;

{X|Y|XY}{S}T*Rsd* = MAX*(TRmd - Rmq_h,* T*Rnd - Rmq_l)* { (NF) } ;

Действие

Функция MAX выполняется над результатами суммы или разности операндов в регистрах T*Rm* и T*Rn.* Более подробно о выполнении функции MAX, см. подраздел «Решётчатые функции».

Примечание – В данной команде поддерживается режим насыщения. Более подробно, см. подраздел «Опция Насыщение».

Флаги состояния

TROV Флаг переполнения (TROV), пересчитывается на каждом такте;

TRSOV Устанавливается при наступлении переполнения и сбрасывается только загрузкой регистров X/YSTAT.

Опции

(NF) Не обновлять флаги (состояния)

Примеры

XYTR1:0 = MAX(TR3:2 + R3:2, TR5:4 + R1:0) ;;
XYTR1:0 = MAX(TR3:2 - R7:6, TR5:4 - R5:4) ;;

10.3.3 *Пересылка регистров результата (TR) и истории (TRH) решётчатых операций, а также коммуникационного управляющего регистра (CMCTL)*

Синтаксис

{X|Y|XY*}Rs* = T*Rm* ; /* пересылка TRx */

{X|Y|XY}*Rsd* = T*Rmd* ;

{X|Y|XY}*Rsq*= TRmq ;

{X|Y|XY}*Rs* = THДт ; /* пересылка THRx */

{X|Y|XY}*Rsd* = TH*Rmd* ;

{X|Y*XY}Rsq* = *THRmq* ;

{X|Y|XY}*Rs* = CMCTL ; /* пересылка CMCTL */

Действие

Данные из регистра источника (справа от знака =) пересылаются в регистр приёмника (слева от знака =). Данные могут быть простыми (32-битными), двойными (64-битными) или квадрословами (128-битными). Номера регистров должны быть согласованы с размером данных. Данные пересылаются на стадии конвейера EX2.

Флаги состояния

Команда не влияет на флаги состояния.

Опции

Отсутствуют.

Примеры

XYR8 = TR4 ;;
XYR1:0 = TR3:2 ;;
XYR11:8 = TR7*:4* ;;
XYR8 = THR3 ;;
XYR1:0 = THR3:2 ;;
XYR11:8 = THR3:0 ;;
XYR8 = CMCTL ;;

См. также пункт «Загрузка регистров CLU результата (TR) и истории (TRH) решётчатых операций, а также коммуникационного управляющего регистра (CMCTL) в подразделе «Команды сдвигового устройства»

10.3.4 *Сжатие частотной полосы сигнала с пересылкой регистра решётчатых операций (двойная операция, CLU)*

Синтаксис

{X|Y|XY}T*Rs* += DESPREAD *[Rmq,* THRd) ;

{X|Y|XY}*Rs* = *TRs, TRs* = DESPREAD *(Rmq, THRd)* {(NF)} ; *(двойная операция)*

{X|Y|XY}*Rsd* = T*Rsd,* T*Rsd* = DESPREAD *(Rmq, THRd)* {(NF)}; *(двойная операция)*

Действие

Команда DESPREAD реализует высокопараллельную операцию комплексного умножения с накоплением, оптимизированную для использования в системах CDMA. Процедура сжатия частотной полосы сигнала включает в себя вычисление нескольких наборов корреляций между комплексными входными данными и предвычисленной комплексной рассеивающей/скремблирующей кодовой последовательностью. Входные данные состоят из отсчётов с 8-битными действительной и мнимой частями. С другой стороны, отсчёты из кодовой последовательности всегда являются элементами множества {1+j,-1+j,-1-j, 1-j} и, следовательно, могут быть заданы 1-битными действительной и мнимой частями. Команда DESPREAD, пользуясь этим свойством, позволяет параллельно вычислять восемь результатов операций умножения с накоплением в каждом вычислительном блоке за один такт. Более подробно о выполнении функции DESPREAD, см. «Функции сжатия частотной полосы сигнала».

Примечание – В данной команде режим насыщения всегда включён. Более подробно, см. подраздел «Опция Насыщение».

При выполнении этой команды в параллель с командой загрузки регистров THR последняя имеет приоритет над сдвигом регистра THR в данной команде.

Флаги состояния

TROV Флаг переполнения (TROV), пересчитывается на каждом такте;

TRSOV Устанавливается при наступлении переполнения и сбрасывается только загрузкой регистров X/YSTAT.

Опции

(NF) Не обновлять флаги (состояния)

Примеры

```
XYTR8 += DESPREAD (R7:4, THR1:0) ;;
XYR8 = TR8, TR8 = DESPREAD (R7:4, THR1:0) ;;
XYR1:0 = TR1:0, TR1:0 = DESPREAD (R7:4, THR1:0) ;;
```

10.3.5 *Вычисление кросс-корреляций с пересылкой регистра решётчатых операций (двойная операция, CLU)*

Синтаксис

{X|Y|XY}TRsa = XCORRS *(Rmq,* TH*Rnq)*

{(CUT <*Imm*>|R)}{(CLR)}{(EXT)}{(NF)};

{X|Y|XY}*Rsq = TRbq, TRsa* = XCORRS(*Rmq, THRnq)*

{(CUT *Imm*>|*R)*}{(CLR)}{(EXT)}{(NF)}; *(двойная операция)*

/* здесь T*Rsa* = TR15:0 или TR31:16 */

Действие

Входной регистр Rmq для команды состоит из 8 комплексных коротких слов, с D7 по DO, в которых каждое комплексное число состоит из 2 байт. Старший байт представляет мнимую часть, а младший байт - действительную часть.

Регистр истории решётчатых операций TH*Rnq* содержит 63 комплексных элемента вектора “C” (с C[62] по C [0]) и два неиспользованных бита. Каждый элемент (C[k]) составлен из двух бит: бит мнимой части занимает бит 2k+3 регистра THRnq, а бит действительной части – бит 2k+2 регистра TH*Rnq*. Два младших бита TH*Rnq* не используются. На каждом проходе вычислений используются только элементы с C[22] по C [0]. Элементы с C[62] по C[23] используются в следующих проходах вычисления после соответствующего сдвига.

Более подробно см. подраздел «Функция вычисления кросс-корреляций».

Флаги состояния

TROV/TRSOV Если переполнение произошло хотя бы в одном из регистров TR.

Опции

(CUT Imm|R) Флаги TROV и TRSOV устанавливаются; в противном случае TROV сбрасывается, а TRSOV не изменяется;

(CLR) Определяет данные, отбрасываемые от множества; если указано R вместо непосредственного значения, то значения CUT берётся из регистра CMCTL; если опущена, значение CUT по умолчанию равно 0. Диапазон допустимых значений для CUT от -22 до 22. Биты 5-0 регистра CMCTL могут содержать значение CUT в формате знакового целого числа; биты 31-6 зарезервированы;

(EXT) Сбрасывает указанные регистры TRsa до следующего суммирования; если опущена, регистры TRsa не очищаются

указывает на входные данные 16-битной точности (32-битное накопление); если опущена - входные данные 8-битной точности (16-битное накопление);

(NF) Не обновлять флаги (состояния).

Примечание – В данной команде режим насыщения всегда включён. Более подробно, см. подраздел «Опция Насыщение».

Примеры

XYTR15:0 = XCORRS (R7:4, THR3:0) (CLR) ;;

XYR11:8 = TR11:8, TR31:16 = XCORRS (R7:4,THR3:0) (CLR)

10.3.6 *Сложить/сравнить/выбрать (CLU)*

Синтаксис

{X|Y|XY}*{S}TRsq* = ACS (T*Rmd,* T*Rnd, Rm)* {(TMAX)} {(NF)} ;

{X|Y|XY}*Rsq* = T*Raq,* {S}T*Rsq* = ACS (T*Rmd,* T*Rnd, Rm)* {(TMAX)}{(NF)} ;
(двойная операция)

Действие

При выполнении команды TRsq = ACS (TRmd, TRnd, Rm) ; каждое короткое слово в Rm прибавляется и вычитается из соответствующих коротких слов в TRmd и TRnd. Затем, как показано на рисунке 70, четыре результата сложения и вычитания сравниваются по решётчатому принципу.

В регистр истории решётчатых операций THR [1:0] заносится значение, определяющее выбор функций MAX/TMAX. После каждой операции ACS содержимое THR1:0 сдвигается вправо на 4 бита, и четыре бита выбранных значений загружаются в биты 31:28 регистра THR1. Бит выбранного значения указывает, какой из входов был выбран функцией MAX или TMAX. При выборе TRm+/-Rm бит выбранного значения равен 1, а при выборе TRn-/+Rm бит выбранного значения равен 0.

При выполнении команды STRsq = ACS (TRmd, TRnd, Rm) ; каждый байт в Rm прибавляется и вычитается из соответствующих байтов в TRmd и TRnd. Затем, как показано на рисунке 71, восемь результатов сложения и вычитания сравниваются по решётчатому принципу.

В регистр истории решётчатых операций THR [1:0] заносится значение, определяющее выбор функций MAX/TMAX. После каждой операции ACS содержимое THR1: 0 сдвигается вправо на 8 бит, и восемь битов выбранных значений загружаются в биты 31:24 регистра THR1. Бит выбранного значения указывает, какой из входов был выбран функцией MAX или TMAX. При выборе TRm+/-Rm бит выбранного значения равен 1, а при выборе TRn-/+Rm бит выбранного значения равен 0.

Эта команда допускает добавление пересылки регистров решётчатых операций (команда с двойной операцией), как например в:

Rsq = T Raq, {S}T Rsq = ACS (T Rmd, T Rnd, Rm) ;

Эта команда может выполняться в параллель с командами сдвигового устройства, ALU, умножителя и командами загрузки регистров CLU. Она не может быть выполнена в параллель с командами CLU того же вычислительного блока.

Более подробно, см. подраздел «Функция сложить/сравнить/выбрать».

Примечание – В данной команде режим насыщения всегда включён. Более подробно, см. подраздел «Опция Насыщение».

Флаги состояния

TROV Флаг переполнения (TROV), пересчитывается на каждом такте;

TRSOV Устанавливается при наступлении переполнения и сбрасывается только загрузкой регистров X/YSTAT.

Опции

(TMAX) Результат сравнения основан на следующем выражении:
Ln(1+e - | A - B |), где A и B - сравниваемые значения;

(NF) Не обновлять флаги (состояния).

Примеры

Эта функция может быть использована в решётчатых операциях следующим образом:

```
Loop:  TR11:8 = ACS (TR1:0, TR5:4, R0);;
       TR15:12 = ACS (TR3:2, TR7:6, R0);;
       TR3:0 = ACS (TR9:8, TR13:12, R1);;
       TR15:4 = ACS (TR11:10, TR15:14, R1); if nLC0E, jump Loop;;
```

10.4 Команды умножителя

Умножитель выполняет в процессоре *операции умножения* над данными в форматах с фиксированной и плавающей точкой и *операции умножения с накоплением* над данными в формате с фиксированной точкой. Это устройство выполняет также *операции комплексного умножения* над данными в формате с фиксированной точкой. И наконец, умножитель выполняет *операции сжатия данных* над накопленными результатами при пересылке данных в регистровый файл в форматах с фиксированной точкой. Описание операций умножителя, флагов состояния умножителя, предикатов умножителя, а также примеры можно найти в разделе «Умножитель».

Соглашения по обозначениям имён регистров, необязательных элементов или вариантов, используемые в справках на команды, подробно освещены в подразделе «Регистры регистровых файлов». Вкратце эти соглашения таковы:

- { } - необязательные элементы указаны в фигурных скобках; фигурные скобки не являются частью синтаксиса команды;
- | - вертикальная черта разделяет варианты значений элементов; вертикальные черты не являются частью синтаксиса команды;
- Rmd - имена регистров, выделенные курсивом, представляют собой подставляемые шаблоны для имён одиночных (Rs, Rm или Rn), двойных (Rsd, Rmd или Rnd) или квадро- (Rsq, Rmq или Rnq) регистров.

Каждая команда, по которой дана справка, занимает одно поле команды в командной строке. Более подробную информацию о командных строках и ограничениях по комбинированию команд можно найти в подразделах «Синтаксис и структура командной строки» и «Правила запараллеливания команд».

10.4.1 *Умножение (обычных слов)*

Синтаксис

{X|Y|XY*}Rs = Rm * Rn* {({U|nU}{I|T}{S}{NF})} ;

{X|Y|XY*}Rsd = Rm * Rn* {({U|nU}{I}{NF})} ;

Действие

Это операция 32-битного умножения обычного слова в регистре *Rm* на обычное слово в регистре *Rn.* В случае дробных операндов, если режим округления включён (не указана опция T), производится округление результата. (Отметим, что опция T неприменима в случае целых данных.) Результат операции помещается в регистр *Rs* (рисунок 161).

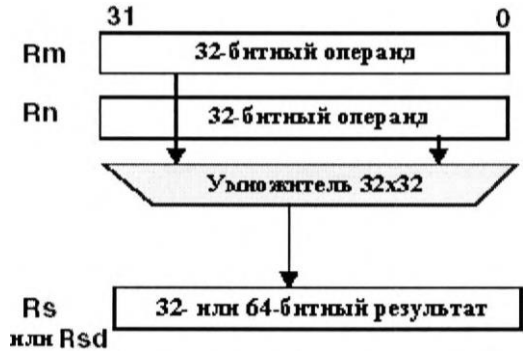


Рисунок 161 – Выполнение команды умножения (обычных слов)

Флаги состояния

MZ Установлен, если все биты результата равны нулю Установлен, если результат отрицателен;

M Установлен или сброшен в соответствии с форматом данных в следующих случаях (MOS не изменяется, если MV сброшен):

- MV Установлен, если старшие 33 бита результата умножения не все 0 и не все 1(MOS) установлен, если старшие 32 бита результата умножения не все 0;
- MU Сброшен (не изменяется).

Опции

( ) Rm, Rn – знаковые дробные, результат округляется (если однословный);

(U) Rm, Rn – беззнаковые;

(nU) Rm – знаковый, Rn – беззнаковый;

(I) Операнды целые;

(T) Усечение результата (только для дробных операндов при однословном результате);

(S) Насыщение (только для целых операндов);

(NF) Не обновлять флаги (состояния).

Надо отметить, что не все допустимые комбинации опций имеют смысл. Например, опции округления и усечения применимы только к дробным операндам и неприменимы к целым. См. подраздел «Опции команд умножителя».

Примеры

Листинг 41 – Умножение двух дробных операндов

```
XR1 = 0xFF46AC0C ;; /* xR1 = -0xB953F4 */
XR2 = 0xAF305216 ;;
XR0 = R1 * R2 (nUT) ;; /* xR0 = 0xFF812CA1 */
```

Это команда 32-битного перемножения двух дробных операндов. В R1 операнд знаковый, а в R2 – беззнаковый. В результате умножения получается 32-битное беззнаковое усечённое значение, и установлен флаг MN, указывающий, что результат отрицательный.

Листинг 42 – Умножение двух целых беззнаковых операндов

```
XR0 = 0x0046AC0C ;;
 XR1 = 0x00005216 ;;
XR3:2 = R0 * R1(UI) ;; /* xR3 = 0x00000016, xR2 = 0xA92EA108 */
```

Это команда 32-битного перемножения двух целых беззнаковых операндов в регистрах R0 и R1. В результате получается 64-битное целое беззнаковое значение. Старшие 32 бита результата размещаются в регистре R3, младшие 32 бита – в регистре R2. В этом примере флаги не устанавливаются.

10.4.2 *Умножение с накоплением (обычных слов)*

Синтаксис

{X|Y|XY}*MRa* += *Rm* * *Rn* {({U}{I}{C|CR}{NF})} ;

{X|Y|XY}*MRa* -= *Rm* * *Rn* {({I}{C|CR}{NF})} ;

/* здесь *MRa* либо MR1:0, либо MR3:2 */

Действие

Эта команда выполняет операцию 32-битного умножения с накоплением. При этом операнды в регистрах *Rm* и *Rn* перемножаются, а затем полученный результат прибавляется к значению (или вычитается из значения) в указанном регистре MR. 80-битный результат операции помещается в тот же регистр накопления MR. Так как регистры MR1:0 и MR3:2 лишь 64-битные, то дополнительные 16 бит (для 80-битного значения накопления) хранятся в регистре MR4. Биты MR4[15:0] хранят дополнительные биты, если в качестве регистра-приёмника умножения с накоплением указан MR1:0, биты MR4[31:16] хранят дополнительные биты, если в качестве регистра-приёмника умножения с накоплением указан MR3:2. Режим насыщения всегда включён.

Флаги состояния

MZ Не изменяется;
MN Не изменяется;
MV Не изменяется
MU (MUS) Не изменяется;
MOS Устанавливается в соответствии с конечным результатом сложения и типом данных.

Флаг MOS устанавливается по следующим правилам:

- Для знаковых дробных операндов: если конечный результат больше или равен 2^{15} или меньше -2^{15};
- Для знаковых целых операндов: если конечный результат больше или равен 2^{79} или меньше -2^{79};
- Для беззнаковых дробных операндов (только в случае сложения): если конечный результат больше или равен 2^{16};
- Для беззнаковых целых операндов (только в случае сложения): если конечный результат больше или равен 2^{80}.

Опции

(U) Rm, Rn – беззнаковые (только в случае сложения);
(I) Операнды целые;
(C) Обнулить MR до накопления;
(CR) Обнулить и округлить;
(NF) Не обновлять флаги (состояния).

Более подробно о возможных опциях см. подраздел «Опции команд умножителя».

Примеры

Листинг 43 – Пример 1 умножения с накоплением

```
/*{X|Y|XY}MRa += Rm * Rn {({U}{I}{C|CR})} ; */
```

```
R2 = -2 ;;
R1 = 5 ;;
MR3:2 += R1 * R2 (I) ;;
```

В примере в листинге 43, если предыдущее содержимое регистра MR3:2 было равно 7, то новое значение будет -3. Флаги устанавливаться не будут.

Листинг 44 – Пример 2 умножения с накоплением

```
R0 = 0xF0060054 ;;
R2 = 0xF036AC42 ;;
MR3:2 += R0 * R2 (UI) ;;
MR3:2 += R0 * R2 (UI) ;;
```

Пример в листинге 44 демонстрирует, как регистр MR4 используется для хранения дополнительных битов 80-битного значения накопления. Содержимое регистров MR4 :0 после второй команды умножения с накоплением таково:

```
MR0 = 0x00000000
MR1 = 0x00000000
MR2 = 0x76F90B50
MR3 = 0xC271C629
MR4 = 0x00010000
```

Листинг 45 – Пример 3 умножения с накоплением

```
R0 = 0xF0060054 ;;
R2 = 0xF036AC42 ;;
MR1:0 += R0 * R2 (UI) ;;
MR1:0 += R0 * R2 (UI) ;;
```

Пример в листинге 45 аналогичен предыдущему с единственным отличием, что используется регистр MR1:0 вместо MR3:2. Содержимое регистров MR4:0 после второй команды умножения с накоплением таково:

```
MR0 = 0x76F90B50
MR1 = 0xC271C629
MR2 = 0x00000000
MR3 = 0x00000000
MR4 = 0x00000001
```

Обратите внимание на разницу в расположении дополнительных 16 бит в регистре MR4 в предыдущих примерах.

Листинг 46 – Пример 4 умножения с накоплением

```
/* {X|Y|XY}MRa -= Rm * Rn {({I}{C|CR})} ; */
R0 = 2;;
R1 = 5;;
R2 = -10;;
R3 = 6;;
MR3:2 -= R0 * R1 (I);;
MR3:2 -= R2 * R3 (I);;
```

В примере в листинге 46 содержимое регистров MR4:0 после первой команды умножения с накоплением таково:

```
MR0 = 0x00000000
MR1 = 0x00000000
MR2 = 0xFFFFFFF6
MR3 = 0xFFFFFFFF
```

MR4 = 0xFFFF0000

После второй команды умножения с накоплением результаты таковы:

MR0 = 0x00000000
MR1 = 0x00000000
MR2 = 0x00000032
MR3 = 0x00000000
MR4 = 0x00000000

Листинг 47 – Пример 5 умножения с накоплением

```
R0 = 2;;
R1 = 5;;
R2 = -10;;
R3 = 6;;
MR1:0 -= R0 * R1 (I);;
MR1:0 -= R2 * R3 (I);;
```

В примере в листинге 47 содержимое регистров MR4:0 после первой команды умножения с накоплением таково:

MR0 = 0xFFFFFFF6
MR1 = 0xFFFFFFFF
MR2 = 0x00000000
MR3 = 0x00000000
MR4 = 0x0000FFFF

После второй команды умножения с накоплением результаты таковы:

MR0 = 0x00000032
MR1 = 0x00000000
MR2 =0x00000000
MR3 = 0x00000000
MR4 = 0x00000000

10.4.3 *Умножение с накоплением и пересылкой регистра MR (двойная операция, обычные слова)*

Синтаксис

{X|Y|XY}Rs = MRa, MRa += Rm * Rn {({U}{I}{C|CR}{NF})}; (двойная операция)

{X|Y|XY}Rsd = MRa, MRa += Rm * Rn {({U}{I}{C}{NF})} ; (двойная операция)

/* здесь MRa либо MR1:0, либо MR3:2 */

Действие

Это 32-битное умножение с накоплением, при котором 32-битные значения в регистрах *Rm* и *Rn* перемножаются, результат прибавляется к регистру *MRa,* и предыдущее значение *MRa* пересылается в *Rs.* Опция Сброса (C) обнуляет регистр *MRa* после записи его значения в *Rs* или *Rsd* и до сложения с новым результатом умножения. При пересылке регистра MR в регистровый файл всегда происходит усечение (дополнительные биты переполнения из регистра MR4 не пересылается).

Когда регистр приёмника в операции пересылки является регистровой парой *Rsd,* 64-битное значение накопления пересылается в *Rsd* (см. подраздел «Операции умножения»).

Когда регистр приёмника в операции пересылки является одиночным *Rs,* часть 64-битного значения накопления, пересылаемая в *Rs,* зависит от типа данных:

- целые данные – пересылается младшая часть;
- дробные данные – пересылается старшая часть.

Что касается операции умножения с накоплением, то дополнительные 16 бит (80-битного значения накопления) хранятся в регистре MR4. Биты MR4[15:0] хранят дополнительные биты, если в качестве регистра-приёмника умножения с накоплением указан MR1:0, биты MR4[31:16] хранят дополнительные биты, если в качестве регистра-приёмника умножения с накоплением указан MR3:2.

Флаги состояния

MZ Не изменяется;
MN Не изменяется;
MV Не изменяется;
MU (MUS) Не изменяется;
MOS Устанавливается в соответствии с конечным результатом сложения и типом данных, (см. ниже).

Флаг MOS устанавливается по следующим правилам:

- Для знаковых дробных операндов - если конечный результат больше либо равен 2^{15} или меньше -2^{15};
- Для знаковых целых операндов - если конечный результат больше либо равен 2^{79} или меньше -2^{79}. Также в случае команды Rs = MRa, MRa += Rm * Rn (Rs одиночный, MRa двойной), если старое значение MRa больше либо равно 2^{31} или меньше -2^{31};
- Для беззнаковых дробных операндов (только в случае сложения), если конечный результат больше либо равен 2^{16}
- Для беззнаковых целых операндов (только в случае сложения), если конечный результат больше либо равен 2^{80}.

Остальные флаги не изменяются.

Опции

(U) Rm, Rn – беззнаковые;
(I) Операнды целые;
(C) Обнулить MR после пересылки в регистровый файл, но до накопления;
(CR) Обнулить и округлить;
(NF) Не обновлять флаги (состояния).

Более подробно о возможных опциях см. подраздел «Опции команд умножителя».

Примеры

Листинг 48 – Пример 1 умножения с накоплением и пересылкой MR

```
R0 = 2 ;;
R1  = 5 ;;
R2 = 10 ;;
R3 = 6 ;;
R4 = MR1:0, MR1:0 += R0 * R1(UIC) ;;
R4 =MR1:0, MR1:0 += R2 * R3 (UIC) ;;
```

В примере в листинге 48 после выполнения первой команды с умножением и пересылкой, содержимое регистров MR4:0 следующее (в предположении, что предыдущее содержимое MR4:0 было нулевым):

MR0 = 0x0000000A
MR1 = 0x00000000
MR4 = 0x00000000

В регистр R4 загружено значение 0x00000000.

После выполнения второй команды с умножением и пересылкой, регистры MR4:0 содержат:
MR0 = 0x0000003C
MR1 = 0x00000000
MR4 = 0x00000000
В регистр R4 загружено значение 0x0000000A

Листинг 49 – Пример 2 умножения с накоплением и пересылкой MR

```
R0 = 0x12345678 ;;
R1 = 0x87 654 321 ;;
R2 = 0xA74EF254 ;;
R3 = 0xB4ED9032 ;;
R4 = MR3:2, MR3:2 += R0 * R1 ;;
R4 = MR3:2, MR3:2 += R2 * R3 ;;
```

В примере в листинге 49 после выполнения первой команды с умножением и пересылкой, содержимое регистров MR4 :0 следующее (в предположении, что предыдущее содержимое MR4:0 было нулевым):
MR2 = 0xE1711AF0
MR3 = 0xEED8ED1A
MR4 = 0xFFFF0000
В регистр R4 загружено значение 0x00000000.
После выполнения второй команды [с умножением и пересылкой], регистры MR4:0 содержат:
MR2 = 0xDC6E43C0
MR3 = 0x22DD717B
MR4 = 0x00000000
В регистр R4 загружено значение 0xEED8ED1A.

Листинг 50 – Пример 3 умножения с накоплением и пересылкой MR

```
R0 = 2000;;
R1 = 4000;;
R2 = 3000;;
R3 = -5000;;
R5:4 = MR1:0, MR1:0 += R0 * R1 (I);;
R5:4 = MR1:0, MR1:0 += R2 * R3 (I);;
R5:4 = MR1:0, MR1:0 += R0 * R1 (I);;
```

В примере в листинге 50 после выполнения первой команды с умножением и пересылкой, содержимое регистров MR 4:0 следующее (в предположении, что предыдущее содержимое MR 4:0 было нулевым):
MR0 = 0x007A1200
MR1 = 0x00000000
MR4 = 0x00000000
В регистры R5 и R4 загружено значение 0x00000000.
После выполнения второй команды с умножением и пересылкой, регистры MR4:0 содержат:
MR0 = 0xFF953040
MR1 = 0xFFFFFFFF
MR4 = 0x0000FFFF
В регистр R5 загружено значение 0x00000000, в регистр R4 загружено значение 0x007A1200.

После выполнения третьей команды с умножением и пересылкой, регистры MR4:0 содержат:
MR0 = 0x000F4240
MR1 = 0x00000000
MR4 = 0x00000000

В регистр R5 загружено значение 0xFFFFFFFF, в регистр R4 загружено значение 0xFF953040.

Листинг 51 – Пример 4 умножения с накоплением и пересылкой MR

```
R0 = 100 ;;
R1 = 50 ;;
R2 = 300 ;;
R3 = 70 ;;
R5:4 = MR1:0, MR1:0 += R0 * R1 (UI) ;;
R5:4 = MR1:0, MR1:0 += R2 * R3 (UI) ;;
```

Предположим, что предыдущие значения в регистрах MR4:0 таковы:
MR0 = 0xFFFFF500
MR1 = 0xFFFFFFFF
MR4 = 0x00000000

В примере в листинге 51 после выполнения первой команды с умножением и пересылкой, содержимое регистров MR 4:0 следующее:
MR0 = 0x00000888
MR1 = 0x00000000
MR4 = 0x00000001

В регистр R5 загружено значение 0xFFFFFFFF, в регистр R4 загружено значение 0xFFFFF500.

После выполнения второй команды [с умножением и пересылкой], регистры MR4:0 содержат:
MR0 = 0x00005A90
MR1 = 0x00000000
MR4 = 0x00000001

В регистр R5 загружено значение 0xFFFFFFFF, в регистр R4 загружено значение 0xFFFFFFFF. Как видно по содержимому регистра MR4, результат в регистре MR1:0 больше, чем [любое] 64-битное беззнаковое целое число; наибольшее беззнаковое целое число, которое может быть представлено 64 битами, записано в двойной регистр R5:4. Флаг MOS не установлен.

Листинг 52 – Пример 5 умножения с накоплением и пересылкой MR

```
R0 = 0x4236745D ;;
R1 = 0x53ACBE34 ;;
R2 = 0xF38D153C ;;
R3 = 0x4129EDA1;;
R5:4 = MR3:2, MR3:2 += R0 * R1;;
R5:4 = MR3:2, MR3:2 += R2 * R3 ;;
```

Предположим, что предыдущие значения в регистрах MR 4:0 таковы:
MR2 = 0x12345678
MR3 = 0xFFFFFFFF
MR4 = 0x7FFF0000

В примере в листинге 52 после выполнения первой команды с умножением и пересылкой, содержимое регистров MR 4:0 следующее:

MR2 = 0xFFFFFFFF
MR3= 0xFFFFFFFF
MR4 =0x7FFF0000

В регистр R5 загружено значение 0x7FFFFFFF, в регистр R4 загружено значение 0xFFFFFFFF. Это произошло из-за того, что дополнительные 16 бит переполнения были установлены в 1, таким образом, произошло насыщение 64-битного результата, записанного затем в регистре R5:4. Флаг MOS также установлен.

После выполнения второй команды [с умножением и пересылкой], регистры MR4:0 содержат:
MR2 = 0xD5FDCD77
MR3 = 0xF9A990DC
MR4 = 0x7FFF0000

В регистр R5 загружено значение 0x7FFFFFFF, в регистр R4 загружено значение 0xFFFFFFFF. Флаг MOS остаётся установленным. Несмотря на то, то данная команда не устанавливает флаг MOS, этот флаг залипает и должен сбрасываться явно.

10.4.4 *Умножение (квадро, коротких слов)*

Синтаксис

{X|Y|XY}*Rsd = Rmd * Rnd* {({U} {I| T} {S } {NF})} ;

{X|Y|XY}*Rsq = Rmd * Rnd* {({U} {I} {NF})} ;

Действие

Это 16-битное квадро-умножение операндов в формате с фиксированной точкой: четырёх коротких слов в регистре *Rm* на четыре коротких слова в *Rn.* В случае дробных операндов, если режим округления включён (не указана опция T), производится округление результата. Отметим, что опция T неприменима в случае целых данных. Результат операции помещается в регистр *Rs* (рисунок 162).

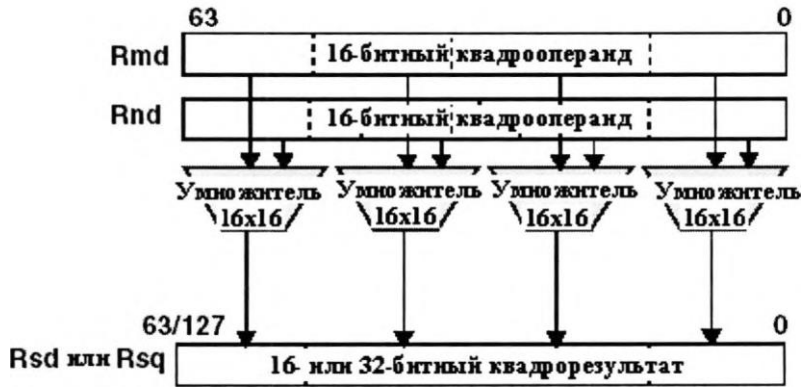


Рисунок 162 – Выполнение команды квадроумножения (коротких слов)

Флаги состояния

MZ Установлен, если все биты хотя бы одного из результатов равны нулю;

MN Установлен, если хотя бы один из результатов отрицателен;

MV (MOS) Установлен или сброшен в соответствии с форматом данных в следующих случаях (MOS не изменяется, если MV сброшен):

*Rsd = Rm * Rn;* =>нет переполнения (сброшен);

Дробные данные => нет переполнения (сброшен)[1];

Знаковые целые данные => установлен, если старшие 17 бит промежуточного 32-битного результата умножения не все 0 и не все 1;

Беззнаковые целые данные => установлен, если старшие 16 бит промежуточного 32-битного результата умножения не все 0;

MU (MUS) Сброшен (не изменяется);

MI Сброшен.

Опции

( ) Rm, Rn – знаковые дробные, результат округляется (если в виде четырёх коротких слов);

(U) Rm, Rn – беззнаковые;

(I) Операнды целые;

(T) Усечение результата (только для дробных операндов при результате в виде четырёх коротких слов);

(S) Насыщение (только для целых операндов);

(NF) Не обновлять флаги (состояния).

Более подробно о возможных опциях см. подраздел «Опции команд умножителя».

Примеры

Листинг 53 – Пример 1 квадро-умножения (коротких слов)

```
XR0 = 0x42861234 ;;
XR1 = 0x782F4217 ;;
XR2 = 0x8AC90FEA ;;
XR3 = 0x724D76EB ;;
XR5:4 = R1:0 * R3:2 (T) ;; /* xR5 = 6B52 3D66, xR4 = C314 0243 */
```

Программа в примере в листинге 53 использует четыре 16-битных умножения [в одной команде] знаковых дробных данных, в результате которых получаются четыре 16-битных усечённых знаковых дробных значения. В этом примере флаг XMN установлен, так как 16-битный результат в старших 16 битах регистра XR4 отрицателен. При таком подходе к генерации флагов нет никаких признаков, показывающих, какое именно умножение каких операндов вызвало установку флага. Флаг сигнализирует лишь о том, что в результате одного из умножений получилось отрицательное значение.

Листинг 54 – Пример 2 квадро-умножения (коротких слов)

```
XR0 = 0x00060004 ;;
XR1 = 0x00030007 ;;
XR2 = 0x00090000 ;;
XR3 = 0x00050008 ;;
XR7:4 = R1:0 * R3:2  (I) ;;
/* xR7 = 0x0000000F, xR6 = 0x00000038, xR5 = 0x00000036 xR4 = 0x00000000 */
```

Программа в примере в листинге 54 использует четыре 16-битных умножения [в одной команде] знаковых целых 16-битных значений, в результате которых

[1] За исключением случая умножения наименьшего отрицательного дробного числа на себя, когда оба флага MV и MOS устанавливаются.

получаются четыре 32-битных знаковых целых значения. В этом примере флаг XMZ установлен, так как один из результатов равен нулю.

10.4.5 *Умножение с накоплением (квадро, коротких слов)*

Синтаксис

{X|Y|XY}MR3:0 += *Rmd* * *Rnd* {({U}{I}{C|CR}{NF})} ;

{X|Y|XY}*MRb* += *Rmd* * *Rnd* {({U}{I}{C}{NF})} ;

/* здесь *MRb* либо MR1:0, либо MR3:2 */

Действие

Эта команда выполняет операцию 16-битного квадро-умножения с накоплением операндов в формате с фиксированной точкой. При этом четыре коротких слова в регистровой паре *Rmd* умножаются покомпонентно на четыре коротких слова в *Rnd,* а затем полученные четыре результата покомпонентно прибавляются к четырём значениям в указанном регистре MR. Результаты операции помещаются в тот же регистр накопления MR.

При использовании регистра MR3:0 четыре результата умножения накапливаются в четырёх регистрах MR с 40-битной точностью. При использовании регистра MR3:2 или MR1:0 четыре результата умножения накапливаются в двух регистрах MR с 20-битной точностью.

Дополнительные биты, полученные в результате умножений, хранятся в регистре MR4. Режим насыщения всегда включён. См. рисунки 163 и 164.

Описание полей регистра MR4 приведено в пункте «Регистр переполнения результата умножителя (MR4)».

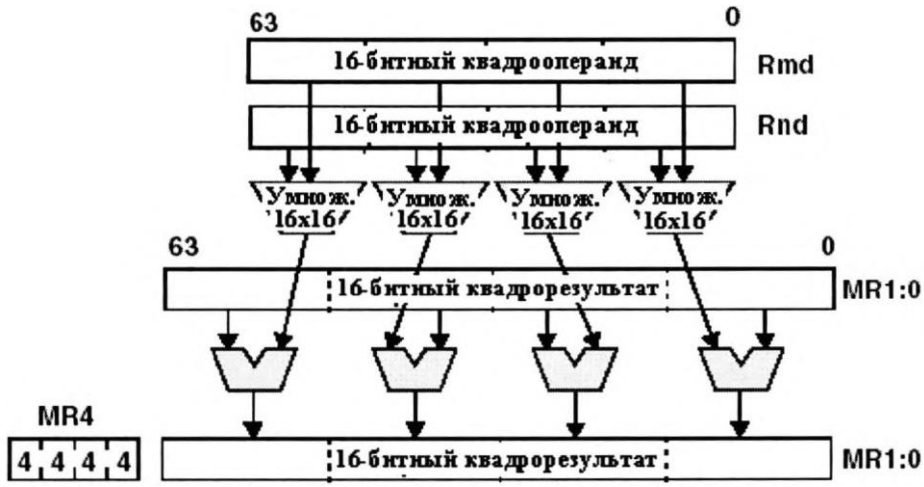


Рисунок 163 – Четыре 16-битных результата

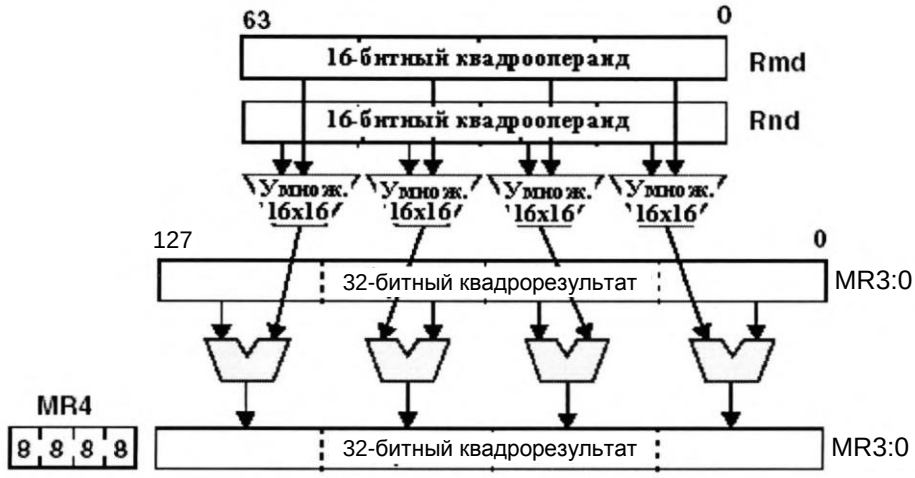


Рисунок 164 – Четыре 32-битных результата

Флаги состояния

MZ Не изменяется;
MN Не изменяется;
MV Не изменяется;
MU (MUS) Не изменяется;
MOS Установлен или сброшен в соответствии с конечным результатом сложения, типом и размером данных (см. ниже).

Для результатов в виде обычных слов (MR3:0 = Rmd * Rnd) флаг MOS устанавливается по следующим правилам:

- для знаковых дробных операндов – установлен, если конечный результат больше либо равен 2^7 или меньше -2^7;
- для знаковых целых операндов – установлен, если конечный результат больше либо равен 2^{39} или меньше -2^{39};
- для беззнаковых дробных операндов – установлен, если конечный результат больше либо равен 2^8;
- для беззнаковых целых операндов – установлен, если конечный результат больше либо равен 2^{40}.

Для результатов в виде коротких слов (MR1:0/3:2= Rmd * Rnd) флаг MOS устанавливается по правилам:

- для знаковых дробных операндов – установлен, если конечный результат больше либо равен 2^3 или меньше -2^3;
- для знаковых целых операндов – установлен, если конечный результат больше либо равен 2^{19} или меньше -2^{19} или если промежуточный результат умножения больше либо равен 2^{15} или меньше -2^{15};
- для беззнаковых дробных операндов – установлен, если конечный результат больше либо равен 2^4;
- для беззнаковых целых операндов – установлен, если:
  - конечный результат больше либо равен 2^{20};
  - промежуточный результат умножения больше либо равен 2^{16}.

Опции

(I) Операнды целые;
(U) Rm, Rn – беззнаковые;
(C) Обнулить MR до накопления;
(CR) Обнулить и округлить;
(NF) Не обновлять флаги (состояния).

Более подробно о возможных опциях см. подраздел «Опции команд умножителя».

Примеры

Листинг 55 – Пример 1 квадро-умножения с накоплением (коротких слов)

```
R0 = 0x70000004 ;;
R1 = 0x06000002 ;;
R2 = 0xF0000005 ;;
R3 = 0xD0000006 ;;
MR3:0 += R1:0 * R3:2 (UIC) ;;
MR3:0 += R1:0 * R3:2 (UI) ;;
```

Пример в листинге 55 демонстрирует 16-битное квадро-умножение с накоплением, результат каждого из которых хранится с полной 32-битной точностью.

Значения в регистрах MR4:0 после второй команды умножения с накоплением таковы:

```
MR0 = 0x00000028
MR1 = 0xD2000000
MR2 = 0x00000018
MR3= 0x09C00000
MR4 =0x00000000
```

В этом примере флаги не устанавливаются.

Листинг 56 – Пример 2 квадро-умножения с накоплением (коротких слов)

```
R0 = 0x70000004 ;;
R1 = 0x06000002 ;;
R2 = 0xF0000005 ;;
R3 = 0xD0000006 ;;
MR3:2 += R1:0 * R3:2 (UIC) ;;
MR3:2 += R1:0 * R3:2 (UI) ;;
```

В примере в листинге 56 значения в регистрах MR4:0 после второй команды умножения с накоплением таковы:

```
MR0 = 0x00000000
MR1 = 0x00000000
MR2 = 0xFFFF0028
MR3= 0xFFFF0018
MR4 =0xF0F00000
```

В этом примере флаг MOS установлен в обеих командах умножения с накоплением, так как имеет место насыщение в двух 16-битных умножениях.

Следующий пример (листинг 57) полностью аналогичен предыдущему за одним исключением: вместо регистра MR3:2 в качестве регистра-приёмника используется MR1:0.

Листинг 57 – Пример 3 квадро-умножения с накоплением (коротких слов)

```
R0 = 0x70000004;;
R1 = 0x06000002;;
R2 = 0xF0000005;;
R3 = 0xD0000006;;
MR1:0 += R1:0 * R3:2 (UIC);;
MR1:0 += R1:0 * R3:2 (UI);;
```

В примере в листинге 57 значения в регистрах MR4:0 после второй команды умножения с накоплением таковы:
MR0 = 0xFFFF0028
MR1 = 0xFFFF0018
MR2 = 0x00000000
MR3 = 0x00000000
MR4 = 0x0000F0F0

В этом примере флаг MOS установлен в обеих командах умножения с накоплением, так как имеет место насыщение в двух 16-битных умножениях.

Обратите внимание на расположение установленных битов регистре MR4. Оно таково, потому что регистр MR4 содержит только по 4 бита переполнения для значений накопления (всего 20 битов).

10.4.6 *Умножение с накоплением и пересылкой регистра MR (двойная операция, квадро, короткие слова)*

Синтаксис

{X|Y|XY} Rsd = MRb, MR3:0 += Rmd * Rnd {({I}{C|CR}{NF})}; (двойная операция)

{X|Y|XY} Rsd = MRb, MRb += Rmd * Rnd {({I}{C} {NF})}; (двойная операция)

/* здесь MRb либо MR1:0, либо MR3:2 */

Действие

Эта команда выполняет операцию 16-битного квадро-умножения с накоплением операндов в формате с фиксированной точкой. При этом четыре коротких слова в регистровой паре *Rmd* умножаются покомпонентно на четыре коротких слова в *Rnd,* полученные четыре результата покомпонентно прибавляются к четырём значениям в указанном регистре *MRb,* и содержимое регистра *MRb* пересылается в *Rs.*

При использовании регистра MR3:0 четыре результата умножения накапливаются в четырёх регистрах MR со всей 40-битной точностью. При использовании регистра MR3:2 или MR1:0 четыре результата умножения накапливаются в двух регистрах MR с 20-битной точностью.

Опция Сброса (C) обнуляет регистр *MRb* после записи его значения в *Rs* и до сложения с новым результатом умножения. Режим насыщения всегда включён для операции пересылки. При пересылке регистра *MRb* в регистровый файл всегда происходит усечение (переполнение не пересылается).

Если результаты операции сохраняются в двойном регистре (MR3:2 или MR1:0), то дополнительные 4 бита размещаются в MR4 согласно описанию в пункте «Регистр переполнения результата умножителя (MR4)». Если результаты операции сохраняются в квадрорегистре (MR3:0), то дополнительные 8 бит размещаются в MR4 согласно описанию в пункте «Регистр переполнения результата умножителя (MR4)». Режим насыщения всегда включён.

Когда регистр-источник в пересылке является квадро-регистром (MR3:0), четыре 40-битных значения накопления усекаются в соответствии с типом данных (целым или дробным) и пересылаются в *Rsd.*

Когда регистр MR источника в пересылке является регистровой парой (MR1:0, MR3:2), четыре 20-битных значения накопления пересылаются в *Rsd.*

Флаги состояния

MZ Не изменяется;
MN Не изменяется;
MV Не изменяется;

MU (MUS) Не изменяется;

MOS Установлен или сброшен в соответствии с конечным результатом сложения, типом и размером данных (см. ниже).

Для результатов в виде обычных слов (MR3:0+=Rmd*Rnd) флаг MOS устанавливается по следующим правилам:

- для знаковых дробных операндов – установлен, если конечный результат больше либо равен 2^7 или меньше -2^7;
- для знаковых целых операндов – установлен, если конечный результат больше либо равен 2^{39} или меньше -2^{39};
- для беззнаковых дробных операндов – установлен, если конечный результат больше либо равен 2^8;
- для беззнаковых целых операндов – установлен, если конечный результат больше либо равен 2^{40}.

Для результатов в виде коротких слов (MR1:0/3:2+=Rmd * Rnd) флаг MOS устанавливается по правилам:

- для знаковых дробных операндов – установлен, если конечный результат больше либо равен 2^3 или меньше -2^3;
- для знаковых целых операндов - установлен, если:
  - конечный результат больше либо равен 2^{19} или меньше -2^{19},
  - промежуточный результат умножения больше либо равен 2^{15} или меньше -2^{15};
- для беззнаковых дробных операндов – установлен, если конечный результат больше либо равен 2^4;
- для беззнаковых целых операндов - установлен, если:
  - конечный результат больше либо равен 2^{20};
  - промежуточный результат умножения больше либо равен 2^{16}.

Опции

(I) Операнды целые;

(U) Rm, Rn – беззнаковые;

(C) Обнулить MR после пересылки усечённого значенияв регистровый файл (до накопления);

(CR) Обнулить MR после пересылки округлённого значения в регистровый файл (до накопления);

(NF) Не обновлять флаги(состояния);

Более подробно о возможных опциях см. подраздел «Опции команд умножителя».

Примеры

Листинг 58 – Пример 1 квадро-умножения с накоплением и пересылкой MR (двойная операция, короткие слова)

```
R0 = 0x00080007 ;;
R1 = 0x00060005 ;;
R2 = 0x00040003 ;;
R3 = 0x00020001 ;;
R5:4 = MR1:0, MR1:0 += R1:0* R3:2 (UIC) ;;
R5:4 = MR1:0, MR1:0 += R1:0* R3:2 (UI) ;
```

Пусть предыдущими значениями регистров MR были:

MR0 = 0x00800050

MR1 = 0x01000200
MR4 = 0x00000001

Тогда в примере в листинге 58 после выполнения первой команды умножения и пересылки регистров у изменившихся регистров окажутся такие значения:
R4 = 0x0080FFFF
R5 = 0x01000200
MR0 = 0x00200015
MR1 = 0x000C0005
MR4 = 0x00000000

После исполнения следующей команды регистры, приведённые выше, получат новые значения:
R4 = 0x00200015
R5 = 0x000C0005
MR0 = 0x0040002A
MR1 = 0x0018000A
MR4 = 0x00000000

Во время выполнения этих операций флаги не устанавливались.

Листинг 59 – Пример 2 квадро-умножения с накоплением и пересылкой MR (двойная операция, короткие слова)

```
R0 = 0x00080007 ;;
R1 = 0x00060005 ;;
R2 = 0x00040003 ;;
R3 = 0x00020001 ;;
R5:4 = MR3:2, MR3:2 += R1:0 * R3:2 (UIC) ;;
R5:4 = MR3:2, MR3:2 += R1:0 * R3:2 (UI) ;;
```

Пусть предыдущими значениями регистров MR были:
MR2 = 0x00300040
MR3 = 0x00500060
MR4 =0x00000001

Тогда в примере в листинге 59 после выполнения первой команды умножения и пересылки регистров у изменившихся регистров окажутся такие значения:
R4 = 0x00300040
R5 = 0x00500060
MR2 = 0x00200015
MR3 = 0x000C0005

После исполнения следующей команды регистры, приведённые выше, получат новые значения:
R4 = 0x00200015
R5 = 0x000C0005
MR2 = 0x0040002A
MR3 = 0x0018000A

Во время выполнения этих операций флаги не устанавливались.

Листинг 60 – Пример 3 квадро-умножения с накоплением и пересылкой MR (двойная операция, короткие слова)

```
R0 = 0x85006300 ;;
R1 = 0x00060005 ;;
R2 = 0x00100020 ;;
R3 = 0x00020001 ;;
R5:4 = MR3:0, MR3:0 += R1:0 * R3:2 (UIC) ;;
R5:4 = MR3:0, MR3:0 += R1:0 * R3:2 (UI) ;;
```

Пусть предыдущими значениями регистров MR были:

```
MR0 = 0x00000050
MR1 = 0x00000200
MR2 = 0x00000040
MR3 = 0x00000060
MR4 = 0x00000000
```

Тогда в примере в листинге 60 после выполнения первой команды умножения и пересылки регистров у изменившихся регистров окажутся такие значения:

```
R4   = 0x02000050
R5   = 0x00600040
MR0 = 0x000C6000
MR1 = 0x00085000
MR2 = 0x00000005
MR3 = 0x0000000C
MR4 = 0x00000000
```

После исполнения следующей команды регистры, приведённые выше, получат новые значения:

```
R4 = 0xFFFFFFFF
R5 =0x000C0005
MR0 = 0x00180000
MR1 = 0x0000000A
MR2 = 0x0000000A
MR3 = 0x00000018
MR4 = 0x00000000
```

Во время выполнения этих операций флаги не устанавливались.

10.4.7 *Комплексное умножение с накоплением (коротких слов)*

Синтаксис

{X|Y|XY} *MRa* +=*Rm* ** *Rn* {({I}{C|CR}{J}{NF})} ;

{X|Y|XY} *MRa* - = *Rm* ** *Rn* {({I}{C|CR}{J}{NF})} ;

/* здесь *MRa* либо MR1:0, либо MR3:2 */

Действие

Эта команда выполняет операцию 16-битного комплексного умножения с накоплением. При этом комплексное значение в регистре *Rm* умножается на комплексное значение в *Rn*, а затем полученный результат прибавляется к значению или вычитается из значения в указанном регистре MR. Результаты операции помещаются в тот же регистр накопления MR. Режим насыщения всегда включён. См. пункт «Опция Комплексное сопряжение» и рисунок 165.

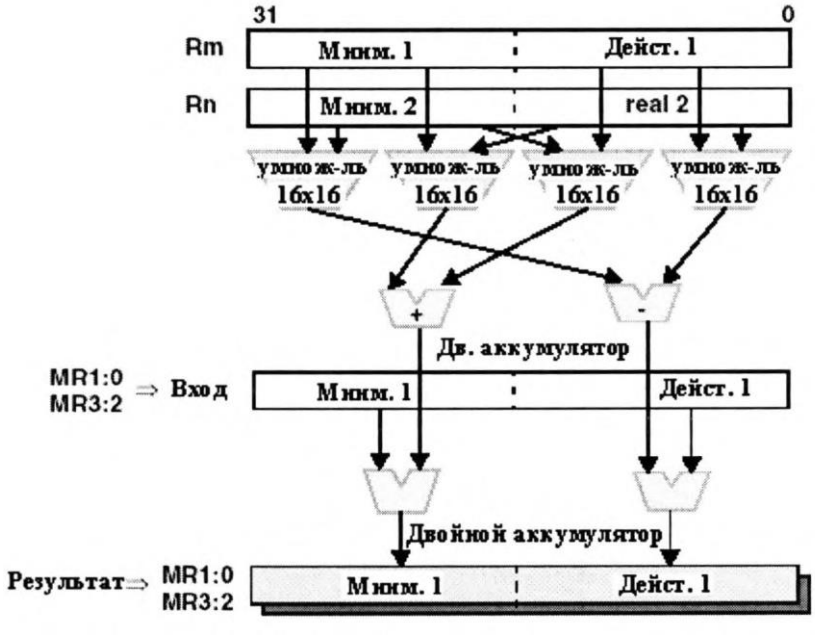


Рисунок 165 – Выполнение команды комплексного умножения с накоплением (коротких слов)

Для действительной и мнимой частей значения накопления имеются по 8 битов переполнения. Для команды MR1:0 += Rm ** Rn биты 7:0 регистра MR4 содержат биты переполнения для действительной части значения накопления, а биты 15:8 содержат биты переполнения для мнимой части значения накопления. Для команды MR3:2+= Rm ** Rn биты 23:16 регистра MR4 содержат биты переполнения для действительной части значения накопления, а биты 31:24 содержат биты переполнения для мнимой части значения накопления.

Важно отметить, что опция сопряжения (J) не выполняет операцию комплексного сопряжения над результатом. Если рассматривать комплексное умножение (A+jB)* (C+jD), то обычным сопряжённым значением к его результату будет (A-jB)*(C-jD). Однако, на процессоре опция сопряжения даст результат вида (A+jB)*(C-jD). Таким образом, результатом сопряжённого комплексного умножения будет комплексное произведение первого операнда на комплексное число, сопряжённое ко второму операнду.

Флаги состояния

MZ	Не изменяется
MN	Не изменяется
MV	Не изменяется
MU (MUS)	Не изменяется
MOS	Установлен или сброшен в соответствии с конечным результатом сложения, типом и размером данных (см. ниже).

- Для знаковых дробных операндов – установлен, если действительная или мнимая часть конечного результата больше либо равна 2^7 или меньше -2^7.
- Для знаковых целых операндов – установлен, если действительная или мнимая часть конечного результата больше либо равна 2^{39} или меньше -2^{39}.

Опции

(U)	Операнды целые
(C)	Обнулить MR

(CR) Обнулить и округлить
(J) Умножение значения в *Rm* на комплексно сопряжённое к значению в *Rn*
(NF) Не обновлять флаги (состояния)

Более подробно о возможных опциях см. подраздел «Опции команд умножителя».

Примеры

Листинг 61 – Пример 1 комплексного умножения с накоплением (коротких слов)

```
R0 = 0xFFF50009;;
R1 = 0x00060004;;
MR1: 0 += R0 ** R1 (IC) ;;
```

Пусть предыдущими значениями регистров MR были:
MR0 = 0x00800050
MR1 = 0x01000200
MR4 =0x00000001

Тогда в примере в листинге 61 после выполнения команды комплексного умножения с накоплением у изменившихся регистров окажутся такие значения:
MR0 = 0x00000066
MR1 = 0x0000000A
MR4 = 0x00000000

Флаги не устанавливаются.

Листинг 62 – Пример 2 комплексного умножения с накоплением (коротких слов)

```
R0 = 0xFFF50009;;
R1 = 0x00060004;;
MR3:2 += R0 ** R1 (ICJ);;
```

Пусть предыдущими значениями регистров MR были:
MR2 = 0x00000040
MR3 = 0x00000060
MR4 =0x00000001

Тогда в примере в листинге 62 после выполнения команды комплексного умножения с накоплением у изменившихся регистров окажутся такие значения:
MR2 = 0xFFFFFFE2
MR3 = 0xFFFFFF9E
MR4 = 0xFFFF0001

В результате операции флаги не устанавливаются.

10.4.8 *Комплексное умножение с накоплением и пересылкой регистра MR (двойная операция, короткие слова)*

Синтаксис

```
{X|Y|XY}Rs = MRa, MRa += Rm ** Rn {({I}{C|CR){J}{NF})}; (двойная операция)
{X|Y|XY}Rsd = MRa, MRa += Rm ** Rn {({I}{C}{J){NF})} ; (двойная операция)
/* здесь MRa либо MR1:0, либо MR3:2 */
```

Действие

Эта команда выполняет операцию 16-битного комплексного умножения с накоплением. При этом комплексное значение в регистре *Rm* умножается на комплексное значение в *Rn,* полученный результат прибавляется к значению или вычитается из значения в указанном регистре MR, и предыдущее содержимое регистра *MRa* пересылается в регистр *Rs.* Опция сброса (C) препятствует тому, чтобы новое значение умножения прибавлялось к старому значению в регистре *MRa*, обнуляя последний. Операции всегда происходят с усечением.

Как показано на рисунке 166 при пересылке регистра MR в регистровый файл 64-битное комплексное значение из регистровой пары MR3:2 или MR1:0 (32-битная действительная и 32-битная мнимая части) записывается в регистр приёмника *Rs* или регистровую пару-приёмник *Rsd.* В случае регистра приёмника *Rs* 32-битное значение действительной части из регистра MR2 или MR0 пересылается в 16 LSB регистра *Rs,* а мнимая часть из регистра MR3 или MR1 – в 16 MSB регистра *Rs.* В случае регистровой пары-приёмника *Rsd* 32-битное значение действительной части из регистра MR2 или MR0 пересылается в младший регистр регистровой пары *Rsd,* а мнимая часть из регистра MR3 или MR1 – в старший регистр регистровой пары *Rsd.*

Если в одиночный регистр *Rs* пересылаются целые данные, то младшие 16 битов действительной и мнимой частей записываются в регистр *Rs.* Если 40-битное значение действительной части знаковых целых данных, расположенное в MR2 или MR0, выходит за границы интервала (2^{15}, -2^{15}), то при пересылке оно не может быть представлено 16 битами, и происходит насыщение. То же правило насыщения применимо и к беззнаковым целым данным, знаковым/беззнаковым дробным данным, а также к мнимой части таких данных, расположенной в регистрах MR3 и MR1. При пересылке дробных данных старшие 16-ти битов действительной и мнимой частей записываются в регистр приёмника.

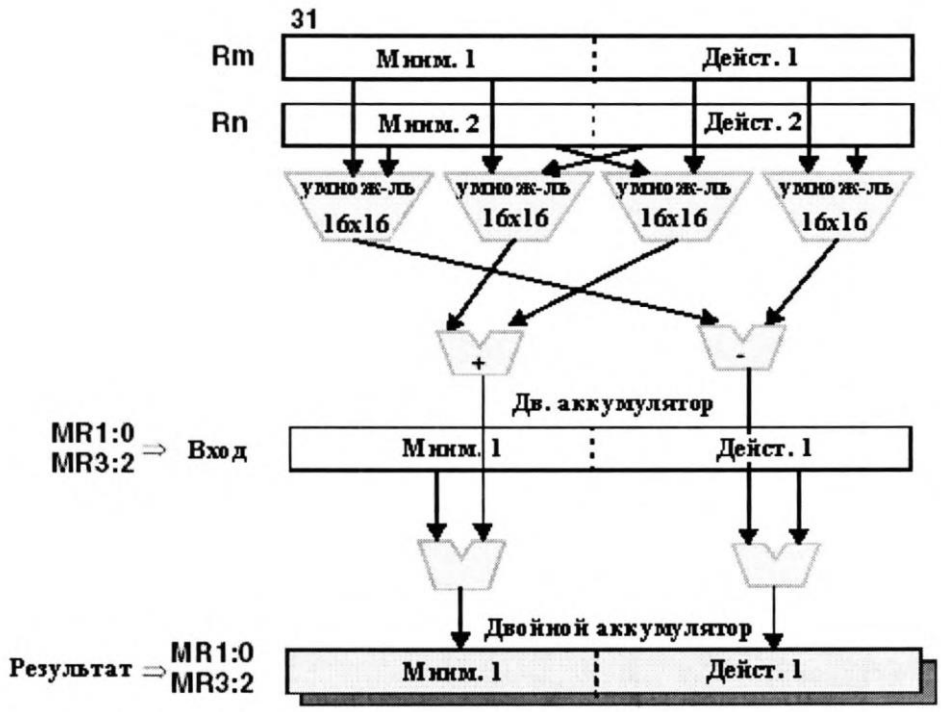


Рисунок 166 – Выполнение команды комплексного умножения с накоплением и пересылкой регистра MR (двойная операция, короткие слова)

Флаги состояния

MZ Не изменяется;

MN Не изменяется;
MV Не изменяется;
MU (MUS) Не изменяется;
MOS Установлен или сброшен в соответствии с конечным результатом сложения, типом и размером данных:

- для знаковых дробных операндов – установлен, если действительная или мнимая часть конечного результата больше либо равна 2^7 или меньше -2^7;
- для знаковых целых операндов – установлен, если действительная или мнимая часть конечного результата больше либо равна 2^{39} или меньше -2^{39}.

Опции

(I) Операнды целые;
(C) Обнулить MR после пересылки в регистровый файл, но до накопления;
(CR) Обнулить и округлить;
(J) Сопряжение;
(NF) Не обновлять флаги (состояния).

Более подробно о возможных опциях см. подраздел «Опции команд умножителя».

Примеры

Листинг 63 – Пример 1 комплексного умножения с накоплением и пересылкой MR (двойная операция, короткие слова)

```
R0 = 0xFFF50009 ;;
R1 =0x00060004 ;;
R4 = MR1:0, MR1:0 += R0 ** R1 (IC) ;;
R4 = MR1:0, MR1:0 += R0 ** R1 (I) ;;
```

Пусть предыдущими значениями регистров MR были:
MR0 = 0x00800050
MR1 = 0x01000200
MR4 = 0x00000001

Тогда в примере в листинге 63 после выполнения первой команды комплексного умножения с накоплением и пересылкой у изменившихся регистров окажутся такие значения:
R4 = 0x7FFF7FFF
MR0 = 0x00000066
MR1 = 0x0000000A
MR4 = 0x00000000

В результате операции флаги не устанавливаются.

После исполнения следующей команды содержимым этих регистров будет:
R4 = 0x000A0066
MR0 = 0x000000CC
MR1 = 0x00000014
MR4 = 0x00000000

Флаги не устанавливаются.

Листинг 64 - Пример 2 комплексного умножения с накоплением и пересылкой MR (двойная операция, короткие слова)

```
R0 = 0xFFF50009;;
R1 = 0x00060004;;
R4 = MR3:2, MR3:2 += R0 ** R1 (ICJ);;
R4 = MR3:2, MR3:2 += R0 ** R1 (IJ);;
```

Пусть предыдущими значениями регистров MR были:

```
MR2 = 0x00000040
MR3 = 0x00000060
MR4 = 0x00000001
```

Тогда в примере в листинге 64 после выполнения первой команды комплексного умножения с накоплением и пересылкой у изменившихся регистров окажутся такие значения:

```
R4 = 0x00600040
MR2 = 0xFFFFFFE2
MR3 = 0xFFFFFF9E
MR4 = 0xFFFF0001
```

В результате операции флаги не устанавливаются.

После исполнения следующей команды содержимым этих регистров будет:

```
R4 = 0xFF9EFFE2
MR2 = 0xFFFFFFC4
MR3 = 0xFFFFFF3C
MR4 = 0xFFFF0001
```

Флаги не устанавливаются.

Листинг 65 – Пример 3 комплексного умножения с накоплением и пересылкой MR (двойная операция, короткие слова)

```
R0 = 0xFFF50009;; R1 = 0x00060004;;
R5:4 = MR1:0, MR1:0 += R0 ** R1 (IC);;
R5:4 = MR1:0, MR1:0 += R0 ** R1 (I);;
```

Пусть предыдущими значениями регистров MR были:

```
MR0 = 0x00800050
MR1= 0x01000200
MR4 =0x00000001
```

Тогда в примере в листинге 65 после выполнения первой команды комплексного умножения с накоплением и пересылкой у изменившихся регистров окажутся такие значения:

```
R4 = 0x7FFFFFFF
R5 = 0x01000200
MR0 = 0x00000066
MR1 = 0x0000000A
MR4 = 0x00000000
```

В результате операции флаги не устанавливаются.

После исполнения следующей команды содержимым этих регистров будет:

```
R4 = 0x00000066
R5 = 0x0000000A
```

MR0 = 0x000000CC
MR1 = 0x00000014
MR4 = 0x00000000

Флаги не устанавливаются.

Листинг 66 – Пример 4 комплексного умножения с накоплением и пересылкой MR (двойная операция, короткие слова)

```
R0 = 0xFFF50009;;
R1 = 0x00060004;;
R5:4 = MR3:2, MR3:2 += R0 ** R1 (ICJ);;
R5:4 = MR3:2, MR3:2 += R0 ** R1 (IJ);;
```

Пусть предыдущими значениями регистров MR были:
MR2 = 0x00000040
MR3= 0x00000060
MR4 =0x00000001

Тогда в примере в листинге 66 после выполнения первой команды комплексного умножения с накоплением и пересылкой у изменившихся регистров окажутся такие значения:
R4 = 0x00000040
R5 = 0x00000060
MR2 = 0xFFFFFFE2
MR3 = 0xFFFFFF9E
MR4 = 0xFFFF0001

В результате операции флаги не устанавливаются.
После исполнения следующей команды содержимым этих регистров будет:
R4 = 0xFFFFFFE2
R5 = 0xFFFFFF9E
MR2 = 0xFFFFFFC4
MR3 = 0xFFFFFF3C
MR4 = 0xFFFF0001

Флаги не устанавливаются.

10.4.9 *Умножение (плавающая арифметика, обычных/расширенных слов)*

Синтаксис

{X|Y|XY}F*Rs* = *Rm* * *Rn* {({T}{NF})} ; /* Обычные 32-битные слова */

{X|Y|XY}F*Rsd* = *Rmd* * *Rnd* {({T}{NF})} ; /* Расширенные 40/64-битные слова */

Действие

Это 32-, 40-, 64-битное умножение операндов в формате с плавающей точкой в регистрах *Rm* и *Rn.* Результат операции помещается в регистр *Rs* и округляется, если не указана опция T. Умножение с одиночными регистрами – это 32-битное умножение чисел в формате с плавающей точкой по стандарту IEEE. Двойные регистры в команде обозначают 40-битное умножение операндов в формате с плавающей точкой расширенной точности либо 64-битное умножение чисел двойной точности в формате с плавающей точкой по стандарту IEEE.

Флаги состояния

MZ Установлен, если результат есть ±ноль в формате с плавающей точкой;

MN Установлен, если результат отрицателен;

MV Установлен, если несмещённое значение порядка результата больше 127;

MVS Установлен, если несмещённое значение порядка результата больше 127; в противном случае не изменяется;

MU Установлен, если несмещённое значение порядка результата меньше -126;

MUS Установлен, если несмещённое значение порядка результата меньше -126; в противном случае не изменяется;

MI Установлен, если хотя бы один операнд есть NAN;

MIS Установлен, если хотя бы один операнд есть NAN; в противном случае не изменяется.

Опции

( ) Округление;

(T) Усечение;

(NF) Не обновлять флаги (состояния).

Примеры

Листинг 67 – Пример 1 умножения (плавающая арифметика, обычное/расширенное слово)

```
R0 = 0x3AF5EC81 ;; /* 0.00187625 */
R1 = 0x3AF58CDF ;; /* 0.0018734 */
FR2 = R0 * R1 ;;
```

В примере в листинге 67 после исполнения команды плавающего умножения результат в регистре R2 равен 0x366BE2AB (3.51497E-006).

10.4.10 *Загрузка/Пересылка регистра результата умножения (MR)*

Синтаксис

{X|Y|XY}{S|L}MRa = *Rmd* { ({SE|ZE}{NF})} ;

{X|Y|XY}MR4 = *Rm* {(NF)} ;

{X|Y|XY}{S) *Rsd* = MRa { ({U} {S } {NF}) } ;

{X|Y|XY} *Rsq* = MR3:0 { ({U} { S } {NF}) } ;

{X|Y|XY} *Rs* = MR4 ;

/* здесь *MRa* либо MR1:0, либо MR3:2 */

Действие

Эти команды пересылают значение из регистра-источника (справа от знака =) в регистр-приёмник (слева от знака =).

Размер регистра-приёмника MR больше, чем размер регистра-источника *Rmd.* Есть две возможности заполнения избыточных битов в MR4: путём распространения знака (SE) или путём дополнения нулями (ZE). Имеются следующие операции:

- S*MRa* обозначает пересылку четырёх 16-битных коротких слов в четыре 20-битных приёмника;
- *MRa* обозначает пересылку двух 32-битных обычных слов в два 40-битных приёмника;
- *LMRa* обозначает пересылку одного 64-битного длинного слова в один 80-битный приёмник.

Так как регистр MR, используемый в качестве источника, может содержать насыщенный операнд, больший, чем помещается в регистре приёмника *Rm,* процессор поддерживает работу с несовпадающим размером данных. При включённой опции насыщения, если операнд слишком велик для регистра *Rs,* происходит насыщение с учётом битов MR4.

- Rsd обозначает пересылку одного 80-битного операнда в 64-битный регистр;
- Rsq обозначает пересылку четырёх 40-битных операндов в четыре 32-битных регистра;
- SRsd обозначает пересылку четырёх 20-битных операндов в четыре 16-битных коротких слова.

См. поясняющие примеры на рисунках 167, 168 и 169.

Rsd = MR1: 0; см. рисунок 167

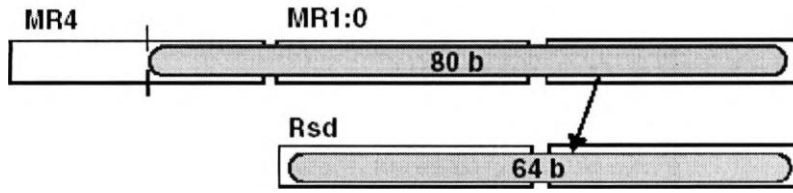


Рисунок 167 – Пересылка одного 80-битного операнда в один 64-битный регистр

SRsd = MR1:0; см. рисунок 168

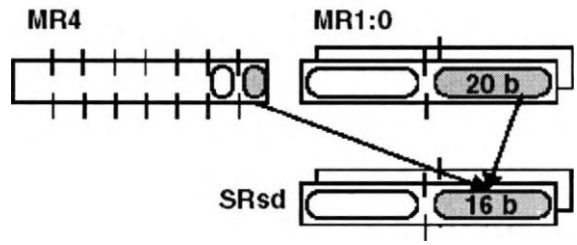


Рисунок 168 – Пересылка четырёх 20-битных операндов в четыре 16-битных коротких слова

Rsq = MR3:0; см. рисунок 169

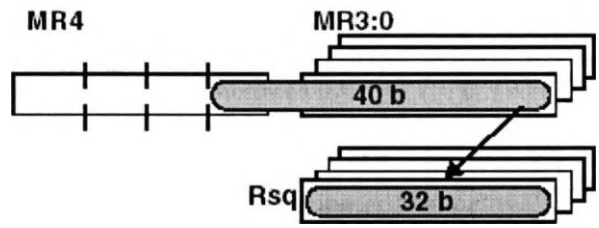


Рисунок 169 – Пересылка четырёх 40-битных операндов в четыре 32-битных регистра

Флаги состояния

MZ Для команды Rs = MR установлен, если результат равен нулю; для команды MR = Rs сброшен;

MN Для команды Rs = MR установлен, если результат отрицателен; длякоманды MR = Rs сброшен;

MV Для команды Rs = MR установлен, если полное значение в MR и MR4 даёт переполнение; для команды MR = Rs сброшен;

MOS Для команды Rs = MR установлен, если полное значение в MR и MR4 даёт переполнение, в противном случае не изменяется; для команды MR = Rs сброшен;

MU (MUS) Сброшен (не изменяется);

MI Сброшен.

Опции

(SE) Распространять знак;

(Z F) Дополнить нулями;

(U) Беззнаковое насыщение;

(NF) Не обновлять флаги (состояния);

Более подробно о возможных опциях см. подраздел «Опции команд умножителя».

Примеры

Во всех примерах в этой главе начальные значения в регистрах MR таковы:

MR0 = 0x00000000
MR1 = 0x11111111
MR2 = 0x22222222
MR3 = 0x0FFFFFF5
MR4 = 0x0000FF00

Листинг 68 – Пример 1 загрузки/пересылки регистра результата умножителя MR

```
R1:0 = MR3:2 (S);;
```

В примере в листинге 68 после выполнения команды пересылки MR3:2 содержимым R1:0 будет:

R0 = 0x22222222
R1 = 0x0FFFFFF5

Флаги не устанавливаются.

Листинг 69 – Пример 2 загрузки/пересылки регистра результата умножителя MR

```
R3:2 = MR3:2;;
```

В примере в листинге 69 после выполнения команды пересылки MR3: 2 содержимым R3:2 будет:

R2 = 0x22222222 R3 = 0x0FFFFFF5

Флаги не устанавливаются.

Листинг 70 – Пример 3 загрузки/пересылки регистра результата умножителя MR

```
R1:0 = MR1:0 (S) ;;
```

В примере в листинге 70 после выполнения команды пересылки MR1:0 содержимым R1:0 будет:

R0 = 0x00000000 R1 = 0x80000000

Установлены флаги MV, MN и MOS.

Листинг 71 – Пример 4 загрузки/пересылки регистра результата умножителя MR

```
R3:2 = MR1: 0 (U) ;;
```

В примере в листинге 71 после выполнения команды пересылки MR1:0 содержимым R3:2 будет:
R2 = 0x00000000 R3 = 0x11111111

Флаги MV и MOS устанавливаются.

Листинг 72 – Пример 5 загрузки/пересылки регистра результата умножителя MR

```
SR1:0 = MR3:2 (S) ;;
```

В примере в листинге 72 после выполнения команды пересылки MR3:2 содержимым R1: 0 будет:
R0 = 0x22222222 R1 = 0x0FFF7FFF

Флаги MV и MOS установлены.

Листинг 73 – Пример 6 загрузки/пересылки регистра результата умножителя MR

```
SR3:2 = MR3:2 (U);;
```

В примере в листинге 73 после выполнения команды пересылки MR3:2 содержимым R3:2 будет:
R2 = 0x22222222 R3 = 0x0FFFFFF5

Флаги не устанавливаются.

Листинг 74 – Пример 7 загрузки/пересылки регистра результата умножителя MR

```
SR1:0 = MR1:0 (S);;
```

В примере на листинге 74 после выполнения команды пересылки MR1:0 содержимым R1: 0 будет:
R0 = 0x00000000 R1 = 0x80008000

Установлены флаги MV, MN и MOS.

Листинг 75 – Пример 8 загрузки/пересылки регистра результата умножителя MR

```
SR3:2 = MR1:O (U);;
```

В примере на листинге 75 после выполнения команды пересылки MR1:0 содержимым R3:2 будет:
R2 = 0x00000000 R3 = 0x11111111

Флаги MZ, MV и MOS установлены.

Листинг 76 – Пример 9 загрузки/пересылки регистра результата умножителя MR

```
R3:0 = MR3:0 (S);;
```

В примере на листинге 76 после выполнения команды пересылки MR3:0 содержимым R3:0 будет:
R0 = 0x00000000
R1= 0x80000000

R2 =0x22222222
R3 =0x0FFFFFF5

Флаги MV, MN, MZ и MOS установлены.

Листинг 77 – Пример 10 загрузки/пересылки регистра результата умножителя MR

```
R3:0 = MR3:0 (U) ;;
```

В примере на листинге 77 после выполнения команды пересылки MR3:0 содержимым R3:0 будет:
R0 = 0x00000000
R1 =0x11111111
R2= 0x22222222
R3 =0x0FFFFFF5

Флаги MV, MZ и MOS установлены.

Листинг 78 – Пример 11 загрузки/пересылки регистра результата умножителя MR

```
R0 = 0x11111111 ;;
R1 = 0x22222222 ;;
R2 = 0x33333333 ;;
R3 = 0x44444444 ;;
R5 = 0x55555555 ;;
MR3:2 = R1:0;;
MR1:0 = R3:2;;
MR4 = R5 ;;
```

В примере в листинге 78 после выполнения команд загрузки MR3:2, MR1:0 и MR4 все установленные флаги умножителя (за исключением залипающих) сбросятся.

10.4.11 *Извлечение слов из регистра результата умножителя (MR)*

Синтаксис

```
{X|Y|XY}Rsd = SMRb {({U}{NF})} ; /* извлечение 2-х коротких слов */
{X|Y|XY}L Rsd = MRb {({U}{NF})} ; /* извлечение 1-ого обычного слова */
/* здесь MRb есть MR0, MR1, MR2 или MR3 */
{X|Y|XY} Rsq = SMRa {({U} {NF})} ; /* извлечение 4-х коротких слов */
{X|Y|XY}L Rsq - MRa {({U}{NF})} ; /* извлечение 2-х обычных слов */
{X|Y|XY}Q Rsq = L MRa {({U}{NF})}; /* извлечение 1-ого длинного слова */
/* здесь MRa либо MR1:0, либо MR3:2 */
```

Действие

Эти команды извлекают значение(я) из регистра приёмника MRb или MRa (справа от знака =), а также дополнительные биты из регистра MR4, затем сохраняют полученные данные в регистре приёмника Rsd или Rsq (слева от знака =). Если опция (U) не указана, то данные расширяются знаком.

Как отмечено в комментариях к синтаксису, в зависимости от используемого вида команды, регистры источника MRb или MRa и приёмника Rsd или Rsq могут содержать разное количество коротких (16-битных), обычных (32-битных) или длинных (64-битных) слов. Более подробную информацию можно найти в разделе «Примеры», а также в примерах, поясняющих расположение битов в регистрах.

XRsd = SMR0; см. рисунок 170

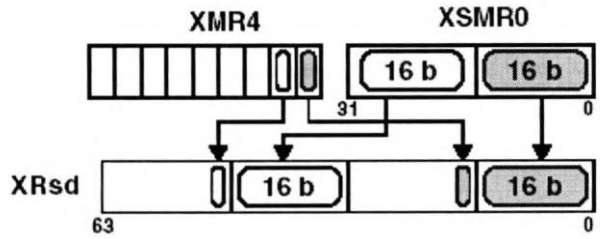


Рисунок 170 – Извлечение/Сохранение двух коротких значений
и их дополнительных битов

XLRsd = MR0; см. рисунок 171

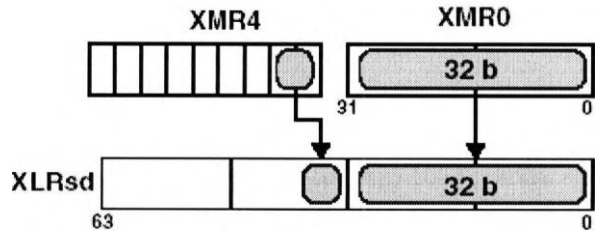


Рисунок 171 – Извлечение/Сохранение одного обычного значения
и его дополнительных битов

XRsq = SMR1: 0; см. рисунок 172

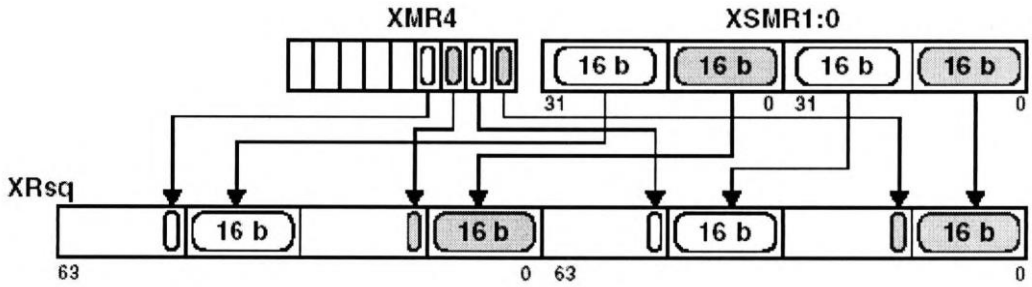


Рисунок 172 – Извлечение/Сохранение четырёх коротких значений
и их дополнительных битов

XLRsq = MR1: 0; см. рисунок 173

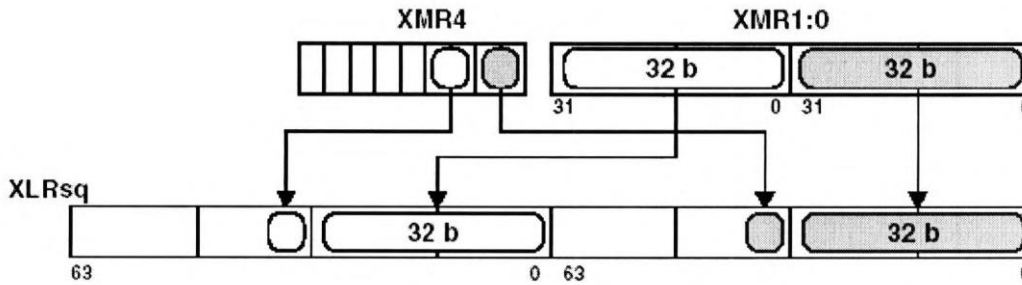


Рисунок 173 – Извлечение/Сохранение двух обычных значений
и их дополнительных битов

XQRsq = LMR1: 0; см. рисунок 174

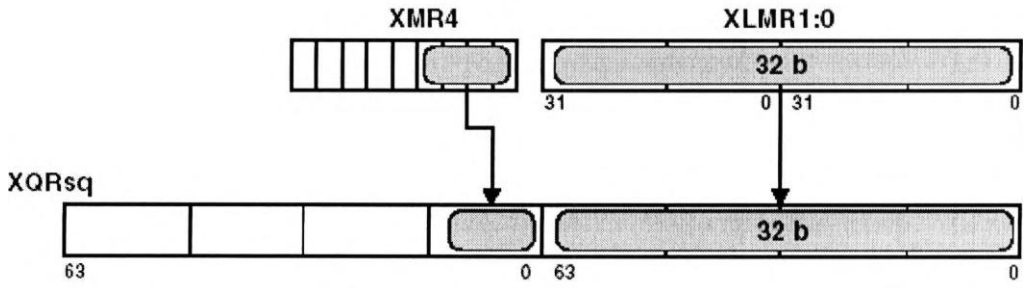


Рисунок 174 – Извлечение/Сохранение одного длинного значения и его дополнительных битов

Флаги состояния

(NF) Не обновлять флаги (состояния)

Опции

MZ Установлен, если операнд (или один из операндов, если тип данных меньше размера данных (например, для команды Rsd = SMR)) равен нулю;

MN Установлен, если операнд отрицателен (не установлен при беззнаковой операции, опция (U));

MV Сброшен (в этих командах переполнение не может иметь место) Не изменяется (в этих командах переполнение не может иметь место);

MOS Сброшен (не изменяется);

MU Беззнаковое.

Более подробно о возможных опциях см. подраздел «Опции команд умножителя».

Примеры

Листинг 79 – Пример 1 извлечения слов из регистра результата умножителя (MR)

```
R1:0 = SMR2 ;;
/* Если MR2 = 0x30f26ab2 и MR4 = 0x00d30000, */
/* то R1 = 0xfffd30f2 и R0 = 0x00036ab2 */
```

Команда *Rsd* = SMR0/1/2/3 (U) ; (см. листинг 79) извлекает два коротких слова из одиночного регистра MR, а также их дополнительные биты в MR4 и помещает их в двойной регистр. Если опция (U) указана, то данные при этом дополняются нулями. Если в листинг 79 добавить опцию (U), то значения регистров будут: R1 = 0x000d30f2 и R0 = 0x00036ab2.

Команда *Rsq* = SMR3: 2/1:0 (U) ; аналогична команде *Rsd* = SMR0/1/2/3 (U);, но выполняет извлечение четырёх коротких слов из двойного регистра MR, а также извлечение их дополнительных битов из регистра MR4.

Листинг 80 – Пример 2 извлечения слов из регистра результата множителя (MR)

```
LR1: 0 = MR2 ;;
/* Если MR2 = 0x30f26ab2 и MR4 = 0x00d30000, */
/* то R1:0 = 0xffffffd330f26ab2 */
```

Команда L*Rsd* = MR0/1/2/3 (U) ; (см. листинг 80) извлекает обычное слово из одиночного регистра MR, а также его дополнительные биты в MR4 и помещает их в двойной регистр. Если опция (U) указана, то данные при этом дополняются нулями. Если в листинг 80 добавить опцию (U), то значения регистров будут:

R1:0 = 0xffffffd330f26ab2.

Команда *LRsq* = MR3: 2/1:0 (U) ; аналогична команде LRsd = MRO/1/2/3 (U) ;, но выполняет извлечение двух обычных слов из двойного регистра MR, а также извлечение их дополнительных битов из регистра MR4.

10.4.12 *Сжатие регистра результата умножителя (MR)*

Синтаксис

{X|Y|XY} *Rs* = COMPACT *MRa* {({U}{I}{ S}{NF})} ;

{X|Y|XY}S *Rsd* = COMPACT MR3: 0 { ({U} {I} {S} {NF}) } ;

/* здесь *MRa* либо MR1:0, либо MR3:2 */

Действие

Команда COMPACT *MRa* сжимает 80-битное значение в регистрах MR и MR4 в 32-битный результат. Если тип данных дробный, то результат всегда усекается. Операция сжатия регистра *MRa* представлена на рисунке 175.

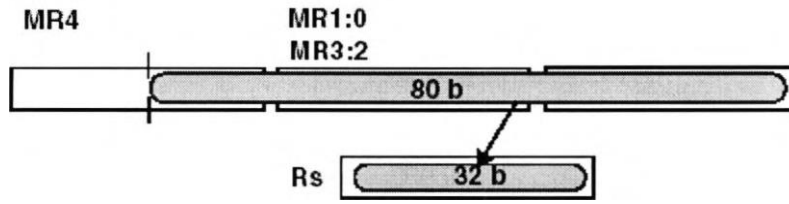


Рисунок 175 – Сжатие MRa

Команда COMPACT MR3:0 сжимает четыре 40-битных значения в MR3:0 и MR4 в четыре 16-битных результата. Если тип данных дробный, то результат всегда усекается. Операция сжатия регистра MR3:0 представлена на рисунке 176.

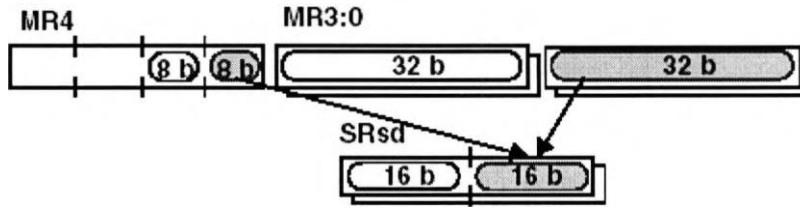


Рисунок 176 – Сжатие MR3:0

Флаги состояния

MZ	Установлен, если все биты результата равны нулю;
MN	Установлен, если результат отрицателен;
MV(MUS)	Установлен в соответствии с форматом данных (см. ниже; (MOS не изменяется, если MV сброшен));
MU(MOS)	Сброшен (не изменяется).

В случае однословного результата, MV устанавливается в следующих случаях:

- для знаковых дробных данных – если старшие 17 бит значения в MR и MR4 не все ноли и не все единицы;
- для знаковых целых данных – если старшие 49 бит значения в MR и MR4 не все ноли и не все единицы;
- для беззнаковых дробных данных – если старшие 16 бит значения в MR и MR4 не все ноли;
- для беззнаковых целых данных – если старшие 48 бит значения в MR и

MR4 не все ноли.

Опции

(I) Целое;
(S) Насыщение;
(U) Беззнаковое;
(NF) Не обновлять флаги (состояния).

Более подробно о возможных опциях см. подраздел «Опции команд умножителя».

Примеры

В примерах в листингах 81 и 82 текущее содержимое регистров MR таково:

MR2 = 0x22222222
MR3 = 0x0FFFFFF5
MR4 = 0x0000FF00

Листинг 81 – Пример 1 сжатия регистра умножителя MR

```
R0 = COMPACT  MR3:2 (UI) ;;
```

В примере в листинге 81 после выполнения команды COMPACT регистр R0 содержит 0x22222222, и флаги MV и MOS установлены.

Листинг 82 – Пример 2 сжатия регистра умножителя MR

```
R0 = COMPACT MR3:2 (U) ;;
```

В примере в листинге 82 после выполнения команды COMPACT регистр R0 содержит 0x0FFFFFF5, флаги не устанавливаются.

В примерах в листингах 83 и 84 текущее содержимое регистров MR таково:

MR0 = 0x00000000
MR1 = 0x11111111
MR2 = 0x22222222
MR3= 0x0FFFFFF5
MR4 =0x0000FF00

Листинг 83 – Пример 3 сжатия регистра умножителя MR

```
R0 = COMPACT MR1:0 (UI) ; ;
```

В примере в листинге 83 после выполнения команды COMPACT регистр R0 содержит 0x00000000, и флаги MZ, MV и MOS установлены.

Листинг 84 – Пример 4 сжатия регистра умножителя MR

```
R0 = COMPACT MR1:0 (U) ;;
```

В примере в листинге 84 после выполнения команды COMPACT регистр R0 содержит 0x11111111, и флаги MV и MOS установлены.

В примерах в листингах 85, 86 и текущее содержимое регистров MR таково:

MR0 = 0x00000000
MR1 = 0x11111111
MR2 = 0x22222222
MR3= 0x0FFFFFF5
MR4 =0x0000FF00

Листинг 85 – Пример 5 сжатия регистра умножителя MR

```
SR1:0 = COMPACT MR3:0 (UI) ;;
```

В примере в листинге 85 после выполнения команды COMPACT регистры R0, R1 содержат 0x11110000, 0xFFF52222, и флаги MZ, MV и MOS установлены.

Листинг 86 – Пример 6 сжатия регистра умножителя MR

```
SR1:0 = COMPACT MR3:0 (S) ;;
```

В примере в листинге 86 после выполнения команды COMPACT регистры R0, R1 содержат 0x80000000, 0x0FFF2222, и флаги MZ, MN, MV и MOS установлены.

10.4.13 *Векторные команды умножения для чисел плавающей запятой одинарной точности.*

Стандартная операция умножения для чисел плавающей запятой (ПЗ) одинарной точности выполняет операцию в одном вычислительном модуле только над одной парой чисел. В процессоре добавлена возможность выполнения операции умножения в одном вычислительном модуле над двумя парами чисел ПЗ одинарной точности. Базовый код операции умножения для чисел с ПЗ имеет вид:

31	30	29:28	27:26	25:24	23:20	19:15	14:10	9:5	4:0
EX	CC/NF	10	XY	11	1111	100Td	Rs	Rm	Rn

Поле opcode (биты 19:15) имеет фиксированное значение. Бит «T» выполняет функцию округления, а бит «d» задает операцию над числами с двойной точностью. Для векторных операций умножения используется код:

31	30	29:28	27:26	25:24	23:20	19:15	14:10	9:5	4:0
EX	CC/NF	10	XY	11	1111	00ST0	Rs	Rm	Rn

Функции бита «T» сохраняются, бит «d» всегда равен нулю и указывает на операнды одинарной точности.

Общая форма записи векторной команды умножения использует два варианта синтаксиса. Первый вариант считается устаревшим, рекомендуется использовать второй вариант. Общая форма записи имеет вид:

{X|Y|XY}SFRsd = Rmd * {Rnd | Rn} (mX)(nX) (T) (NF); (вариант 2)

{X|Y|XY}FRsd = Rm{H|L} * {-}Rn{H|L}, Rm{L|H} * {-}Rn{L|H} (NF); (вариант 1)

Опция mX меняет местами регистры в паре Rmd, а опция nX – в паре Rnd.

Бит “S” в поле команды выполняет дополнительную функцию:

Бит “S” равен нулю.

В этом случае две пары операндов перемножаются и формируют два результата. Если значение регистра в поле Rm или Rn нечетное, то это вызывает перестановку операндов в регистровой паре.

Бит “S” равен единице.

В этом случае операнд Rn для обоих умножений будет один и тот же. Его значение определяется кодом Rn и может быть как четным, так и нечетным регистром. В зависимости от того, четное значение регистра Rm или нет, возможна инверсия знака одного из множителей. Это может быть полезно, например, для умножения комплексного числа на мнимую часть другого комплексного числа. Форма записи:

{X|Y|XY}*SFRsd = Rmd * Rn* (mX) (T)(NF);

Таблица 55

Rm	Rn	Функция
четное	четное	Умножение каждого операнда регистровой пары Rmd на значение регистра Rn
четное	нечетное	Умножение каждого операнда регистровой пары Rmd на значение регистра Rn
нечетное	четное	Умножение старшего(левого) операнда регистровой пары Rmd на значение регистра -Rn. Умножение младшего(правого) операнда регистровой пары Rmd на значение регистра Rn
нечетное	нечетное	Умножение старшего(левого) операнда регистровой пары Rmd на значение регистра Rn. Умножение младшего(правого) операнда регистровой пары Rmd на значение регистра -Rn
Примечание – Если Rm нечётное, операнды меняются местами		

Из таблицы 55 следует, что в случае четного значения регистра Rm, мы имеем возможность умножить оба операнда на общую переменную. Если же значение Rm нечетное, мы получаем специальный вариант умножения с модификацией знака результата. Данный специальный случай позволяет реализовать процедуры комплексного умножения в зависимости от того в каких регистрах расположены действительная и мнимая части комплексного числа.

Итоговый список операций умножения над числами ПЗ приведен в таблице 56.

Таблица 56 – Коды операций умножения

Синтаксис	Тип	Код операции
FRs = Rm * Rn	1111	1 0 0T0
FRsd = Rmd * Rnd	1111	1 0 0 T 1
FRsd = RmH * RnH, RmL * RnL	1111	0 0 0 T 0
FRsd = RmH * RnL, RmL * RnH	1111	0 0 0 T 0
FRsd = RmL * RnH, RmH * RnL	1111	0 0 0 T 0
FRsd = RmL * RnL, RmH * RnH	1111	0 0 0 T 0
FRsd = RmH * Rn, RmL * Rn	1111	0 0 1 T 0
FRsd = RmH * -Rn, RmL * Rn	1111	0 0 1 T 0
FRsd = RmH * Rn, RmL * -Rn	1111	0 0 1 T 0

Второй вариант синтаксиса векторного умножения является предпочтительным по той причине, что в нём есть возможность использовать константу. Константа пишется вместо регистра Rnd. Также существует опция FJ, которая заставляет команду умножать по правилам регистра Rnd (полезно для комплексного умножения). Пример:

```
SFR5:4 = R1:0 * 5.0  (mX);;   // R5 = R0*5. R4 = R1*5

SFR5:4 = R1:0  * 5.0 (mX)(FJ);; // R5 = -R0*5. R4 = R1*5

SFR5:4 = R1:0  * 5.0 (mX)(nX)(FJ);; // R5 = R0*5. R4 = -R1*5
```

10.4.13.1 Векторное умножение (одинарная точность)

Синтаксис

{X|Y|XY}SFRsd = Rmd * {Rnd | Rn} {({mX}{nX}{T}{NF})}; (вариант 2)

{X|Y|XY}FRsd = Rm{H|L} * {-}Rn{H|L}, Rm{L|H} * {-}Rn{L|H} (NF); (вариант 1)

Действие

Это 32-битное умножение операндов в формате с плавающей точкой в регистре Rm и регистре Rn. Результат операции помещается в регистр Rs и округляется, если не указана опция T. Умножение чисел в формате с плавающей точкой выполняется согласно стандарта IEEE. Двойные регистры в команде обозначают умножение двух пар операндов с формированием двух результатов. Описанные ниже флаги состояния соответствуют алгоритму формирования набора флагов для одной пары операндов. Итоговое значение флагов состояния есть логическое поэлементное ИЛИ соответствующих значений флагов двух наборов.

Флаги состояния

MZ Установлен, если результат есть ±ноль в формате с плавающей точкой;
MN Установлен, если результат отрицателен;
MV Установлен, если несмещённое значение порядка результата больше 127;
MVS Установлен, если несмещённое значение порядка результата больше 127; в противном случае не изменяется;
MU Установлен, если несмещённое значение порядка результата меньше -126;
MUS Установлен, если несмещённое значение порядка результата меньше -126; в противном случае не изменяется;
MI Установлен, если хотя бы один операнд есть NAN;
MIS Установлен, если хотя бы один операнд есть NAN; в противном случае не изменяется.

Опции

( ) Округление;
(T) Усечение;
(NF) Не обновлять флаги (состояния);
(mX) Поменять местами регистры в паре Rmd;
(nX) Поменять местами регистры в паре Rnd;
(FJ) Комплексное умножение.

10.5 Команды сдвигового устройства

Сдвиговое устройство выполняет *побитовые операции* (арифметические и логические сдвиги) и *операции с битовыми полями* (извлечение и размещение полей) процессора. Также сдвиговым устройством выполняются *операции преобразования данных,* такие как преобразование форматов с фиксированной/плавающей точкой. Описание операций сдвигового устройства, его флагов состояния и предикатов, а также примеры можно найти в разделе «Сдвиговое устройство».

Соглашения по обозначениям имён регистров, необязательных элементов или вариантов, используемые в справках на команды, подробно освещены в подразделе «Регистры регистровых файлов». Вкратце эти соглашения таковы:

- { } – необязательные элементы указаны в фигурных скобках; фигурные скобки не являются частью синтаксиса команды;
- | – вертикальная черта разделяет варианты значений элементов; вертикальные черты не являются частью синтаксиса команды;
- Rmd – имена регистров, выделенные курсивом, представляют собой подставляемые шаблоны для имён одиночных (Rs, Rm или Rn), двойных (Rsd, Rmd или Rnd) или квадро- (Rsq, Rmq или Rnq) регистров.

Каждая команда, по которой дана справка, занимает одно поле команды в командной строке.

10.5.1 *Арифметический/Логический сдвиг*

Синтаксис

{X|Y|XY}{B|S} *Rs* = LSHIFT|ASHIFT *Rm* BY Rn|<Imm> {(NF)} ;

{X|Y|XY}{B|S|L} *Rsd* = LSHIFT|ASHIFT *Rmd* BY Rn|<Imm> {(NF)} ;

Действие

Эти команды проводят арифметический сдвиг (с распространением знака) или логический сдвиг (без распространения знака) операнда в регистре *Rm* на число битов, определяемое операндом в регистре *Rn* или непосредственным значением в команде. Положительное значение в *Rn* означает сдвиг влево, а отрицательное значение в *Rn* означает сдвиг вправо Результат сдвига помещается в регистр результата *Rs.* Префиксы L, S и B указывают тип операнда, а суффикс d указывает на двойной размер регистра.

Все значения параметра сдвига – это числа в дополнительном представлении. Положительные значения дают сдвиг влево, отрицательные значения дают сдвиг вправо. Величина параметра сдвига для операций с параметром в регистре определяется по младшим *B* битам регистра *Rn* и маскируя оставшиеся биты в Rn, здесь *B* = 8 для длинных слов, *B* = 7 для обычных, *B* = 6 для коротких и *B* = 5 для однобайтных. Таким образом, достигаются полномасштабные правый и левый сдвиги. Величина параметра сдвига для операций с непосредственным параметром использует только младшие *B* битов непосредственного значения, здесь *B* = 7 для

длинных слов, *B* = 6 для обычных, *B* = 5 для коротких и *B* = 4 для однобайтных (см. раздел «Сдвиговое устройство»).

Флаги состояния

SZ Установлен, если результат на сдвиговом устройстве равен нулю;

SN Равен MSB результата; в случае когда LSHIFT сброшен для всех правыхсдвигов на ненулевое значение.

Опции

(NF) Не обновлять флаги (состояния).

Примеры

R5 = LSHIFT R3 BY -4 ;;
Если R3 = 0x140056A3,
то R5 = 0x0140056A.

R5 = LSHIFT R3 BY 4 ;;
Если R3 = 0x140056A3,
то R5 = 0x40056A30.

R5 :4 = LSHIFT R3:2 BY -4 ;;
Если R3 = 0x140056A3 и R2 = 0x87654321,
то R5 = 0x0140056A и R4 = 0x76543210.

R5:4 = LS HIFT R3:2 BY 4 ;;
Если R3 = 0x14 0056A3 и R2 =0x87654321,
то R5 = 0x40056A30 и R4 = 0x76543210.

SR5 = LSHIFT R3 BY -4 ;;
Если R3 = 0x140056A3,
то R5 = 0x0140056A.

SR5 = LSHIFT R3 BY 4 ;;
Если R3 = 0x140056A3,
то R5 = 0x40006A30.

BR5 = LSHIFT R3 BY -4 ;;
Если R3 = 0x140056A3,
то R5 = 0x0100050A.

BR5 = LSHIFT R3 BY 4 ;;
Если R3 = 0x140056A3,
то R5 = 0x40006030.

R5 = ASHIFT R3 BY -4 ;;
Если R3 = 0x840056A3,
то R5 = 0xF840056A.

R5 = ASHIFT R3 BY -4 ;;
Если R3 = 0x140056A3,
то R5 = 0x0140056A.

R5:4 = A SHIFT R3:2 BY -4 ;;
Если R3 = 0x840056A3 и R2 = 0x87654321,
то R5 = 0xF840056A и R4 = 0x F8765432.

R5:4 = A SHIFT R3:2 BY 4 ;;
Если R3 = 0x840056A3 и R2 = 0x87654321,
то R5 = 0x40056A30 и R4 = 0x76543210.

SR5 = ASHIFT R3 BY -4 ;;
Если R3 = 0x840056A3,

то R5 = 0xF840056A.

SR5 = ASHIFT R3 BY 4 ;;
Если R3 = 0x840056A3,
то R5 = 0x40006A30.

BR5 = ASHIFT R3 BY -4 ;;
Если R3 = 0x840056A3,
то R5 = 0xF80005FA.

BR5 = ASHIFT R3 BY 4 ;;
Если R3 = 0x840056A3,
то R5 = 0x40006030.

10.5.2 *Циклический сдвиг*

Синтаксис

{X|Y|XY} *Rs* = ROT *Rm* BY *Rn*|<*Imm6*> {(NF)} ;

{X|Y|XY}{L} *Rsd* = ROT *Rmd* BY Rn|<Imm> {(NF)} ;

Действие

Эта команда циклически сдвигает операнд в регистре Rm на число битов, определяемое операндом в регистре Rn или непосредственным значением в команде. Результат циклического сдвига помещается в регистр Rs. Префикс L указывает на длинный тип операнда, а суффикс d указывает на двойной размер регистра.

Все значения параметра циклического сдвига – числа в дополнительном представлении. Положительные значения дают сдвиг влево, отрицательные значение дают сдвиг вправо. Величина параметра сдвига для операций с параметром в регистре определяются по младшим B битам регистра Rn (оставшиеся биты в Rn маскируются), здесь B = 8 для длинных слов, B = 7 для обычных. Таким образом, достигаются полномасштабные правый и левый циклические сдвиги. Величина параметра сдвига для операций с непосредственным параметром использует только младшие B битов непосредственного значения, здесь B = 7 для длинных слов, B = 6 для обычных (см. раздел «Сдвиговое устройство»).

Флаги состояния

SZ Установлен, если результат равен нулю;
SN Равен MSB результата;

Опции

(NF) Не обновлять флаги (состояния).

Примеры

R5 = ROT R3 BY -4;;
Если R3 = 0x140056A3,
то R5 = 0x3140056A.

R5 = ROT R3 BY 4;;
Если R3 = 0x140056A3,
то R5 = 0x40056A31.

R5:4 = ROT R3:2 BY -4;;
Если R3 = 0x140056A3 *и* R2 = 0x87654321,
то R5 = 0x3140056A *и* R4 = 0x18765432.

R5:4 = ROT R3:2 BY 4;;
Если R3 = 0x140056A3 *и* R2 = 0x87654321,
то R5 = 0x40056A31 *и* R4 = 0x76543218.

LR5:4 = ROT R3:2 BY -4;;

Если R3 = 0x140056A3 и R2 = 0x87654321,
то R5 = 0x1140056A и R4 = 0x38765432.

LR5:4 = ROT R3:2 BY 4;;
Если R3 = 0x140056A3 и R2 = 0x87654321,
то R5 = 0x40056A38 и R4 = 0x76543211.

10.5.3 *Извлечение поля*

Синтаксис

{X|Y|X Y} *Rs* = FEXT *Rm* BY *Rn*|*Rnd* {({SE}{NF})};

{X|Y|XY}L *Rsd* = FEXT *Rmd* BY *Rn*|*Rnd* {({SE}{NF})}

Действие

Эти команды извлекают битовое поле, определённое управляющей информацией в регистре *Rn,* из регистра *Rm* и помещают его в регистр *Rs.*

Существует два варианта этой команды. В первом варианте команда получает управляющую информацию (позиция и длина поля) из регистровой пары *Rnd.* Во втором случае команда получает управляющую информацию из одиночного регистра *Rn* (см. рисунок 177).

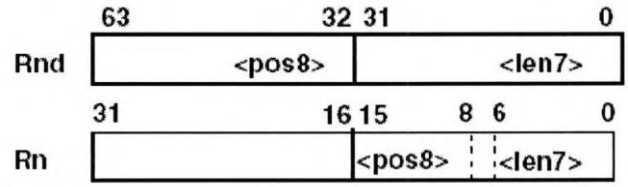


Рисунок 177 – Поля Len7 и Pos8 в двойном и одиночном регистре

Поле длины выровнено по правой границе в регистре *Rn* и имеет длину 7 битов, что позволяет использовать поля длиной от 64 до 0 бит включительно. Поле позиции имеет длину 8 бит, что позволяет выполнять извлечение полей, не укладывающихся в естественную границу слева.

Если включена опция SE, то биты слева от извлечённого поля в регистре-приёмнике *Rs* устанавливаются равными MSB извлечённого поля; если опция не используется – биты слева от извлечённого поля в регистре-приёмнике *Rs* устанавливаются равными нулю (см. рисунок 178).

Префикс L указывает на длинный тип операнда, а суффикс *d* указывает на двойной размер регистра.

Флаги состояния

SZ Установлен, если результат равен нулю;
SN Равен MSB результата.

Опции

(SE) Распространять знак;
(NF) Не обновлять флаги (состояния).

Примеры

Команда FEXT извлекает битовое поле, параметры которого заданы в *Rn*, из *Rm и* помещает его в *Rs*:

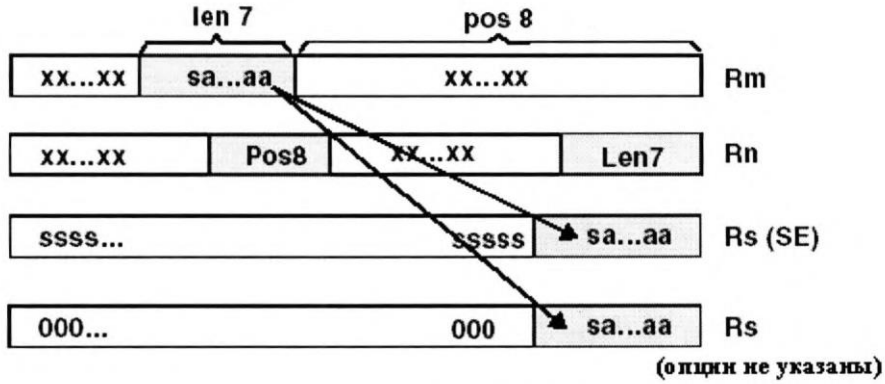


Рисунок 178 – Выполнение команды FEXT с одиночными регистрами

10.5.4 *Размещение поля*

Синтаксис

{X|Y|XY}*Rs* += FDEP *Rm* BY *Rn*|*Rnd* {({SE|ZF} {NF})} ;

{X|Y|XY}L*Rsd* += FDEP *Rmd* BY *Rn*|*Rnd*{({SE|ZF}{NF})} ;

Действие

Эта команда размещает выровненное по правой границе битовое поле из регистра *Rm* в регистре *Rs,* позиция и длина поля в регистре-приёмнике *Rs* определены управляющей информацией в регистре *Rn.*

Если включена опция SE, то биты слева от размещённого поля в регистре-приёмнике *Rs* устанавливаются равными MSB размещённого поля; если опция не используется – биты слева от размещённого поля в регистре-приёмнике *Rs* не изменяются. Если включена опция ZF, то биты слева от размещённого поля в регистре *Rs* устанавливаются равными нулю; если опция не используется – биты слева от размещённого поля в регистре *Rs* не изменяются.

Существует два варианта получения командой управляющей информации (позиция и длина поля):

1 из регистровой пары *Rnd*;

2 из одиночного регистра *Rn*.

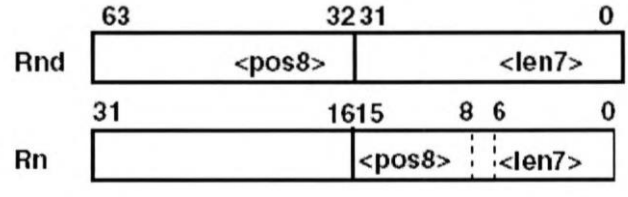


Рисунок 179 – Поля Len7 и Pos8 в двойном и одиночном регистре

Поле длины выровнено по правой границе в регистре *Rn* и имеет длину 7 битов, что позволяет использовать поля, длиной от 64 до 0 бит включительно. Поле позиции имеет длину 8 бит, что позволяет выполнять размещение полей, не укладывающихся в естественную границу слева (рисунок 180).

Префикс L указывает на длинный тип операнда, а суффикс *d* указывает на двойной размер регистра.

Флаги состояния

SZ Установлен, если результат равен нулю;

SN Равен MSB результата.

Опции

(SE) Распространять знак;

(ZF) Заполнить нулями;

(NF) Не обновлять флаги (состояния).

Пример

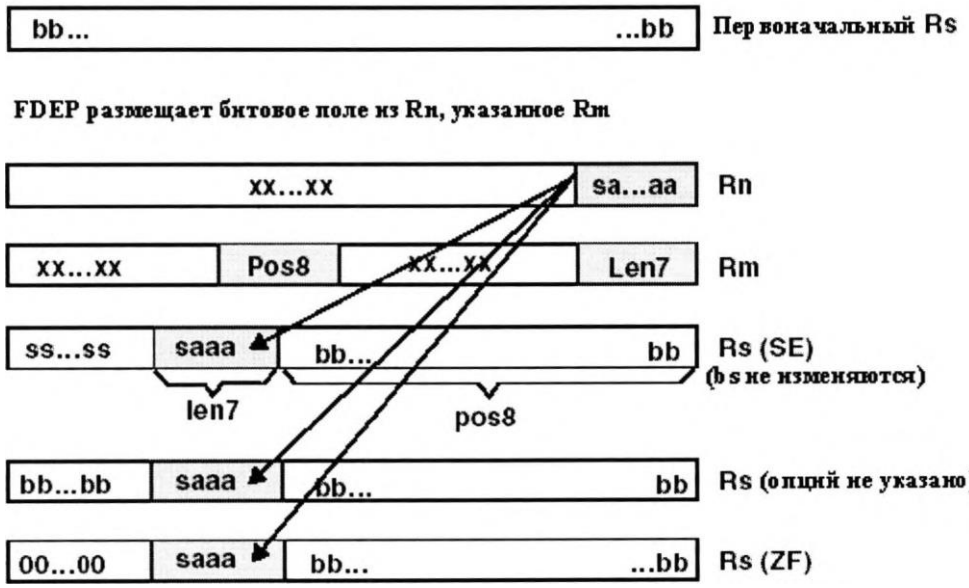


Рисунок 180 – Выполнение команды FDEP с одиночными регистрами

10.5.5 *Поле/Битовая маска*

Синтаксис

{X|Y|XY}*Rs* += MASK *Rm* BY *Rn* {(NF)};

{X|Y|XY}L*Rsd* += MASK *Rmd* BY *Rnd* {(NF)} ;

Действие

Эта команда заменяет некоторые биты в регистре Rs битами регистра Rm, причём замена производится в битах, соответствующих битам, установленным в регистре Rn (рисунок 181). Регистр Rs используется в режиме чтение-модификация-запись. Логически эта операция может быть представлена следующим образом:

Rs = (Rs И-НЕ Rn) ИЛИ (Rm И Rn),

где И, ИЛИ и И-НЕ – логические побитовые операции.

Префикс L указывает на длинный тип операнда, а суффикс d указывает на двойной размер регистра.

Флаги состояния

- SZ Установлен, если результат равен нулю;
- SN Равен MSB результата.

Опции

(NF) Не обновлять флаги (состояния).

Пример

```
Rs += MASK Rm BY Rn;;
```

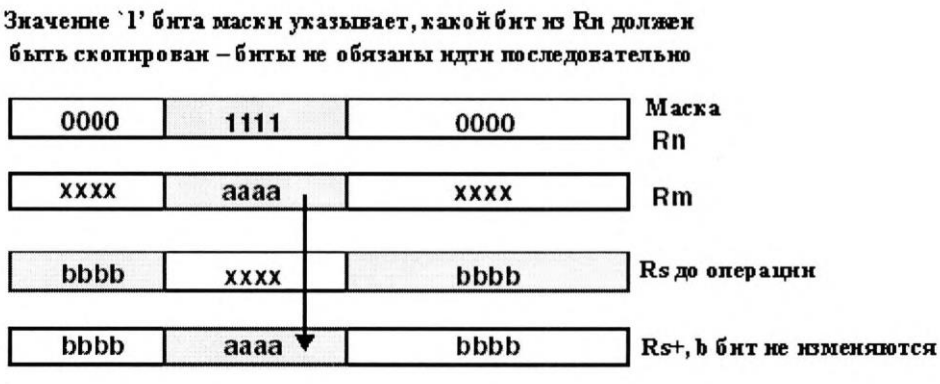


Рисунок 181 – Выполнение команды Поле/Битовая маска

```
XR0 = IMASKH ;; /* загрузить старшую часть IMASK в R0 */
XR1 = 0xFFEF ;; /* загрузить значение маски  в R1 */
XR2 += MASK XR0 BY XR1 ;;
/* использовать MASK для очистки бита 4, остальные биты - не тронуты */
IMASKH = XR2 ;; /* загрузить в IMASKH данные*/
```

10.5.6 *Получить биты*

Синтаксис

{X|Y|XY}*Rsd* = GETBITS *Rmq* BY *Rnd* {({SE}{NF})} ;

Действие

Эта команда извлекает битовое поле из непрерывного битового потока, расположенного в квадро-регистре *Rmq,* и сохраняет извлечённое поле в *Rsd* в соответствии с управляющей информацией в регистре *Rnd.* Эта команда совместно с командами сдвигового устройства PUTBITS и BFOTMP и командой ALU BFOINC используется для реализации однобитного FIFO.

Если включена опция SE, то биты слева от извлечённого поля в регистровой паре приёмника *Rsd* устанавливаются равными MSB извлечённого поля; в противном случае эти биты в регистровой паре приёмника *Rsd* устанавливаются равными нулю.

Управляющая информация (текущий указатель FIFO или BFP и длина извлекаемого поля) содержится в регистровой паре *Rnd.* Команда GETBITS использует BFP и информацию о длине, хранящиеся в *Rnd,* для выполнения операции извлечения битового поля, однако, не обновляет BFP. Обновление BFP должно выполняться с помощью команды BFOINC ALU. Пример, приведенный ниже, содержит рекомендуемую последовательность команд в коде.

Суффикс q обозначает размер регистра (см. подраздел «Регистры регистровых файлов»).

Флаги состояния

SZ Установлен, если извлечённое поле состоит только из нулевых битов;

SN Равен MSB результата.

Опции

(SE) Распространять знак;

(NF) Не обновлять флаги (состояния).

Пример

```
R1:0 = GETBITS R7:4 BY R17:16 ;;
R17 = BFOINC R17:16 ;;
```

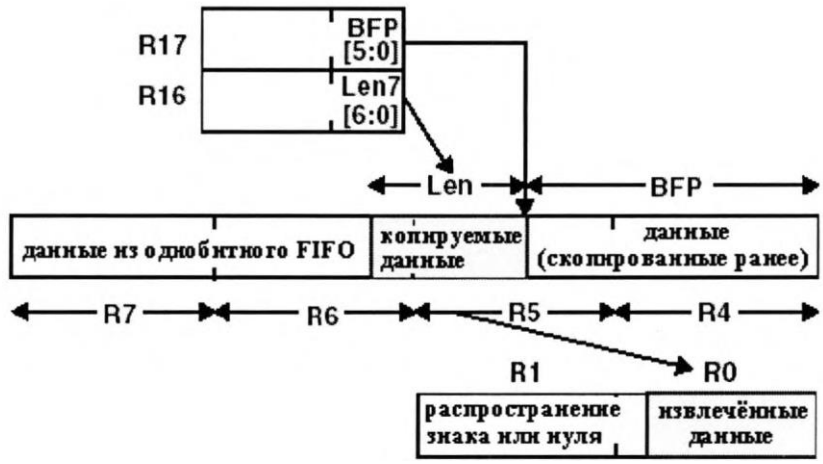


Рисунок 182 – Движение данных при операции получения битов

10.5.7 *Вставить биты*

Синтаксис

{X|Y|XY}*Rsd* += PUTBITS *Rmd* BY *Rnd* {(NF)};

Действие

Эта команда вставляет 64 бита из Rmd в непрерывный битовый поток, расположенный в квадро-регистре, составленном из BFOTMP в старшей части и *Rsd* в младшей. Данные вставляются, начиная с позиции бита, на который указывает поле BFP в регистре *Rnd.* Поле Len7 регистра *Rnd* игнорируется. Эта команда совместно с командами сдвигового устройства GETBITS и BFOTMP и командой ALU BFOINC используется для реализации однобитного FIFO.

Управляющая информация (текущий указатель FIFO или BFP и длина извлекаемого поля) содержится в регистровой паре *Rnd.* Команда PUTBITS использует только поле BFP6, хранящееся в *Rnd,* поле длины в *Rnd* не используется. Команда PUTBITS также не обновляет BFP. Обновление BFP должно выполняться с помощью команды BFOINC ALU.

При выполнении операции вставки битов содержимое регистровой пары *Rmd* полностью помещается квадро-регистр, составленный из BFOTMP и *Rsd.* Как правило, битовое поле, вставляемое в однобитное FIFO, имеет длину, меньшую 64, в этом случае интересующее поле должно быть выровнено по правой границе регистра *Rmd.*

Отметим, что оставшиеся биты слева от интересующего битового поля не играют никакой роли. Когда BFP выйдет за 64-ый бит (событие, которое регистрируется ALU и отражается во флаге AN), содержимое в регистре *Rsd* должно быть выгружено, и на его место должно быть загружено содержимое регистра BFOTMP. Пример на рисунке 182 содержит рекомендуемую последовательность команд в коде, реализующем вставку в однобитное FIFO.

Суффикс d обозначает двойной размер регистра (см. подраздел «Регистры регистровых файлов»).

Флаги состояния

SZ Сброшен;

SN Сброшен.

Опции

(NF) Не обновлять флаги (состояния).

Примеры

На рисунке 183 изображено движение данных при исполнении команды PUTBITS, а в листинге 87 приведён код, соответствующий рисунку.

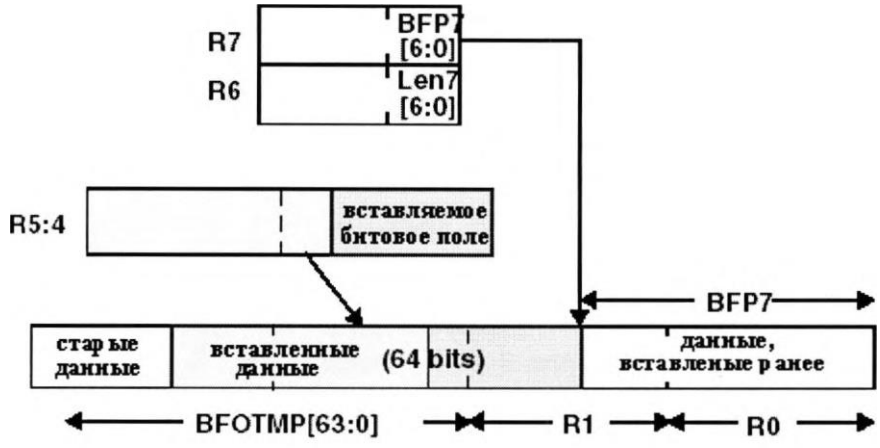


Рисунок 183 – Движение данных для команды: R1:0 += PUTBITS R5:4 BY R7:6;;

Листинг 87 – Пример команды PUTBITS

```
r7 = 0x30;; // вставка начинается с 48-го бита
r6 = 0x20;; // длина поля для обновления указателя однобитного FIFO
bfotmp = r7:6;; // может иметь любое начальное значение
r4 = 0xaaaaaaaa;;
r5 = 0x55555555;;
                // в R5:4 содержится вставляемое битовое поле
r1:0 = r1:0 - r1:0;;
                // для лучшей иллюстрации начальное значение в R1:0 равно 0
r1:0 += putbits r5:4 by r7:6;;
                // R5:4 вставлен в BFOTMP и R1:0, начиная с 48-го бита
                // R1 = 0xaaaa0000; R0 = 0x00000000
                // BFOTMP = 0x00005555 5555aaaa
r7 = bfoinc r7:6;; // указатель битового FIFO обновлён
                /* В данном случае R7 становится равным 0x50*mod(64) = 0x10,
                указатель перешёл за 63-ий бит и флаг AN установлен. */
if xALT; do, 1r3:2 = pass r1:0; r1:0 = bfotmp;;
                /* Если указатель перешёл за 63-ий бит, сохранить значение
                регистра R1:0 в регистре R3:2 и поместить новое значение BFOTMP в
                регистр R1:0, таким образом, BFOTMP может быть использован в
                следующей операции. */
                // r3:2 = 0xaaaa0000 00000000
                // r1:0 = 0x00005555 5555aaaa
```

10.5.8 *Проверить бит*

Синтаксис

{X|Y|XY}BITEST Rm BY Rn|<Imm5> ;

{X|Y|XY}BITEST Rmd BY Rn|<Imm6> ;

Действие

Эта команда проверяет бит #n в регистре *Rm* на равенство нулю; номер бита указан как операнд в регистре *Rn* или как непосредственное значение в команде. Флаг SZ устанавливается, если проверяемый бит равен нулю, и сбрасывается, если он равен единице. Флаг SZ также устанавливается, если номер проверяемого бита превышает 31 для обычных операндов и 63 для длинных операндов. Номер бита

задаётся 6- или 5-битным значением в регистре *Rn* (для длинных или обычных слов соответственно) или непосредственной константой в команде. Например, при операции с обычными словами значение 0x00000000 в *Rn* приведёт к проверке LSB регистра *Rm,* а значение 0x0000001f – к проверке MSB регистра *Rm.* В этой команде суффикс d обозначает двойные регистры.

Флаги состояния

SZ Сброшен, если проверяемый бит равен 1.
Установлен, если проверяемый бит равен 0 или номер проверяемого бита больше 31 (или 63);

SN Равен MSB входного операнда.

10.5.9 *Сбросить/Установить/Переключить бит*

Синтаксис

{X|Y|XY}*Rs* = BCLR|BSET|BTGL *Rm* BY *Rn*|<*Imm5*> {(NF)} ;

{X|Y|XY}*Rsd* = BCLR|BSET|BTGL *Rmd* BY *Rn*|<*Imm6*> {(NF)} ;

Действие

Эти команды сбрасывают (BCLR), устанавливают (BSET) или переключают (BTGL) бит #n в регистре *Rm;* номер бита указан как операнд в регистре *Rn* или как непосредственное значение в команде. Результат операции помещается в регистре Rs. Номер бита задаётся 6- или 5-битным значением в регистре *Rn* (для длинных или обычных слов соответственно) или непосредственной константой в команде. Например, при операции BCLR с обычными словами значение 0x00000000 в *Rn* приведёт к сбросу LSB регистра *Rm,* а значение 0x0000001f – к сбросу MSB регистра *Rm.* Суффикс d обозначает двойной размер регистра.

Флаги состояния

SZ Установлен, если результат равен нулю;

SN Равен MSB результата.

Опции

(NF) Не обновлять флаги (состояния).

Примеры

R5 = BCLR R3 BY 5;
Если R3 = 0x140056A3,
то R5 = 0x14005683.

R5:4 = BCLR R3:2 BY 5;
Если R3 = 0x140056A3, *и* R2 = 0x87654321,
то R5 = 0x14005683 *и* R4 = 0x87654301.

R5 = BSET R3 BY 6;
Если R3 = 0x140056A3,
то R5 = 0x140056E3.

R5:4 = BSET R3:2 BY 6;
Если R3 = 0x140056A3, *и* R2 = 0x87654321,
то R5 = 0x140056A3 и R4 = 0x87654361.

R5 = BTGL R3 BY 6;
Если R3 = 0x140056A3,
то R5 = 0x140056E3.

R5:4 = BTGL R3:2 BY 6;
Если R3 = 0x140056A3, *и* R2 = 0x87654321,
то R5 = 0x140056A3 *и* R4 = 0x87654361.

10.5.10 *Выдать число ведущих нулей/единиц*

Синтаксис

{X|Y|XY}*Rs* = LD0|LD1 *Rm*|*Rmd* {(NF)} ;

Действие

Эта команда вычисляет количество ведущих битовых нулей (LD0) или единиц (LD1) в операнде в регистре *Rm* Полученное число помещается в шесть младших битов *Rs.*

Флаги состояния

SZ Установлен, если результат равен нулю. Этот случай реализуется, когда MSB в регистре *Rm* равен 1 для команды LD0 или MSB в регистре *Rm* равен 0 для команды LD1.

SN Сброшен

Опции

(NF) Не обновлять флаги (состояния)

Примеры

R5 = LD0 R3;
Если R3 = 0x140056A3,
то R5 = 0x3.

R5 = LD1 R3:2;
Если R3 = 0xE40056A3 *и* R2 = 0x87654321,
то R5 = 3.

10.5.11 *Вычислить порядок*

Синтаксис

{X|Y|XY}*Rs* = EXP *Rm*|*Rmd* {(NF)} ;

Действие

Эта команда вычисляет порядок операнда в регистре *Rm.* Если количество начальных бит операнда в *Rm,* совпадающих со знаковым, равно n, тогда результатом этой операции будет значение (n-1) в *Rs.*

Флаги состояния

SZ Установлен, если извлечённый порядок равен нулю;

SN Установлен, если операнд с фиксированной точкой в регистре *Rm* отрицателен (MSB равен единице).
Флаг SN используется для получения знака входного операнда. Это необходимо при реализации плавающей арифметики не по стандарту IEEE, а также при программной реализации плавающей арифметики двойной точности.

Опции

(NF) Не обновлять флаги (состояния).

Примеры

R5 = EXP R3;
Если R3 = oxE4000000,
то R5 = -2.
Если R3 = 0x00000101,
то R5 = -22.

R5 = EXP R3:2;
Если R3 = 0x140056A3 и R2 = 0x87654321,
то R5 = -2 (в шестнадцатеричной записи, R5 = 0xFFFFFFFE).

10.5.12 *Блочная плавающая арифметика*

Синтаксис

{X|Y|XY}BKFPT *Rmd, Rnd* ;

Действие

Эта команда используется для определения масштабирующего множителя при работе с 16-битной блочной плавающей арифметикой. В двух регистрах источника Rmd и Rnd, содержатся восемь коротких слов. После выполнения этой команды два флага блочной плавающей арифметики BF1:0 в регистре X/YSTAT устанавливаются равными наибольшему значению из (a) числа избыточных знаковых битов в восьми входных коротких словах и (b) предыдущего значения BF1:0.

Команда BKFPT сопоставляет каждому из восьми входных коротких слов двухбитное значение, равное трём минус количество знаковых битов. Это значение зависит от трёх MSB входного короткого слова (остальные знаковые биты, если есть, игнорируются):

- если три MSB операнда равны 000 или 111, то значение 00;
- если три MSB операнда равны 001 или 110, то значение 01;
- если два MSB операнда равны 01 или 10, то значение 10.

После вычисления этих двух битов для каждого короткого слова конечный результат определяется как максимум из вычисленных восьми двухбитных значений и текущего значения в BF1:0. И наконец, результат заносится в BF1:0. Эта команда регистрирует до двух избыточных битов знака.

Например, если BF1:0 = b#00 и три MSB всех восьми входных чисел равны либо b#000, либо b#111, то BF1:0 = b#00.

Если три MSB хотя бы одного из восьми входных чисел равны b#001 или b#110, то BF1:0 = b#01.

Если три MSB хотя бы одного из восьми входных чисел равны b#011 или b#100, то BF1:0 = b#10.

Флаги состояния

Команда изменяет флаги BF0 и BF1. Остальные флаги состояния не изменяются.

Примеры

Предположим, что изначально BF1:0 = b#00:

XR0 = b#0010 0000 ... 0000

XR1 = 0

После выполнения команды XBKFPT R1:0,R1:0;; флаги будут установлены следующим образом: BF1:0 = b#01.

XR0 = 0

XR1 = 0

С этими входными данными повторное выполнение команды XBKFPT R1:0,R1:0;; приведёт к следующему результату: BF1:0 = b#01.

10.5.13 *Загрузка/Пересылка регистра временного однобитного FIFO переполнения (BFOTMP)*

Синтаксис

{X|Y|XY}*Rsd* = BFOTMP {(NF)} ;

{X|Y|XY}BFOTMP = *Rmd* {(NF)} ;

Действие

Эти команды загружают значение из регистра BFOTMP в регистре *Rsd* или загружают операнд из регистра *Rmd* в регистр BFOTMP. Регистр BFOTMP является внутренним регистром сдвигового устройства. Эта команда используется для временного сохранения битов переполнения после выполнения команды PUTBITS.

Флаги состояния

Операция не влияет на флаги состояния.

Опции

(NF) Не обновлять флаги (состояния).

Примеры

R9:8 = BFOTMP;;
Если BFOTMP = 0x17000016,
то R9: 8 = 0x17000016.

BFOTMP = R9;8;;
Если R9:8 = 0x40000005,
то BFOTMP = 0x40000005.

10.5.14 *Загрузка/Пересылка статусных регистров вычислительных блоков (X/YSTAT)*

Синтаксис

{X|Y}STAT = *Rm* ;

{X|Y}STAT_L = *Rm* ;

{X|Y}Rs = {X|Y)STAT ;

Действие

Эти команды загружают операнд из регистра *Rmd* в регистр статусный регистр вычислительного блока X/Y или сохраняют значение статусного регистра вычислительного блока X/Y в регистре *Rs.* Имеется два вида команд загрузки регистра X/YSTAT. Команды одного вида загружают в регистр X/YSTAT значение регистра *Rm* целиком, тогда как команды второго вида загружают только 15 LSB операнда в регистре *Rm* в X/YSTAT. Второй вид команд используется, когда необходимо восстановить значения только динамических флагов состояния, не изменяя залипающие флаги и биты режима.

Флаги состояния

После команды X/YSTAT = *Rm* значение из регистра *Rm* записывается в статусный регистр (X/YSTAT), и флаги состояния выставляются в соответствии с записанным значением регистра *Rm.*

Команда *Rm* = X/YSTAT не влияет на флаги состояния.

Примеры

XSTAT_L = R6;;
Если R 6 = 0x910,
то XSTAT = 0x910.

Если R6 = 0x90010,
то XSTAT = 0x10.

XR6 = STAT;;
Если XSTAT = 0x10,
то R6 = 0x10.

10.5.15 *Загрузка регистров CLU результата (TR) и истории (TRH) решётчатых операций, а также коммуникационного управляющего регистра (CMCTL)*

Синтаксис

{X|Y|XY}T*Rs* = *Rm* ; /* загрузка TRx */

{X|Y|XY}T*Rsd* = *Rmd ;*

{X|Y|XY}T*Rsq* = *Rmq* ;

{X|Y|XY}TH*Rs* = *Rm* ; /* загрузка THRx */

{X|Y|XY}TH*Rsd* = *Rmd* { (I) } ;

{X|Y|XY}TH*Rsq* = *Rmq* ;

{X|Y|XY}CMCTL = *Rm* ; /* CMCTL load */

Действие

Данные из регистра-источника (справа от знака =) пересылаются в регистр-приёмник (слева от знака =). Данные могут быть обычными (32 бита), двойными (64 бита) или квадро-словами (128 бит). Номера регистров должныбыть выровнены в соответствии с размером данных.

Данные из регистра Rm или Rmd пересылаются на стадии конвейера EX2.

При исполнении команды THRs = Rm; или THRsd = Rmd { (I) }; параллельно с командами, сдвигающими содержимое регистра THR (например, команды ACS, DESPREAD и другие), команда загрузки THR имеет больший приоритет, чем команда сдвига THR.

Флаги состояния

Операция не влияет не флаги состояния.

Опции

Использование опции Чередовать (I) допустимо только в командах с двойными или квадрорегистрами. Команда с этой опцией чередует биты младшего слова *Rmd* с битами старшего слова *Rmd* и загружает полученный результат в регистр THR. Рисунок 184 иллюстрирует такое чередование для двойных регистров.

Рисунок 184 – Опция Чередовать для двойных регистров

При использовании опции (I) в квадрозагрузке THR (TH*Rq=Rmq* (I) ;;) биты двух младших длинных слов (слова 1:0) чередуются с битами старших двух длинных слов (слова 3:2). Рисунок 185 иллюстрирует такое чередование для квадрорегистров.

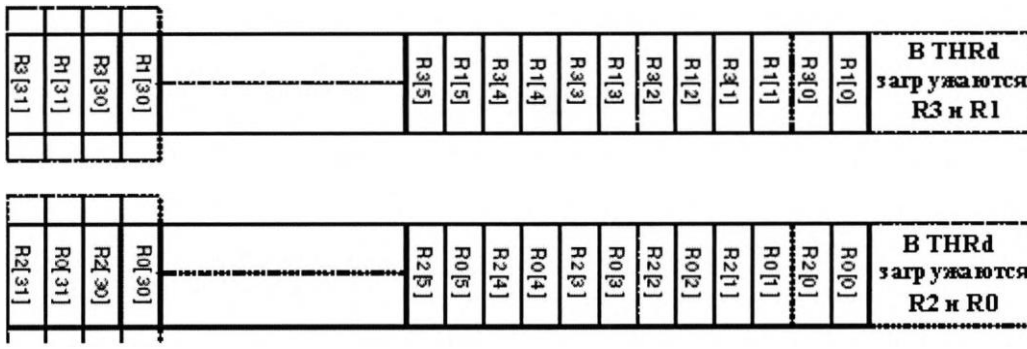


Рисунок 185 – Опция Чередовать для квадрорегистров

Если входные и выходные данные команды [с опцией (I)] – квадрослова (обозначены in/out[127:0] соответственно), то команда выполняется следующим образом:

Out [127:0] == {in [127], in [63], in[126], in[62], in[125], in[61] in[65], in[1], in[64], in[0]}

Примеры

```
XYTR8 = R4 ;;
XYTR1:0 = R3:2 ;;
XYTR11:8 = R7:4 ;;
XYTHR3 = R4 ;;
XYTHR1:0 = R3:2 (I) ;;
XYTHR3:0 = R7:4 ;;
XYCMCTL = R4 ;;
```

10.6 Команды (целочисленные) IALU

Два независимых целочисленных арифметико-логических устройства (IALU) процессора называются J-IALU и K-IALU. IALU может выполнять как обычные операции ALU, так и операции адресации данных.

Целочисленными операциями ALU являются:

- сложение и вычитание с использованием или без использования бита переноса (займа);
- арифметический сдвиг вправо, логический сдвиг вправо и циклические сдвиги;
- логические операции: AND, AND NOT, NOT, OR и XOR;
- функции: абсолютное значение, минимум, максимум, сравнение.

Описание операций IALU, флагов состояния IALU, IALU-предикатов, а также конкретные примеры приведены в разделе «Целочисленное АЛУ (IALU)». В IALU реализованы также команды, обеспечивающие поддержку адресации данных. Справочная информация по командам IALU адресации данных приведена в пункте «Операции IALU загрузки, сохранения и пересылки».

Соглашения по обозначениям имён регистров, необязательных элементов или вариантов, используемых в справках на команды, подробно освещены в разделе «Целочисленное АЛУ (IALU)». Вкратце эти соглашения таковы:

- { } – необязательные элементы указаны в фигурных скобках; фигурные скобки не являются частью синтаксиса команды;
- | – вертикальная черта разделяет варианты значений элементов; вертикальные черты не являются частью синтаксиса команды;
- *Jm* или *Km* – буква J или K в именах регистров, выделенных курсивом, указывает на регистр J-IALU или K-IALU;
- *Jm* – имена регистров, выделенные курсивом, означают шаблоны, вместо

которых могут быть подставлены имена одиночных (шаблоны *Jm, Jn, Js, Rs, Ureg_s),* двойных (шаблоны *Rsd, Rmd, Ureg_sd, Ureg_md)* или квадро- (шаблоны *Rsq, Rmq, Ureg sq, Ureg mq)* регистров.

Каждая команда, по которой дана справка, занимает одно поле команды в командной строке. Более подробную информацию о командных строках и ограничениях по комбинированию команд можно найти в разделах «Синтаксис и структура командной строки» и «Правила запараллеливания команд».

10.6.1 *Сложение/Вычитание (целочисленное)*

Синтаксис

Js = Jm +|- *Jn*|<*Imm8*>|<*Imm32*> {({CJMP|CB|BR}{NF})} ;
JB0|JB1|JB2|JB3|JL0|JL1|JL2|JL3 = *Jm* +|-*Jn*|<*Imm8*>|<*Imm32*> {(NF)},

Ks = Km +|- *Kn*|<*Imm8*>|<*Imm32*> {({CJMP|CB|BR}{NF})} ;
KB0|KB1|KB2|KB3|KL0|KL1|KL2|KL3 = *Km* +|-*Kn*|<*Imm8*>|<*Imm32*> {(NF)},

Действие

Эти команды складывают или вычитают операнды в регистрах Jm/Km и Jn/Kn. Результат помещается в регистр Js/Ks. Операция использует в качестве операндов либо значения в двух регистрах, либо значение в одном регистре и непосредственное значение. Константа Imm может быть 8- или 32-битной (в последнем случае будет использовано непосредственное расширение (см. пункт «Операции непосредственного расширения»). Буква J отмечает регистры J-IALU; буква K – регистры K-IALU.

При использовании опции CJMP результат операции помещается не только в Js/Ks, но и в регистр CJMP. Опция CB может быть использована только для регистров Jm/Km с номерами 0, 1, 2 или 3 и указывает, что операция должна быть проведена для кольцевого буфера. Опция BR указывает, что должен использовать сумматор с обратным переносом. Никакие две из упомянутых опций не могут использоваться одновременно.

Другой вариант формата команды Js = Jm + Jn – это JBi/JLi = Jm + Jn. Эти команды действуют одинаково, однако результат последней операции сохраняется в регистрах кольцевого буфера JBi или JLi, а не в регистре Js.

Если выбрана опция CB или BR, переполнения быть не может. Флаги состояния:

JZ/KZ Установлен, если все биты результата равны 0;

JN/KN Равен MSB результата;

JV/KV Установлен, если имеет место переполнение; иначе – сброшен;

JC/KC Установлен, если был перенос (заём) из старшего разряда; при использовании опции CB или BR сброшен.

Опции

CJMP Вычисляемый адрес перехода – результат операции загружается в регистр CJMP;

CB Кольцевой буфер – провести операцию как для кольцевого буфера; может быть использована только для *Jm/Km* = J/K [3:0];

BR Обратный перенос – использовать для бит-реверсивной адресации;

(NF) Не обновлять флаги (состояния).

Примеры

```
J3 = J5 + J29;;
K2 = K2 + 0x25;;
K6 = K1 + K8 (CB);;
```

```
J1 = J5 + J10 (BR);;
JB0 = J31 + 0x5;;
J4 = J2 + J3 (CJMP);;
J3 = J5 - J29;;
K2 = K2 - 0x25;;
K6 = K1 - K8 (CB);;
J1 = J5 - J10 (BR);;
JB0 = J31 - 0x5;;
J4 = J2 - J3 (CJMP);;
```

10.6.2 *Сложение/Вычитание с переносом/займом (целочисленное)*

Синтаксис

```
Js = Jm+Jn|<Imm8>|<Imm32>+ JC{(NF)} ;
Js = Jm- Jn|<Imm8>|<Imm32>+ JC - 1{(NF)} ;
Ks = Km+Kn |<Imm8>|<Imm32>+KC{(NF)} ;
Ks = Km- Kn|<Imm8>|<Imm32>+KC - 1{(NF)} ;
```

Действие

Эти команды складывают операнды в регистрах *Jm/Km* и *Jn/Kn,* добавляя бит переноса, или вычитают их друг из друга, вычитая также бит займа, в обоих случаях при этом используется флаг переноса J-/K-IALU из регистра J/KSTAT. Результат помещается в регистр Js/Ks. Операция использует в качестве операндов либо значения в двух регистрах, либо значение в одном регистре и непосредственное значение. Константа Imm может быть 8- или 32-битной (в последнем случае используется непосредственное расширение (см. пункт «Операции непосредственного расширения»). Буква J отмечает регистры J-IALU; буква K – регистры K-IALU.

Флаги состояния

JZ/KZ Установлен, если все биты результата равны 0;

JN/KN Равен MSB результата;

JV/KV Установлен, если имеет место переполнение;

JC/KC Установлен, если был перенос (заём) из старшего разряда;

Опции

(NF) Не обновлять флаги (состояния)

Примеры

```
J4 = J2 + J8 + JC;;
K8 = K1 + K4 + KC;;
K3 = K1 + 0x2 + KC;;
J4 = J2 - J8 + JC-1;;
K8 = K1 - K4 + KC-1;;
K3 = K1 - 0x2 + KC-1;;
```

10.6.3 *Сложение/Вычитание с делением пополам (целочисленное)*

Синтаксис

```
Js = (Jm +|- Jn |<Imm8>|<Imm32>) /2 {(NF)} ;
Ks = (Km +|- Kn |<Imm8>|<Imm32>) /2 {(NF)} ;
```

Действие

Эти команды складывают или вычитают операнды в регистрах Jm/Km и Jn/Kn, а затем делят результат пополам. Конечный результат помещается в регистр Js/Ks. Операция использует в качестве операндов либо значения в двух регистрах, либо значение в одном регистре и непосредственное значение. Константа Imm может быть 8- или 32-битной (в последнем случае используется непосредственное расширение, см. пункт «Операции непосредственного расширения»).

В качестве операции деления эти команды используют арифметический сдвиг вправо. Таким образом, происходит округление к минус бесконечности, а не к нулю: если результат отрицательный, то он будет округлён к меньшему отрицательному числу.

Например, результатом операции (-1-2)/2 будет -2, операции (-1+0)/2 будет -1, в то время как результатом операции (1+2)/2 будет 1, операции (1 +0)/2 будет 0.

Буква J отмечает регистры J-IALU; буква K – регистры K-IALU.

Флаги состояния

JZ/KZ Установлен, если все биты результата равны 0;
JN/KN Равен MSB результата;
JV/KV Сброшен;
JC/KC Сброшен.

Опции

(NF) Не обновлять флаги (состояния).

Примеры

```
J4 = (J2 + J8) / 2;;
K9 = (K2 + 0x2) / 2 ;;
J4 = (J2 - J8) / 2;;
K9 = (K2 - 0x2) / 2 ;;
```

10.6.4 *Сравнение (целочисленное)*

Синтаксис

COMP(Jm, Jn|<Imm8>|<Imm32>) {(U)} ;

COMP(Jm, Jn|<Imm8>|<Imm32>) { (U) } ;

Действие

Эта команда сравнивает операнд в регистре *Jm/Km* с операндом в регистре *Jn/Kn.* Команда устанавливает флаг JZ/KZ, если операнды равны, и флаг JN/KN, если операнд в регистре *Jm/Km* меньше операнда в регистре *Jn/Kn.* Буква J отмечает регистры J-IALU; буква K – регистры K-IALU.

Операция использует в качестве операндов либо значения в двух регистрах, либо значение в одном регистре и непосредственное значение. Константа *Imm* может быть 8- или 32- битной (в последнем случае используется непосредственное расширение, см. пункт «Операции непосредственного расширения»). Опция Беззнаковое указывает, что сравниваемые числа должны интерпретироваться как беззнаковые.

Флаги состояния

JZ/KZ Установлен, если входные операнды равны;
JN/KN Установлен, если операнд в *Jm/Km* меньше операнда в *Jn/Kn;*
JV/KV Сброшен;
JC/KC Сброшен.

Опции

( ) Знаковое;
(U) Беззнаковое сравнение.

Примеры

COMP(J4,J8) (U);;
Если J4 = 0xFFFFFFFC и J8 = 0x00000003,
то флаги состояния устанавливаются следующим образом:
JZ = 0
JN = 0

COMP(J4,J8) ;;
Если J4 = 0xFFFFFFF C и J8 = 0x00000003,
то флаги состояния устанавливаются следующим образом:
JZ = 0
JN = 1

COMP(J4,0x00000003) (U);;
Если J4 = 0xFFFFFFF C,
то флаги состояния устанавливаются следующим образом:
JZ = 0
JN = 0

10.6.5 *Максимум/Минимум (целочисленный)*

Синтаксис

Js = MAX|MIN (Jm, Jn |<mm8>|<Imm32>) {(NF)} ;

Ks = MAX|MIN (Km, Kn |<Imm8>|<Imm32>) {(NF)} ;

Действие

Эти команды вычисляют максимум (больший) или минимум (меньший) из двух операндов в регистрах *Jm/Km* и *Jn/Kn.* Результат помещается в регистр *Js/Ks.* Операция MAX всегда знаковая. Операция использует в качестве операндов либо значения в двух регистрах, либо значение в одном регистре и непосредственное значение. Константа *Imm* может быть 8- или 32-битной (в последнем случае используется непосредственное расширение, см. пункт «Операции непосредственного расширения»). Буква J отмечает регистры J-IALU; буква K – регистры K-IALU.

Флаги состояния

JZ/KZ Установлен, если все биты результата равны 0;
JN/KN Равен MSB результата;
JV/KV Сброшен;
JC/KC Сброшен.

Опции

(NF) Не обновлять флаги (состояния).

Примеры

К4 = MAX(К2,К 9) ;;
Если К2 = 5 и К9 = 1,
то К4 = 5.

К4 = MAX (К2, 0x1) ;;
Если К2 = 5,
то К4 = 5.

J2 = MIN(J7,J8) ;;
Если J7 = 4 и J8 = 3,
то J2 = 3.

J2 = MIN(J7,0x3) ;;
Если J7 = 4,

то J2 = 3.

10.6.6 *Абсолютное значение (целочисленное)*

Синтаксис

Js = ABS *Jm* {(NF)} ;

Ks= ABS *Km* {(NF)} ;

Действие

Эта команда вычисляет абсолютное значение операнда в регистре *Jn/Kn.* Результат помещается в регистр *Js/Ks.* Буква K отмечает регистры K-IALU, а J - регистры J-IALU.

Операция ABS на минимальном отрицательном числе (0x80000000) возвращает максимальное положительное число (0x7FFFFFFF) и устанавливает флаг переполнения.

Флаги состояния

JZ/KZ Установлен, если все биты результата равны 0;
JN/KN Равен MSB входного операнда;
JV/KV Установлен, если на входе минимальное отрицательное число;
JC/KC Сброшен.

Опции

(NF) Не обновлять флаги (состояния)

Примеры

J5 = ABS J4 ;;
Если J4 = 0x80000000,
то J5 = 0x7FFFFFFF *и оба флага* JV *и* JN *установлены*.
Если J4 = 0x7FFFFFFF,
то J5 = 0x7FFFFFFF.
Если J4 = 0xF0000000,
то J5 = 0x10000000 *и флаг* JN *установлен*.

10.6.7 *Логическое И/И-НЕ/ИЛИ/ИСКЛ-ИЛИ/НЕ (целочисленное)*

Синтаксис

Js = *Jm* OR|AND|XOR|AND NOT *Jn*|<*Imm8*>|<*Imm32*> {(NF)} ;

Js = NOT *Jm* {(NF)} ;

Ks = *Km* OR|AND|XOR|AND NOT *Kn* |<*Imm8*>|<*Imm32*> {(NF)} ;

Ks = NOT *Km* {(NF)} ;

Действие

Эти команды проводят побитовые логические операции И(AND)/И-НЕ(AND NOT)/ИЛИ(OR) или ИСКЛ-ИЛИ(XOR) над операндами в регистрах *Jm/Km* и *Jn/Kn.* Команда NOT проводит операцию логического дополнения (НЕ) над операндом в регистре *Jm/Km.* Результат помещается в регистр *Js/Ks.* Операция использует в качестве операндов либо значения в двух регистрах, либо значение в одном регистре и непосредственное значение. Константа *Imm* может быть 8- или 32-битной, в последнем случае используется непосредственное расширение (см. пункт «Операции непосредственного расширения»). Буква J отмечает регистры J-IALU; буква K – регистры K-IALU.

Флаги состояния

JZ/KZ Установлен, если все биты результата равны 0;
JN/KN Равен MSB результата;

JV/KV Сброшен;
JC/KC Сброшен.

Опции

(NF) Не обновлять флаги (состояния).

Примеры

J5 = J4 AND J8 ;;
Если J4 = b#... 1001 и J8 = b#...1100,
то J5 = b# ... 1000.

K5 = K4 AND 0xC ;;
Если K4 = b#1001 (в двоичной записи),
то K5 = b#1000 (в двоичной записи).

J6 = J2 AND NOT J5 ;;
Если J2 = b#...1001 и J5 = b#...1100,
то J6 = b#...0001.

J6 = J2 AND NOT 0xC ;;
Если J2 = b#...1001 (в двоичной записи),
то J6 = b#... 0001 (в двоичной записи).

J5 = J3 OR J4 ;;
Если J3 = b#...1001 и J4 = b#...1100,
то J5 = b#...1101.

K5 = K3 OR 0xC ;;
Если K3 = b#...1001 (в двоичной записи),
то K5 = b#...1101 (в двоичной записи).

J3 = J2 XOR J7 ;;
Если J2 = b#...1001 и J7 = b#...1100,
то J3 = b#. . . 0101.

J3 = J2 OR 0xC ;;
Если J2 = b#... 1001 (в двоичной записи),
то J3 = b# ... 0101 (в двоичной записи).

J7 = NOT J6 ;;
Если J6 = b#...0110,
то J7 = b#...1001.

10.6.8 *Арифметический сдвиг/Логический сдвиг (целочисленный)*

Синтаксис

Js = ASHIFTR|LSHIFTR *Jm* {(NF)};

Ks = ASHIFTR|LSHIFTR *Km* {(NF)} ;

Действие

Эти команды проводят арифметический сдвиг (с распространением знака) или логический сдвиг (без распространения знака) операнда в регистре *Jm/Km* вправо на один бит. Результат сдвига помещается в регистр *Js/Ks.* Сдвигаемые значения – числа в дополнительном представлении. Буква J отмечает регистры J-IALU; буква K – регистры K-IALU.

Флаги состояния

JZ/KZ Установлен, если все биты результата равны 0;
JN/KN Равен MSB результата;
JV/KV Сброшен;
JC/KC Равен LSB входного операнда.

Опции

(NF) Не обновлять флаги (состояния).

Примеры

```
K4 = ASHIFTR K3 ;;
Если K3 = 0x86000000,
то K4 = 0xC3000000 и флаг JN установлен.
Если K3 = 0x08600000,
то K4 = 0x04300000 и флаг JN сброшен.
Если K3 = 0xFFFFFFFF,
то K4 = 0xFFFFFFFF и оба флага JN и JC установлены.

K4 = LSHIFTR K3 ;;
Если K3 = 0x08000000,
то K4 = 0x04000000.
Если K3 = 0x86000000,
то K4 = 0x43000000.
Если K3 = 0xFFFFFFFF,
то K4 = 0x7 FFFFFFF и флаг JC установлен.
```

10.6.9 *Циклический сдвиг влево/ Циклический сдвиг вправо (целочисленный)*

Синтаксис

Js = ROTR|ROTL *Jm* {(NF)} ;

Ks = ROTR|ROTL *Km* {(NF)} ;

Действие

Эти команды циклически сдвигают операнд в регистре *Jm/Km* влево или вправо. Результат циклического сдвига помещается в регистр *Js/Ks.* Буква J отмечает регистры J-IALU; буква K – регистры K-IALU.

Флаги состояния

JZ/KZ	Установлен, если все биты результата равны 0;
JN/KN	Равен MSB результата;
JV/KV	Сброшен;
JC/KC	Сброен.

Опции.

(NF) Не обновлять флаги (состояния).

Примеры

```
J5 = ROTR J3 ;;
Если J3 = b#1000…0101,
то J5 = b#1100…010.

J5 = ROTL J3 ;;
Если J3 = b#1000…0101,
то J5 = b#000…01011.
```

10.6.10 *Дополнительные арифметические команды IALU*

10.6.10.1 Логический сдвиг влево на заданное число бит

Синтаксис

Js = LSHIFTL Jm by <Imm5> {(NF)};

Ks = LSHIFTL Km by <Imm5> {(NF)};

Код команды

31	30	29	28	27:26	25:21	20:16	15:14	13	12:8	7:6	5:1	0
EX	CC/NF	0	JK	11	11111	Js/Ks	01	0	#k	00	Jn/Kn	0

Действие

Команда выполняет логический сдвиг влево 32-разрядного значения регистра Jm на константное 5-разрядное значение, задаваемое в поле команды. При выполнении сдвига в освобождаемые справа разряды заносится ноль. Результат сдвига помещается в регистр *Js/Ks.* Буква J отмечает регистры J-IALU; буква K – регистры K-IALU.

Флаги состояния

JZ/KZ Установлен, если все биты результата равны 0;
JN/KN Равен MSB результата;
JV/KV Сброшен;
JC/KC Сброшен.

Опции

(NF) Не обновлять флаги (состояния).

10.6.10.2 Арифметический сдвиг вправо на заданное число бит

Синтаксис

Js = ASHIFTR Jm by <Imm5> {(NF)};

Ks = ASHIFTR Km by <Imm5> {(NF)};

Код команды

31	30	29	28	27:26	25:21	20:16	15:14	13	12:8	7:6	5:1	0
EX	CC/NF	0	JK	11	11111	Js/Ks	10	0	#k	00	Jn/Kn	0

Действие

Команда выполняет арифметический сдвиг вправо 32-разрядного значения регистра Jm на константное 5-разрядное значение, задаваемое в поле команды. Результат сдвига помещается в регистр *Js/Ks.* Значение константы лежит от 0 до 31. При выполнении сдвига в освобождаемые слева разряды заносится 31-й знаковый разряд. Буква J отмечает регистры J-IALU; буква K – регистры K-IALU.

Флаги состояния

JZ/KZ Установлен, если все биты результата равны 0;
JN/KN Равен MSB результата;
JV/KV Сброшен;
JC/KC Сброшен.

Опции

(NF) Не обновлять флаги (состояния).

10.6.10.3 Логический сдвиг вправо на заданное число бит

Синтаксис

Js = LSHIFTR Jm by <Imm5> {(NF)};

Ks = LSHIFTR Km by <Imm5> {(NF)};

Код команды

31	30	29	28	27:26	25:21	20:16	15:14	13	12:8	7:6	5:1	0
EX	CC/NF	0	JK	11	11111	Js/Ks	11	0	#k	00	Jn/Kn	0

Действие

Команда выполняет логический сдвиг вправо 32-разрядного значения регистра Jm на константное 5-разрядное значение, задаваемое в поле команды. Результат сдвига помещается в регистр *Js/Ks.* Значение константы лежит от 0 до 31. При выполнении сдвига в освобождаемые слева разряды заносится ноль. Буква J отмечает регистры J-IALU; буква K – регистры K-IALU.

Флаги состояния

JZ/KZ Установлен, если все биты результата равны 0;
JN/KN Равен MSB результата;
JV/KV Сброшен;
JC/KC Сброшен.

Опции

(NF) Не обновлять флаги (состояния).

10.6.10.4 Реверс бит регистра

Синтаксис

Js = BITREV Jm {(NF)};

Код команды

31	30	29	28	27:26	25:21	20:16	15:14	13	12:8	7:6	5:1	0
EX	CC/NF	0	JK	11	00110	Js/Ks	01	0	Jm/Km	00	11111	0

Действие

Команда выполняет реверс (перестановку в обратном порядке) бит регистра *Jm* и запись результата в регистр *Js*.

Флаги состояния

JZ/KZ Установлен, если все биты результата равны 0;
JN/KN Равен MSB результата;
JV/KV Сброшен;
JC/KC Сброшен.

Опции

(NF) Не обновлять флаги (состояния).

10.6.10.5 Атомарная операция RMW

Синтаксис

atomic; <операция J-АЛУ>; <операция XY-АЛУ>; <операция K-АЛУ >;;

Код команды

Префикс *atomic* имеет значение 0x33C0001F

Действие

Префикс обеспечивает атомарное выполнение последовательности инструкций в линии инструкций. Такая последовательность не может быть прервана прерыванием. Существуют строгие ограничения на порядок записи и вычислительные модули, в которых выполняются инструкции такой линии:

1. Первой инструкцией в линии должен быть префикс *atomic*.
2. Вторая инструкция должна быть инструкцией J АЛУ. Например, загрузкой значения из памяти.
3. Третья инструкция должна быть инструкцией X или Y АЛУ.
4. Четвертая инструкция должна быть инструкцией K АЛУ. Например, запись значения из регистра в память.
5. Линия инструкций с префиксом *atomic* не может быть условной

Линия команд с префиксом *atomic* выполняет следующие действия:

- ожидает на стадии дешифрации, пока все команды конвейера закончат выполнение;
- передает выполнение команды в модуле J со стадии дешифрации на стадию исполнения, но при этом удерживает команды в остальных модулях на стадии дешифрации;
- передает выполнение команды в модулях ХУ со стадии дешифрации на стадию исполнения, но при этом удерживает команды в остальных модулях на стадии дешифрации;
- разрешает обычное выполнение оставшихся команд.

Примеры

В качестве примера возьмем следующую линию инструкций:

```
atomic; xr0 = [j0+0]; xr2 = r0 or r1; [k0+0] = xr2;;
```

Команды будут выполнены в следующей последовательности:

```
xr0 = [j0+0];;
xr2 = r0 or r1;;
[k0+0] = xr2;;
```

Данная последовательность не может быть прервана внешним прерыванием и обеспечивает выполнение атомарной операции чтение-модификация-запись. Если в процессе выполнения такой линии инструкций возникает запрос на прерывание, то его обработка начинается после выполнения данной линии. Префикс *atomic* может использоваться для упрощения доступа к разделяемым ресурсам. Хотя команды образуют одну линию команд, но выполняются они последовательно. В связи с этим, пользователь должен гарантировать отсутствие генерации исключительной ситуации при выполнении данной линии команд. В противном случае поведение команды непредсказуемо.

10.6.10.6 Команды IALU (загрузка/сохранение/пересылка)

Два независимых целочисленных арифметико-логических устройства (IALU) процессора называются J-IALU и K-IALU. IALU вычисляют адреса в памяти при пересылках данных между памятью и регистрами. Пара IALU позволяет одновременно вычислять адреса для операций чтения или записи с множественными операндами.

Операциями IALU типа загрузка, сохранение и пересылка данных (адресация и перемещение данных) являются:

- прямой и косвенный режимы адресации памяти;
- режим адресации кольцевых буферов;
- режим бит-реверсивной адресации;
- пересылки и загрузка универсальных регистров (Ureg);
- вычисление указателей памяти.

Описание операций IALU, флагов состояния IALU, IALU-предикатов, а также примеры можно найти в разделе «Целочисленное АЛУ (IALU)». В IALU реализованы также команды целочисленной арифметики. Справочная информация по командам целочисленной арифметики IALU приведена в разделе «Команды (целочисленные) IALU».

Соглашения по обозначениям имён регистров, необязательных элементов или вариантов, используемые в справках на команды, подробно освещены в разделе «Целочисленное АЛУ (IALU)». Вкратце эти соглашения таковы:

- { } – необязательные элементы указаны в фигурных скобках; фигурные скобки не являются частью синтаксиса команды;
- | – вертикальная черта разделяет варианты значений элементов;

вертикальная черта не являются частью синтаксиса команды.

- *Jm* или *Km* – буква J или K в именах регистров, выделенных курсивом, указывает на регистр J-IALU или K-IALU соответственно;
- *Jm* – имена регистров, выделенные курсивом, означают шаблоны, вместо которых могут быть подставлены имена одиночных (шаблоны *Jm, Jn, Js, Rs, Ureg_s),* двойных (шаблоны *Rsd, Rmd, Ureg_sd, Ureg_md)* или квадро- (шаблоны *Rsq, Rmq, Ureg_sq, Ureg_mq*) регистров.

Каждая команда, по которой дана справка, занимает одно поле команды в командной строке. Более подробную информацию о командных строках и ограничениях по комбинированию команд можно найти в подразделах «Синтаксис и структура командной строки» и «Правила запараллеливания команд»

Загрузка Ureg-регистра (адресация данных)

Синтаксис

```
Ureg_s = {CB|BR} [Jm +| += Jn | < Imm8>| < Imm32>];
Ureg_sd = {CB|BR} L [Jm +|+= Jn | <Imm8> | <Imm32>];
Ureg_sq = {CB|BR} Q [Jm +|+= Jn |<Imm8>|<Imm32>] ;
Ureg_s = {CB|BR} [Km +|+= Kn |<Imm8>|<Imm32>];
Ureg_sd = {CB|BR} L [Km +|+= Kn |<Imm8>|<Imm32>];
Ureg_sq = {CB|BR} Q [Km +|+= Kn |<Imm8>|<Imm32>];
/* Суффикс в Ureg обозначает: _s – одиночный, _sd – двойной, _sq – квадро-
регистр. */
```

Действие

Эти команды загружают в регистр-приёмник (слева от знака =) содержимое ячейки памяти источника (справа от знака =). При доступе к памяти эти команды поддерживают режимы адресации с предмодификацией без обновления (оператор [+]) и с постмодификацией и обновлением (оператор [+=]). Описание режимов адресации и типов доступа к памяти можно найти в подразделе «Команды IALU (загрузка/сохранение/пересылка)».

Флаги состояния

Команды не влияют на флаги состояния.

Опции

CB Режим адресации кольцевых буферов.

BR Режим бит-реверсивной адресации.

Примеры

Примеры можно найти в подразделе «Примеры программирования IALU».

Сохранение Ureg-регистра (адресация данных)

Синтаксис

```
[Jm +|+= Jn |<Imm8>|<Imm32>] = Ureg_s ;
L [Jm +|+= Jn |<Imm8>|<Imm32>] = Ureg_sd ;
Q [Jm +|+= Jn |<Imm8>|<Imm32>] = Ureg_sq ;
[Km +|+= Kn |<Imm8>|<Imm32>] = Ureg_s ;
L [Km +|+= Kn |<Imm8>|<Imm32>] = Ureg_sd ;
Q [Km +|+= Kn |<Imm8>|<Imm32>] = Ureg_sq ;
```

Действие

Эти команды загружают в ячейку памяти приёмник (слева от знака =) значение из регистра-источника (справа от знака =). При доступе к памяти эти команды поддерживают режимы адресации с предмодификацией без обновления (оператор [+]) и с постмодификацией и обновлением (оператор [+=]). Описание режимов адресации и типов доступа к памяти можно найти в подразделе «Команды IALU (загрузка/сохранение/пересылка)».

Флаги состояния

Команды не влияют на флаги состояния.

Опции

Отсутствуют.

Примеры

Примеры можно найти в подразделе «Примеры программирования IALU».

Загрузка Dreg-регистра с использованием DAB/SDAB (адресация данных)

Синтаксис

```
{X|YIXY}Rs = {CB|BR}  [Jm+= Jn|<Imm8>|<Imm32>] ;
{X|Y|XY}Rsd = {CB|BR} L [Jm +=Jn|<Imm8>|<Imm32>] ;
{XY|YX}Rs = {CB|BR} L [Jm +=Jn|<Imm8>|<Imm32>] ;
{X|Y|XY}Rsq = {CB|BR|DAB|SDAB} Q [Jm += Jn|<Imm8>|<Imm32>] ;
{XY|YX}Rsd = {CB|BR} Q [Jm += Jn\<Imm8>|<Imm32>] ;

{X|Y|XY}Rs = {CB|BR} [Km += Kn|<Imm8>|<Imm32>] ;
{X|Y|XY}Rsd = {CB|BR} L [Km+= Kn|<Imm8>|<Imm32>] ;
{XY|YX}Rs = {CB|BR} L [Km += Kn|<Imm8>|<Imm32>] ;
{X|Y|XY}Rsq = {CB|BR|DAB|SDAB} Q [Km += Kn |<Imm8>|<Imm32>] ;
{XY|YX}Rsd = {CB|BR} Q      [Km += Kn|<Imm8>|<Imm32>] ;
/* Суффикс в R обозначает: _s – одиночный, _sd – двойной, _sq – квадрорегистр. */
/* m должно быть равно 0, 1, 2 или 3 при использовании обратного переноса или
кольцевых буферов */
```

Действие

Эти команды загружают в регистр-приёмник (слева от знака =) содержимое ячейки памяти источника (справа от знака =). При доступе к памяти эти команды поддерживают режим адресации с постмодификацией и обновлением (оператор [+=]). Описание режимов адресации и типов доступа к памяти можно найти в подразделе «Команды IALU (загрузка/сохранение/пересылка)».

Флаги состояния

Команды не влияют на флаги состояния.

Опции

( )	Линейная адресация
CB	Режим адресации кольцевых буферов
BR	Режим бит-реверсивной адресации
DAB	Доступ через буфер выравнивания данных
SDAB	Доступ через буфер выравнивания коротких данных

Примеры

Примеры можно найти в подразделе «Примеры программирования IALU».

Загрузка Dreg-регистра с использованием DAB со смещением (адресация данных)

Синтаксис

```
{XY}Rsq = XDAB Q [Jm += Jn|<Imm8>|<Imm32>] ;
```

/* в DAB X сдвинуться на одно 32-битное слово от смещения, необходимого для получения невыровненных данных; в DAB Y сдвинуться ровно на смещение, необходимое для получения невыровненных данных */

{XY}Rsq = XSDAB Q [Jm +=Jn |<Imm8>|<Imm32>] ;

/* в DAB X сдвинуться на одно 16-битное слово от смещения, необходимого для получения невыровненных данных; в DAB Y сдвинуться ровно на смещение, необходимое для получения невыровненных данных */

{XY}Rsq = YDAB Q [Jm += Jn|<Imm8>|<Imm32>] ;

/* в DAB Y сдвинуться на одно 32-битное слово от смещения, необходимого для получения невыровненных данных; в DAB X сдвинуться ровно на смещение, необходимое для получения невыровненных данных */

{XY}Rsq = YSDAB Q [Jm +=Jn|<Imm8>|<Imm32>] ;

/* в DAB Y сдвинуться на одно 16-битное слово от смещения, необходимого для получения невыровненных данных; в DAB X сдвинуться ровно на смещение, необходимое для получения невыровненных данных */

{XY}Rsqr = XDAB Q [*Km += Kn* |< *Imm8*>|< *Imm32*>] ;

/* в DAB X сдвинуться на одно 32-битное слово от смещения, необходимого для получения невыровненных данных; в DAB Y сдвинуться ровно на смещение, необходимое для получения невыровненных данных */

{XY}Rsq- = XSDAB Q [*Km +=Kn* | <*Imm8*> | < *Imm32*>] ;

/* в DAB X сдвинуться на одно 16-битное слово от смещения, необходимого для получения невыровненных данных; в DAB Y сдвинуться ровно на смещение, необходимое для получения невыровненных данных */

(XY}Rsq = YDAB Q [Km += Kn|<Imm8>|<Imm32>] ;

/* в DAB Y сдвинуться на одно 32-битное слово от смещения, необходимого для получения невыровненных данных; в DAB X сдвинуться ровно на смещение, необходимое для получения невыровненных данных */

(XY}Rsq = YSDAB Q [Km +=Kn |<Imm8>|<Imm32>] ;

/* в DAB Y сдвинуться на одно 16-битное слово от смещения, необходимого для получения невыровненных данных; в DAB X сдвинуться ровно на смещение, необходимое для получения невыровненных данных */

/* m должно быть равно 0, 1, 2 или 3 при использовании обратного переноса или кольцевых буферов */

Действие

Эти команды загружают в регистр-приёмник (слева от знака =) содержимое ячейки памяти источника (справа от знака =). При доступе к памяти эти команды поддерживают режим адресации с постмодификацией и обновлением (оператор [+=]). Описание режимов адресации и типов доступа к памяти можно найти в подразделе «Команды IALU (загрузка/сохранение/пересылка)».

Флаги состояния

Команды не влияют на флаги состояния.

Опции

Отсутствуют.

Примеры

Примеры можно найти в подразделе «Примеры программирования IALU».

Сохранение Dreg-регистра (адресация данных)

Синтаксис

{X|YIXY}Rs = {CB|BR} [Jm+= Jn|<Imm8>|<Imm32>] ;

{X|Y|XY}Rsd = {CB|BR} L [Jm +=Jn|<Imm8>|<Imm32>] ;

{XY|YX}Rs = {CB|BR} L [Jm +=Jn|<Imm8>|<Imm32>] ;

{X|Y|XY}Rsq = {CB|BR|DAB|SDAB} Q [Jm += Jn|<Imm8>|<Imm32>] ;

{XY|YX}Rsd = {CB|BR} Q [Jm += Jn\<Imm8>|<Imm32>] ;

{X|Y|XY}Rs = {CB|BR} [Km += Kn|<Imm8>|<Imm32>] ;

{X|Y|XY}Rsd = {CB|BR} L [Km+= Kn|<Imm8>|<Imm32>] ;

{XY|YX}Rs = {CB|BR} L [Km += Kn|<Imm8>|<Imm32>] ;

{X|Y|XY}Rsq = {CB|BR|DAB|SDAB} Q [Km += Kn |<Imm8>|<Imm32>] ;

{XY|YX}Rsd = {CB|BR} Q [Km += Kn|<Imm8>|<Imm32>] ;

/* Суффикс в *R* обозначает: _s – одиночный, _sd – двойной, _sq – квадрорегистр. */
/* m должно быть равно 0, 1, 2 или 3 при использовании обратного переноса или кольцевых буферов */

Действие

Эти команды загружают в ячейку памяти приёмник (слева от знака =) значение из регистра-источника (справа от знака =). При доступе к памяти эти команды поддерживают режим адресации с постмодификацией и обновлением (оператор [+=]). Описание режимов адресации и типов доступа к памяти можно найти в подразделе «Команды IALU (загрузка/сохранение/пересылка)».

Флаги состояния

Команды не влияют на флаги состояния.

Опции

( ) Линейная адресация
CB Режим адресации кольцевых буферов
BR Режим бит-реверсивной адресации

Примеры

Примеры можно найти в подразделе «Примеры программирования IALU».

Пересылка Ureg-регистра

Синтаксис

Ureg_s = <Imm15>|<Imm32> ;

Ureg_s = Ureg_m ; Ureg_sd = Ureg_md ; Ureg_sq = Ureg_mq ;

Действие

Команды *Ureg=Ureg* пересылают данные из любого регистра-источника на кристалле в любой регистр-приёмник на кристалле. Регистры приёмника и источника определяются номером группы (шесть бит) и номером регистра (пять бит).

Команда такого типа может быть исполнена одним из IALU. Какое именно из IALU должно исполнить эту команду, определяется программно ассемблером. Чтобы выполнить пересылку регистров на обоих вычислительных блоках (пересылка типа Одна-Команда, Множественные-Данные (SIMD)), используйте широковещательный адрес вычислительного блока в номерах группы и Ureg-регистра-источника, и Ureg-регистра-приёмника.

Примечание – Этот класс команд является единственным классом команд, для которых разрешён доступ к группам 63:32. Для всех остальных команд доступ разрешён только к группам 31:0. Информацию о группах регистров можно найти в Спецификации процессора.

Команда *Ureg=Imm* загружает слово данных в регистр-приёмник. Регистр-приёмник определяется номером группы (шесть бит) и номером регистра (пять бит). Здесь слово данных – это 15-битное значение с распространённым знаком, если

только не используется поле непосредственного расширения в командной строке. Команды непосредственного расширения описаны в пункте «Операции непосредственного расширения».

Этот класс команд может быть исполнен одним из IALU: J-IALU или K-IALU. Какое именно из IALU исполняет такую команду определяется ассемблером.

Регистрами УУ являются: CJMP, RETI/IB/S, DBGE, LCx, IVSW, FLAGREG/ST/CL, SQCTL/ST/CL, SQSTAT и SFREG.

Регистрами отладки являются: WPxCTL, WPxSTAT, WPxL, WPxH, CCNTx, PRFM, PRFCNT, TRCBMASK, TRCBPTR, TRCB31-0 и TRCBVAL.

Примеры

XR3:0 = YR31:28;;

Эта команда пересылает четыре регистра между вычислительными блоками X и Y.

SDRCON = K0;;

Эта команда пересылает данные из регистра K0 в регистр SDRCON.

R3:2 = R9:8;;

Эта команда пересылает данные из регистра R9:8 *в регистр* R3:2 *на обоих вычислительных блоках одновременно.*

XR0 = 0x12345678;;

Эта команда загружает в регистр R0 вычислительного блока X константное значение.

Загрузка байта

Синтаксис

Ureg = B [Jm +|+= Jn|<Imm7>|<Imm31>] ({ZF}|{SE});

Действие

Команда загружает байт данных по указанному адресу в регистр-приемник. Новое значение попадает в разряды регистра-приемника [7:0]. Остальные разряды заполняются нулями (если ZF) или битом 7 считанного байта (если SE). Адрес трактуется как указатель на байт, в котором биты с 31-го по 2-й – это адрес слова, а биты 1 и 0 – номер байта в слове. Смещение в поле команды является знаковым.

Ограничения

Если в качестве регистра базы *Jm* используются регистры J26, K26, J27, K27, то при установленном бите SP_byte (бит PU_CR[9]) значение регистра базы предварительно умножается на 4. Если бит SP_byte равен нулю, то вычисление адреса происходит обычным образом.

Не все группы регистров могут использовать данный тип команд (номер группы состоит из трех бит и имеет значение от 0 до 7). Полный номер группы образуется следующим образом: 0-0(X), 1-2(Y), 2-4, 3-5, 4-8(XY), 5-10, 6-12(J), 7-13(K). Таким образом, для загрузки байт должны использоваться только номера 0, 1, 4, 6, 7. Регистр-приемник *Ureg* должен быть регистром модулей X/Y ALU или J/K ALU.

Флаги состояния

Не изменяются.

Опции

- (ZF) Беззнаковое;
- (SE) Знаковое.

Примеры:

K5 = 0x20000. В ячейке памяти по адресу 0x8000 хранится значение 0xC2345678.

J0 = B[K5+1] (ZF);;

J1 = B[K5+3] (SE);;

После выполнения команд, J0 = 0x00000056, J1 = 0xFFFFFFC2.

Сохранение байта

Синтаксис

B [Jm +|+= Jn|<Imm7>|<Imm31>] = bUreg;

Действие

Команда сохраняет из регистра-источника *Ureg* в памяти байт данных по указанному адресу. Сохраняются только разряды [7:0] регистра-источника. Остальные разряды отбрасываются. Адрес трактуется как указатель на байт, в котором биты с 31-го по 2-й – это адрес слова, а биты 1 и 0 – номер байта в слове. Смещение в поле команды является знаковым. При записи не имеет значения знаковый байт или беззнаковый.

Ограничения

Если в качестве регистра базы *Jm* используются регистры J26, K26, J27, K27, то при установленном бите SP_byte (бит PU_CR[9]) значение регистра базы предварительно умножается на 4. Если бит SP_byte равен нулю, то вычисление адреса происходит обычным образом.

Не все группы регистров могут использовать данный тип команд (номер группы состоит из трех бит и имеет значение от 0 до 7). Полный номер группы образуется следующим образом: 0-0(X), 1-2(Y), 2-4, 3-5, 4-8, 5-10, 6-12(J), 7-13(K). Таким образом, для сохранения байт должны использоваться только номера 0, 1, 6, 7. Регистр-источник *Ureg* должен быть регистром модулей X/Y ALU или J/K ALU.

Флаги состояния

Не изменяются.

Опции

Команда может выполняться условно в командной строке. В этом случае бит CC будет установлен в 1. Если команда выполняется безусловно, то бит NF не используется.

Примеры:

K5 = 0x20000.J0 = 0x87654321. В ячейке памяти по адресу 0x8000 хранится значение 0xC2345678.

B[K5+2] = J0;;

После выполнения команды, в памяти по адресу 0x8000 значение 0xC2215678.

Загрузка короткого

Синтаксис

Ureg = S [Jm +|+= Jn|<Imm7>|<Imm31>] ({ZF}|{SE});

Действие

Команда загружает короткое (16 бит) по указанному адресу в регистр-приемник. Новое значение попадает в разряды регистра-приемника [15:0]. Остальные разряды заполняются нулями (если ZF) или битом 15 (если SE) считанного короткого. Адрес трактуется как указатель на короткое, в котором биты с 31-го по 1-й – это адрес слова, а бит 0 – номер короткого в слове. Смещение в поле команды является знаковым.

Ограничения

Если в качестве регистра базы *Jm* используются регистры J26, K26, J27, K27, то при установленном бите SP_byte (бит PU_CR[9]) значение регистра базы предварительно умножается на 2. Если бит SP_byte равен нулю, то вычисление адреса происходит обычным образом.

Не все группы регистров могут использовать данный тип команд (номер группы состоит из трех бит и имеет значение от 0 до 7). Полный номер группы образуется следующим образом: 0-0(X), 1-2(Y), 2-4, 3-5, 4-8(XY), 5-10, 6-12(J), 7-13(K). Таким образом, для загрузки коротких должны использоваться только номера 0, 1, 4, 6, 7. Регистр-приемник *Ureg* должен быть регистром модулей X/Y ALU или J/K ALU.

Флаги состояния

Не изменяются.

Опции

(ZF) Беззнаковое

(SE) Знаковое

Примеры

K5 = 0x10000. В ячейке памяти по адресу 0x8000 хранится значение 0xC2345678.

J0 = S[K5+1] (ZF);;

J1 = S[K5+1] (SE);;

После выполнения команд J0 = 0x0000C234, J1 = 0xFFFFC234.

Сохранение короткого

Синтаксис

S [Jm +|+= Jn|<Imm7>|<Imm31>] = sUreg;

Действие

Команда сохраняет из регистра-источника Ureg короткое (16 бит) в памяти по указанному адресу. Сохраняются только разряды [15:0] регистра-источника. Остальные разряды отбрасываются. Адрес трактуется как указатель на короткое, в котором биты с 31-го по 1-й – это адрес слова, а бит 0 – номер короткого в слове. Смещение в поле команды является знаковым. При записи не имеет значения знаковое короткое или беззнаковое.

Ограничения

Если в качестве регистра базы Jm используются регистры J26, K26, J27, K27, то при установленном бите SP_byte (бит PU_CR[9]) значение регистра базы предварительно умножается на 2. Если бит SP_byte равен нулю, то вычисление адреса происходит обычным образом.

Не все группы регистров могут использовать данный тип команд (номер группы состоит из трех бит и имеет значение от 0 до 7). Полный номер группы образуется следующим образом: 0-0(X), 1-2(Y), 2-4, 3-5, 4-8, 5-10, 6-12(J), 7-13(K). Таким образом, для сохранения короткого должны использоваться только номера 0, 1, 6, 7. Регистр-источник Ureg должен быть регистром модулей X/Y ALU или J/K ALU.

Флаги состояния

Не изменяются

Опции

Команда может выполняться условно в командной строке. В этом случае бит CC будет установлен в 1. Если команда выполняется безусловно, то бит NF не используется.

Примеры

K5 = 0x10000.J0 = 0x87654321. В ячейке памяти по адресу 0x8000 хранится значение 0xC2345678.

S[K5+1] = J0;;

После выполнения команды в памяти по адресу 0x8000 значение 0x43215678.

Загрузка слова или двойного слова по указателю на байт (быстрый вариант)

Синтаксис

{X|Y|XY} Rs = BDAB [Jm +|+= Jn|<Imm8>|<Imm32>] ;

{X|Y|XY} Rsd = BDAB L[Jm +|+= Jn|<Imm8>|<Imm32>] ;

Действие

Команда загружает 32-разрядное слово (4 байта) или 64-разрядное двойное слово (8 байт) по указанному адресу в регистр-приемник. Адрес трактуется как указатель на байт, в котором биты с 31-го по 2-й –это адрес слова, а биты 1 и 0 – номер байта в слове. Операция может рассматриваться как загрузка одинарных или двойных слов по невыровненному (по границе 32-разрядного слова) адресу.

Операция использует DAB и аналогична по своей сути обычной конвейерной загрузке данных с использованием DAB. Ячейка оперативной памяти, буфер DAB и регистр приемник образуют конвейер, по которому выполняется передача данных. В одном такте старое содержимое DAB (частично или целиком) копируется в регистр приемник, и в тоже время новое значение из памяти загружается в DAB. При этом считываемый из памяти операнд, а также операнд в DAB, образуют операнд удвоенной длины, из которого извлекается искомое значение с учетом байтового смещения.

Операция всегда выполняется за один такт.

Ограничения

Регистр-приемник *Rs* должен быть регистром модулей X/Y ALU.

Для реализации операции неявно используется DAB.

Флаги состояния

Не изменяются.

Опции

Отсутствуют.

Загрузка слова или двойного слова по указателю на байт (стандартный вариант)

Синтаксис

{X|Y|XY} Rs = PB [Jm +|+= Jn|<Imm8>|<Imm32>] ;

{X|Y|XY} Rsd = LPB [Jm +|+= Jn|<Imm8>|<Imm32>] ;

Действие

Команда загружает 32-разрядное слово (4 байта) или 64-разрядное двойное слово (8 байт) по указанному адресу в регистр-приемник. Адрес трактуется как указатель на байт, в котором биты с 31-го по 2-й – это адрес слова, а биты 1 и 0 – номер байта в слове. Операция может рассматриваться, как загрузка одинарных или двойных слов по невыровненному (по границе 32-разрядного слова) адресу.

Операция может использовать DAB. Если получаемый после вычисления итоговый адрес оказывается выравнен на границе слова (для PB) или двойного слова (для LPB), команда выполняется за один такт без использования DAB аналогично стандартной команде загрузки слова или двойного слова. Если итоговый адрес оказывается невыровненным, команда выполняется за два такта. Команда использует DAB и по своей сути эквивалентна исполнению двух последовательных операций загрузки быстрого варианта. В первом такте значение из памяти загружается в DAB. Во втором такте следующее значение считывается из памяти и вместе со значением из DAB образует операнд удвоенной длины, из которого извлекается искомое значение и загружается в регистр приемника.

Операция выполняется за один или два такта.

Ограничения

Регистр-приемник *Rs* должен быть регистром модулей X/Y ALU.

Для реализации операции неявно может использоваться DAB.

Флаги состояния

Не изменяются.

Опции

Отсутствуют.

Примеры

```
R0 = PB[J5+=4];;
```

Сохранение слова по указателю на байт

Синтаксис

PB [Jm +|+= Jn|<Imm8>|<Imm32>] = Ureg;

Действие

Команда записывает 32-разрядное слово (4 байта) из регистра источника Rs по указанному адресу в память. Адрес трактуется как указатель на байт, в котором биты с 31-го по 2-й – это адрес слова, а биты 1 и 0 – номер байта в слове. Операция может рассматриваться как сохранение слова по невыровненному (по границе 32-разрядного слова) адресу.

Если получаемый после вычисления итоговый адрес оказывается выравнен на границе слова, команда выполняется за один такт аналогично стандартной команде записи слова. Если же итоговый адрес оказывается невыровненным, команда выполняется за два такта: в первом такте пишется одна часть слова, а во втором – вторая часть слова, по увеличенному на 4 значению адреса. При этом в первом и во втором такте пишутся только нужные байты

Операция выполняется за один или два такта.

Ограничения

Не все группы регистров могут использовать данный тип команд (номер группы состоит из трех бит и имеет значение от 0 до 7). Полный номер группы образуется следующим образом: 0-0(X), 1-2(Y), 2-4, 3-5, 4-8, 5-10, 6-12(J), 7-13(K). Таким образом, для сохранения слова должны использоваться только номера 0, 1, 6, 7. Регистр-источник *Ureg* должен быть регистром модулей X/Y ALU или J/K ALU.

Также имеются ограничения на использование регистра Ureg в качестве приемника результата операции в одной командной строке с командой сохранения Ureg. Из-за того, что команда может выполняться за два такта, в данном сочетании во втором такте возможна запись нового значения регистра вместо старого.

Флаги состояния

Не изменяются.

Опции

Отсутствуют.

Примеры:

```
PB[J5+=4] = J3;;
```

Загрузка слова или двойного слова по указателю на короткое (быстрый вариант)

Синтаксис

{X|Y|XY} Rs = SDAB [Jm +|+= Jn|<Imm8>|<Imm32>] ;

{X|Y|XY} Rsd = SDAB L[Jm +|+= Jn|<Imm8>|<Imm32>] ;

Действие

Команда загружает 32-разрядное слово (2 коротких) или 64-разрядное двойное слово (4 коротких) по указанному адресу в регистр-приемник. Адрес

трактуется как указатель на короткое, в котором биты с 31-го по 1-й – адрес слова, а бит 0 – номер короткого в слове. Операция может рассматриваться как загрузка одинарных или двойных слов по невыровненному (по границе 32-разрядного слова) адресу.

Операция использует DAB и аналогична по своей сути обычной конвейерной загрузке данных с использованием DAB. Ячейка оперативной памяти, буфер DAB и регистр приемник образуют конвейер, по которому выполняется передача данных. В одном такте старое содержимое DAB (частично или целиком) копируется в регистр приемник, и в тоже время новое значение из памяти загружается в DAB. При этом считываемый из памяти операнд, а также операнд в DAB, образуют операнд удвоенной длины, из которого извлекается искомое значение с учетом смещения.

Операция всегда выполняется за один такт.

Ограничения

Регистр-приемник *Rs* должен быть регистром модулей X/Y ALU.

Для реализации операции неявно используется DAB.

Флаги состояния

Не изменяются.

Опции

Отсутствуют.

Загрузка слова или двойного слова по указателю на короткое (стандартный вариант)

Синтаксис

{X|Y|XY} Rs = PS [Jm +|+= Jn|<Imm8>|<Imm32>] ;

{X|Y|XY} Rsd = LPS [Jm +|+= Jn|<Imm8>|<Imm32>] ;

Действие

Команда загружает 32-разрядное слово (2 коротких) или 64-разрядное двойное слово (4 коротких) по указанному адресу в регистр-приемник. Адрес трактуется как указатель на короткое, в котором биты с 31-го по 1-й – это адрес слова, а бит 0 – номер короткого в слове. Операция может рассматриваться как загрузка одинарных или двойных слов по невыровненному (по границе 32-разрядного слова) адресу.

Операция может использовать DAB. Если получаемый после вычисления итоговый адрес оказывается выравнен на границе слова (для PS) или двойного слова (для LPS), команда выполняется за один такт без использования DAB аналогично стандартной команде загрузки слова или двойного слова. Если же итоговый адрес оказывается не выровненным, команда выполняется за два такта. Команда использует DAB и по своей сути эквивалентна исполнению двух последовательных операций загрузки быстрого варианта. В первом такте значение из памяти загружается в DAB. Во втором такте следующее значение считывается из памяти и вместе со значением из DAB образует операнд удвоенной длины, из которого извлекается искомое значение и загружается в регистр приемника.

Операция выполняется за один или два такта.

Ограничения

Регистр-приемник *Rs* должен быть регистром модулей X/Y ALU.

Для реализации операции неявно может использоваться DAB.

Флаги состояния

Не изменяются.

Опции

Отсутствуют.

Примеры:

```
R0 = PS[J5+=4];;
```

Сохранение слова по указателю на короткое

Синтаксис

PS [Jm +|+= Jn|<Imm8>|<Imm32>] = Ureg;

Действие

Команда записывает 32-разрядное слово (4 байта) из регистра источника Rs по указанному адресу в память. Адрес трактуется как указатель на короткое, в котором биты с 31-го по 1-й – это адрес слова, а бит 0 – номер короткого в слове. Операция может рассматриваться как сохранение слова по невыровненному (по границе 32-разрядного слова) адресу.

Если получаемый после вычисления итоговый адрес оказывается выравнен на границе слова, команда выполняется за один такт аналогично стандартной команде записи слова. Если же итоговый адрес оказывается невыровненным, команда выполняется за два такта: в первом такте пишется одна часть слова, а во втором – вторая часть слова по значению адреса, увеличенному на 2. При этом в первом и во втором такте пишутся только нужные байты.

Операция выполняется за один или два такта.

Ограничения

Не все группы регистров могут использовать данный тип команд (номер группы состоит из трех бит и имеет значение от 0 до 7). Полный номер группы образуется следующим образом: 0-0(X), 1-2(Y), 2-4, 3-5, 4-8, 5-10, 6-12(J), 7-13(K). Таким образом, для сохранения слова должны использоваться только номера 0, 1, 6, 7. Регистр-источник *Ureg* должен быть регистром модулей X/Y ALU или J/K ALU.

Также имеются ограничения на использование регистра Ureg в качестве приемника результата операции в одной командной строке с командой сохранения Ureg. Из-за того, что команда может выполняться за два такта, в данном сочетании во втором такте возможна запись нового значения регистра вместо старого.

Флаги состояния

Не изменяются.

10.7 Команды устройства управления (УУ)

УУ осуществляет выборку команд из памяти и выполняет команды управления потоком исполнения программы.

УУ поддерживает следующие операции управления потоком исполнения программы:

- вычисление адреса следующей команды для выборки;
- обслуживание буфера выравнивания команд (IAB), кэширующего выбранные команды;
- обслуживание буфера целевых адресов перехода (BTB), уменьшающего задержку при переходах;
- декремент счётчика циклов;
- вычисление предикатов (для условных команд);
- реакция на прерывания (с переключением потока управления).

Описание операций устройства управления, условного выполнения, конвейерных эффектов, а также примеры можно найти в разделе «Устройство управления».

Соглашения по обозначениям имён регистров, необязательных элементов или вариантов, используемые в справках на команды, приведены в подразделе «Регистры регистровых файлов». Вкратце эти соглашения таковы:

- { } – необязательные элементы указаны в фигурных скобках; фигурные скобки не являются частью синтаксиса команды;
- | – вертикальная черта разделяет варианты значений элементов;

вертикальная черта не является частью синтаксиса команды;

- *Label* – шаблон задаваемой пользователем программной метки (выделяется курсивом). При использовании *Label* в программе в качестве ссылки преобразуется ассемблером либо в 15- или 32-битный PC-относительный адрес, либо в 32-битный абсолютный адрес. Замена на 15-битный адрес (если это удаётся) производится в случае, когда метка *Label* и команда перехода не неё содержатся в одной и той же программной секции (. SECTION). Замена на 32-битный адрес производится в случае, когда метка и команда перехода содержатся в разных секциях.

Каждая команда, по которой дана справка, занимает одно поле команды в командной строке. Более подробную информацию о командных строках и ограничениях по комбинированию команд можно найти в подразделах «Синтаксис и структура командной строки» и «Правила запараллеливания команд».

10.7.1 *Переход/Вызов*

Синтаксис

{IF *Condition,* } JUMP|CALL <*Label*> { (NP) } {(ABS)} ;

Действие

Эти команды обеспечивают выполнение ветвлений посредством переходов (JUMP) и вызовов (CALL). Команда JUMP передаёт управление по адресу Label. Команда CALL также передаёт управление по адресу Label. Отличие в том, что при обработке вызова устройство управления сначала записывает адрес возврата (адрес командной строки, непосредственно следующей за командной строкой с вызовом) в регистр CJMP, а затем переходит по адресу подпрограммы.

Если переход является условным (присутствует префикс-предикат if condition), то он выполняется тогда, когда значение предиката есть «истина». Если опция прогнозирования перехода установлена в Не-прогнозируемый (NP), устройство управления предполагает, что переход не выполняется. Команда использует PC-относительный адрес перехода, если не указана опция абсолютного адреса (ABS). Более подробно см. в подразделе «Отработка переходов».

Опции

NP Прогнозирование перехода устанавливается в состояние Не-прогнозируемый.

ABS Адрес перехода – в поле непосредственных расширений; при использовании опции ABS отрицательное число в этом поле (вместо *<Label>)* дополняется нулём (не знаком).

Примеры:

```
R21 = R21 + R31;;
IF AEQ, JUMP label1;;
/* Используется логическое ИЛИ значений AEQ вычислительных блоков X и Y. Если
результат «истина», произойдёт переход по адресу label1. */
XFR31 = R6 * R2;;
IF XMLT, JUMP label2;;
/* Используется предикат MLT (умножителя) вычислительного блока X. Если
результат «истина», произойдёт переход по адресу label2. */
XR31 += ASHIFT R0 BY 1;;
IF XSEQ, JUMP label3 (NP);;
/* Используется предикат SEQ (сдвигового устройства) вычислительного блока X.
Если результат «истина», произойдёт переход по адресу 1abel3. NP обозначает «без
прогнозирования перехода». */
```

10.7.2 *Вычисляемый переход/вызов*

Синтаксис

{IF *Condition*,) CJMP|CJMP_CALL {(NP)} {(ABS)} ;

Действие

Эти команды обеспечивают выполнение ветвлений посредством вычисляемых переходов (CJMP) и вызовов (CJMP_CALL). Команда CJMP в конце подпрограммы (вызванной с помощью CALL) предписывает управляющему устройству выполнить переход по адресу в регистре CJMP. Команда CJMP_CALL передаёт управление на подпрограмму, используя вычисляемый адрес перехода (CJMP). Один из способов загрузить вычисляемый адрес перехода – это использовать опцию (CJMP) в команде сложения/вычитания IALU. Команда CJMP_CALL передаёт управление по адресу, указанному в регистре CJMP, а затем загружает адрес возврата в CJMP.

Если переход является условным (присутствует префикс-предикат if condition), то он выполняется тогда, когда значение предиката есть «истина». Если опция прогнозирования перехода установлена в Не-прогнозируемый (NP), устройство управления предполагает, что переход не выполняется. Команда использует PC-относительный адрес перехода, если не указана опция абсолютного адреса (ABS). Более подробно см. в подразделе «Отработка переходов».

Если прогнозирование переходов включено (опция NP не указана), вызов должен быть по абсолютному адресу (опция ABS включена).

Опции

NP Прогнозирование перехода устанавливается в состояние Не-прогнозируемый.

ABS Адрес перехода – в поле непосредственных расширений.

Более подробно об опциях NP и ABS см. в подразделе «Буфер целевых адресов перехода (BTB)».

Примеры

```
IF AEQ, CJMP (ABS) ;; /* прогнозируемый, абсолютный адрес */
IF AEQ, CJMP (ABS) (NP) ;;  /* НЕ прогнозируемый, абсолютный адрес*/
IF AEQ, CJMP (NP) ;; /* НЕ прогнозируемый, PC-относительный адрес*/
IF AEQ, CJMP_CALL (ABS) ;; /* прогнозируемый, абсолютный адрес */
IF AEQ, CJMP_CALL (ABS) (NP) ;; /* НЕ прогнозируемый, абсолютный адрес */
IF AEQ, CJMP_CALL (NP) ;; /* НЕ прогнозируемый, PC-относительный адрес */
```

10.7.3 *Возврат (из прерывания)*

Синтаксис

{IF *Condition,*} RETI|RTI {(NP)}{(ABS)} ;

Действие

УУ поддерживает механизм прерывания исполнения посредством аппаратных прерываний (внешние линии IRQ3-0 и внутренние причины) и программных прерываний (программа устанавливает бит в регистре-защёлке прерываний). На рисунке 186 сравниваются способы обработки прерывания с использованием команд RETI и RTI.

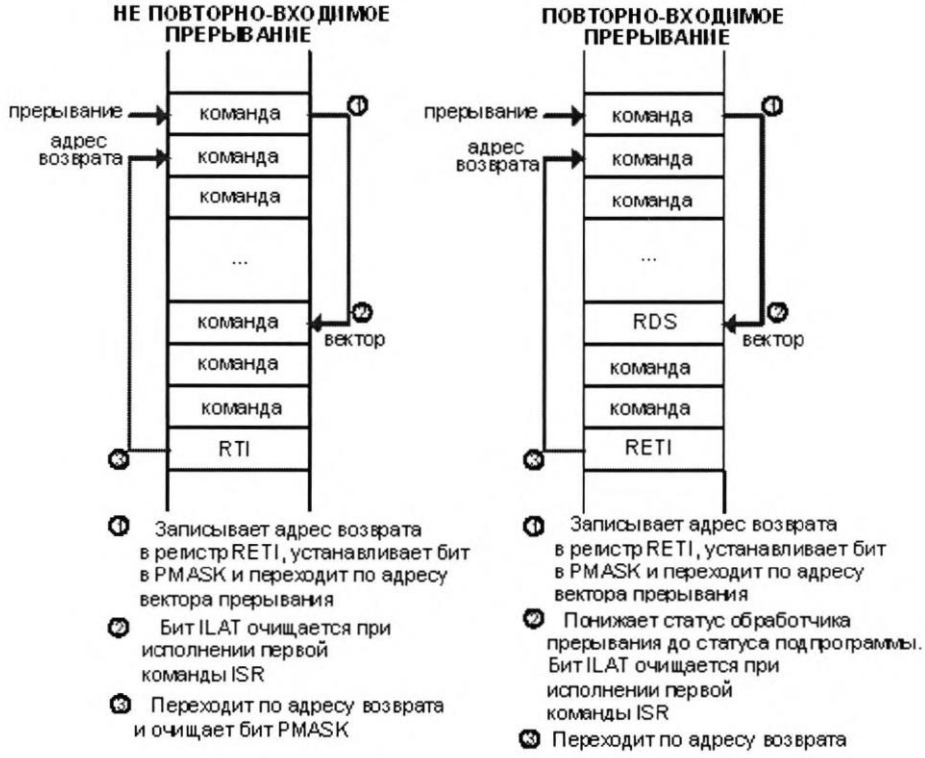


Рисунок 186 – Обработка не-повторно-входимых и повторно-входимых прерываний

Если возврат является условным (присутствует префикс-предикат if *condition*), то он выполняется тогда, когда значение предиката есть «истина». Если опция прогнозирования перехода установлена в Не-прогнозируемый (NP), УУ предполагает, что возврат не выполняется. Опция абсолютного адреса (ABS) обязательна, т.к. с регистром RETI невозможна PC-относительная адресация. Более подробно см. в подразделе «Отработка прерываний».

Опции

NP Прогнозирование перехода устанавливается в состояние Не-прогнозируемый.

Примеры

```
IF AEQ, RTI (ABS) ;; /* прогнозируемый, абсолютный адрес */
IF AEQ, RTI (NP) (ABS) ;; /* НЕ прогнозируемый, абсолютный адрес */
IF AEQ, RETI (ABS) ;; /* прогнозируемый, абсолютный адрес */
IF AEQ, RETI (NP) (ABS) ;; /* НЕ прогнозируемый, абсолютный адрес */
```

10.7.4 *Понизить статус (прерывания до подпрограммы)*

Синтаксис

{IF *Condition*,} RDS ;

Действие

Эта команда переводит процессор из режима выполнения обработчика прерывания в режим выполнения подпрограммы, превращая тем самым прерывание в повторно-входимое. Подробная информация о функциях обработки прерываний для не-повторно-входимых и повторно-входимых прерываний представлена на рисунке 186.

Если возврат является условным (присутствует префикс-предикат *if condition*), то он выполняется тогда, когда значение предиката есть «истина». Более подробно см. в подразделе «Отработка прерываний».

Опции

Отсутствуют.

Примеры

<код обработчика прерывания, часть 1>
RDS;;
<код обработчика прерывания, часть 2>
RETI;; /* После использования команды RDS возврат осуществляется по RETI, а не по RTI. */

В части 1 обработчика все прерывания с приоритетом меньшим, чем приоритет обрабатываемого прерывания, запрещены. В части 2 после команды RDS (которая также может быть условной) все прерывания разрешены. Если данное прерывание происходит во время выполнения обработчика прерывания более низкого приоритета, то бит прерывания более низкого приоритета в PMASK остаётся установленным и после выполнения команды RDS.

10.7.5 *Если - то (условное выполнение)*

Синтаксис

IF *Condition*;

DO, *instruction*; DO, *instruction*; DO, *instruction* ;;

/* Такой синтаксис позволяет подчинить одному предикату до трёх условных команд. Пропуск DO перед командой делает её безусловной. */

Действие

Любая команда (кроме перечисленных ниже) из всего набора может быть условной. Здесь каждая команда instruction выполняется, если предикат Condition принимает значение «истина». Более подробно см. в подразделе «Отработка условных команд».

Команды NOP, IDLE, BTBINV, BTBEN, BTBDIS, BTBLOCK, BTBELOCK, TRAP и EMUTRAP не могут быть условными. Например, следующая командная строка некорректна:

IF AEQ; DO IDLE; /* некорректно! */

Примеры

IF JEQ; DO, J0 = J1 + J2; K0 = K1 + K2;;

Если значение JEQ есть «истина», то команда J-1ALU выполняется, а если – «ложь», то команда J-IALU не выполняется. Команда K-IALU выполняется всегда.

10.7.6 *Если - иначе (условное ветвление и выполнение)*

Синтаксис

IF *Condition,* JUMP|CALL|CJMP|CJMP_CALL ;

ELSE, *instruction*; ELSE, *instruction*; ELSE, *instruction* ;;

/* Такой синтаксис позволяет подчинить одному предикату до трёх условных команд. Пропуск ELSE перед командой делает её безусловной. */

Действие

Любая команда (кроме перечисленных ранее) из всего набора может быть условной. Условием выполнения команд instruction является невыполнение условия

перехода (при любом типе команд перехода). Более подробно см. в подразделе «Отработка условных команд».

Примеры

IF MEQ, CJMP; ELSE, XR0 = R5 + R6; YR8 = R9*R10;;

Если предыдущий результат умножения равен нулю, то выполняется CJMP, а команда сложения не выполняется. Если предыдущий результат умножения не равен нулю, то выполняется команда сложения. Команда умножения выполняется всегда.

10.7.7 *Загрузить значение предиката во флаг статических предикатов*

Синтаксис

{X|Y|XY)SF1|SF0 = *Предикаты_Выч. Блк*. ;

{X|Y|XY}SF1|SF0 += AND|OR|XOR *Предикаты_Выч. Блк*. ;

ISF1 | ISF0 = Предикаты_IALU| *Предикаты_Выч. Блк*. |*Предикаты_УУ* ;

ISF1 | ISF0 += AND | OR | XOR *Предикаты_IALU* | *Предикаты_Выч. Блк.* |*Предикаты_УУ*;

Действие

Эти команды устанавливают значения флагов статических предикатов, определяя тем самым значения предикатов, зависящих от этих флагов.

Предикаты вычисляются заново каждый раз, как изменяются флаги, от которых они зависят. Для сохранения значения предиката, вычисленного несколько циклов назад и с тех пор изменившегося, можно воспользоваться регистром флагов статических предикатов (SFREG). Значение предиката может быть загружено в SFREG в нужный момент, а использовано позже. Другой способ использования SFREG – для вычисления сложных предикатов. Функции могут зависеть от старых значений флагов статических предикатов и от других предикатов. Более подробную информацию о предикатах можно найти в подразделе «Сводка команд устройства управления».

Примеры

```
SF0 = XMLT ;;          /* Статический флаг xSF0 равен значению предиката XMLT. */
SF1 += OR XAEQ ;;      /* Статический флаг xSF1 равен его предыдущему значению,
                          скомбинированному по ИЛИ со значением предиката xAEQ. */
ISF0 = JLT ;;          /* Статический флаг ISF0 равен значению предиката JLT. */
ISF1 += OR KEQ ;;      /* Статический флаг ISF1 равен его предыдущему значению,
                          скомбинированному по ИЛИ со значением предиката KEQ. */
```

10.7.8 *Бездействие*

Синтаксис

IDLE ;

Действие

Эта команда переводит процессор в состояние IDLE, когда процессор перестаёт выполнять команды в ожидании какого-либо прерывания.

10.7.9 *Включить/Выключить обработку прерываний*

Синтаксис

IRQLOCK ;

IRQELOCK ;

Действие

IRQLOCK – запрет на обработку прерываний;

IRQELOCK – снятие запрета на обработку прерываний.

Команды выполняются только в режиме супервизора. При выполнении этих инструкций не имеет значения, разрешены или запрещены прерывания. Выполнение команды IRQELOCK не означает разрешение прерываний. Она только снимает блокировку, если блокировка была установлена с помощью команды IRQLOCK.

Использование данных команд позволяет более оперативно выполнить критичный к прерываниям код.

Примечание – Аналогичную процедуру (с блокировкой и разблокировкой прерываний) можно выполнить с помощью регистра SQCTL. Однако, запись в регистр SQCTL приводит к перезапуску конвейера, а это 10 тактов потерь. Таким образом, использование регистра SQCTL – это как минимум 20 дополнительных тактов потерь.

10.7.10 *Включить/Выключить BTB*

Синтаксис

BTBEN ;

BTBDIS ;

Действие

Эта команда включает и выключает режим использования буфера целевых адресов перехода BTB. После выполнения команды BTBEN включается режим использования BTB, в котором процессор сохраняет адреса выполненных прогнозируемых переходов и при выборке каждой команды проверяет её на присутствие в кэш-памяти BTB для прогнозирования изменений в потоке исполнения программы.

Использование команд BTBEN, BTBDIS и BTBINV влечёт задержку в работе конвейера команд на девять тактов. Эти команды вызывают очистку конвейера, выборка продолжается только после того, как команда управления BTB достигнет фазы EX2 конвейера.

10.7.11 *Сделать записи в BTB недействительными*

Синтаксис

BTBINV ;;

Действие

Эта команда делает все записи в BTB недействительными. Эта команда должна выполняться всякий раз, когда код во внутренней памяти процессора перегружается.

10.7.12 *Ловушка*

Синтаксис

TRAP (*<Inm5>)* ;;

Действие

Эта команда вызывает программное исключение (ловушку) и записывает 5-битное непосредственное значение в поле SPCCMD регистра SQSTAT. См. Спецификацию процессора, раздел «Прерывания», подраздел «Исключения».

10.7.13 *Ловушка эмулятора*

Синтаксис

EMUTRAP ;;

Действие

Эта команда вызывает программное исключение эмулятора (ловушку эмулятора) сразу после выполнения текущей командной строки. Следующее значение PC сохраняется в регистре DBGE, и устройство управления начинает читать команды из регистра EMUIR, получаемые по каналу JTAG. См. Спецификацию процессора, разделы «Эмулятор» и «Отладка в режиме эмуляции».

10.7.14 *Нет операции*

Синтаксис

NOP ;;

Действие

Операции не выполняются – команда просто занимает одно поле команды.

Команда NOP может быть исполнена наряду с другими командами устройства управления в той же командной строке. В одной строке может быть несколько команд NOP.

Приложение А
Краткий справочник

Настоящее приложение содержит сжатое описание программной модели процессора и его языка ассемблера. Предназначено для использования в качестве справочника по программированию на ассемблере, описывающего синтаксис языка и содержащего типичные фрагменты кода. В приложении не дается сведений ни по функциональному назначению команд процессора, ни по ограничениям на них, накладываемым конвейером, ни по механизмам их запараллеливания.

Ниже приводятся некоторые разделы настоящего документа, с которыми программист должен ознакомиться перед использованием краткого справочника:

- Архитектура процессора;
- Синтаксис и структура командной строки;
- Правила запараллеливания команд;
- Регистры регистровых файлов;
- Операции ALU;
- Операции CLU;
- Операции умножителя;
- Операции сдвигового устройства;
- Операции IALU;
- Операции устройства управления.

А.1 Краткий справочник по ALU

Примеры использования этих команд можно найти в подразделе «Примеры использования ALU».

Листинг А-1 – Команды ALU операций с фиксированной точкой

{X|Y|XY}{S|B} Rs = *Rm* +|- *Rn* {({S | SU} {NF})} ;[1]
{X|Y|XY}{L|S|B}*Rsd* = *Rmd* +|- *Rnd* {({S|SU}{NF})} ;[1]
{X|Y|XY}*Rs* = *Rm* + CI {-1} ;
{X|Y|XY}*LRsd* = *Rmd* + CI {-1} ;
{X|Y|XY}*Rs* = *Rm* + *Rn* + CI { ({S|SU}{NF}) } ;[1]
{X|Y|XY}*Rs* = *Rm* - *Rn* + CI -1 {({S|SU}{NF})} ;[1]
{X|Y|XY}*LRsd* = *Rmd* + *Rnd* + CI {({S|SU}{NF})} ;[1]
{X|Y|XY}*LRsd* = *Rmd* - *Rnd* + CI -1 {({S|SU}{NF})} ;[1]
{X|Y|XY}{S|B}*Rs=[Rm*+|-*Rn)*/2{({T}{U}{NF})} ;[2]
{X|Y|XY}{L|S|B}*Rsd=[Rmd*+|-*Rnd)/2*{({T}{U}{NF})} ;[2]
{X|Y|XY}{S|B}*Rs* = ABS *Rm* {(NF)} ;
{X|Y|XY}{L|S|B}*Rsd* = ABS *Rmd* {(NF)} ;

{X|Y|XY}{S|B} Rs = ABS (*Rm+Rn)* {({X} {NF})} *;*[3]

[1] Допустимые опции: ( ): без насыщения, (S): с насыщением, знаковое, (SU): с насыщением, беззнаковое.
[2] Допустимые опции: ( ) округление к ближайшему чётному, (T): знаковое, с усечением, (U): беззнаковое, округление к ближайшему чётному, (TU): беззнаковое, с усечением.
[3] Допустимые опции: ( ): знаковое, с насыщением, (X): расширенное для ABS с беззнаковым насыщением.

{X|Y|XY}{L|S|B}Rsd = ABS *(Rmd + Rnd)* {({X} {NF})} ;[1]
{X|Y|XY}{S|*B*}*Rs* = ABS *(Rm - Rn)* {({X|U}{NF})} ;[1]
{X|Y|XY}{L|*S*|B}*Rsd* = ABS *(Rmd-Rnd)* {({X|U}{NF})} ;[2]
{X|Y|XY}{S|B}*Rs* = - *Rm* {(NF)} ;
{X|Y|XY}{L|S|B}*Rsd* = - *Rmd* {(NF)} ;
{X|Y|XY}{ S|B}Rs = MAX | MIN *(Rm, Rn)* {({U} {Z}{NF})} ;[2]
{X|Y|XY}{ L|S|B}Rsd = MAXI MIN *(Rmd, Rnd)* {({U} {Z} {NF})} ;[3]
{X|Y|XY} S|BRsd = VMAX | VMIN *(Rmd, Rnd)* {(NF)} ;
{X|Y|XY}{S|B}Rs= INC | DEC *Rm*{({S|SU} {NF})} ;[3]
{X|Y|XY}{L|S|B}Rsd= INC | DEC*Rmd* {({S|SU} {NF})} ;[4]
{X|Y|XY}{S|B}COMP(*Rm, Rn)* {(U)} ;[4]
{X|Y|XY}{L|S|B}COMP*(Rnd,Rnd)* {(U)} ;[5]
{X|Y|XY}{S|B}*Rs* = CLIP *Rm* BY *Rn* {(NF)} ;
{X|Y|XY}{L|S|B}Rsd = CLIP *Rmd* BY *Rnd* {(NF)} ;
{X|Y|XY}*Rs* = SUM *SRm*|*BRm* {({U}{NF})} ;[5]
{X|Y|XY} *Rs* = SUM S*Rmd* | B*Rmd* {({U} {NF})} ;[5]
{X|Y|XY}*Rs* = ONES *Rm*|*Rmd* {(NF)} ;
{X|Y|XY}PR1:0 = *Rmd* {(NF)} ;
{X|Y|XY}*Rsd* = PR1:0 {(NF)} ;
{X|Y|XY}*Rs* = BFOINC *Rmd* {(NF)} ;
{X|Y|XY}PRO|PR1 += ABS (S*Rmd - SRnd)*{({U}{NF})} ;[5]
{X|Y|XY}PRO|PR1 += ABS *(*B*Rmd - BRnd)*{({U}{NF})} ;[5]
{X|Y|XY}PRO|PR1 += SUM S*Rm* {({U}{NF})} ;[5]
{X|Y|XY}PRO|PR1 += SUM S*Rmd* {({U}{NF})};[5]
{X|Y|XY}PRO|PR1 += SUM B*Rm* {({U}{NF})} ;[5]
{X|Y|XY}PRO|PR1 += SUM B*Rmd* {({U}{NF})} ;[5]
{X|Y|XY}{S |B} *Rs = Rm + Rn, Ra = Rm - Rn* {(NF)} ; *(двойная операция)*
{X|Y|XY}{ L|S|B}*Rsd = Rmd + Rnd, Rad = Rmd-Rnd*{(NF)} ; *(двойная операция)*

Листинг А-2 – Команды ALU логических операций

{X|Y|XY}*Rs* = PASS *Rm ;*
{X|Y|XY}*LRsd* = PASS *Rmd* ;
{X|Y|XY}*Rs* = *Rm* AND|AND NOT|OR|XOR *Rn* {(NF)} ;
{X|Y|XY}*LRsd* = *Rmd* AND | AND NOT | OR | XOR *Rnd* {(NF)} ;
{X|Y|XY}*Rs* = NOT *Rm* {(NF)} ;
{X|Y|XY}*LRsd* = NOT *Rmd* {(NF)} ;

Листинг А-3 – Различные команды ALU операций с фиксированной точкой

{X|Y|XY}Rsd = EXPAND *SRm* {+|- *SRn}* {({I|IU}{NF})} ;[5]
{X|Y|XY}*Rsq* = EXPAND *SRmd* {+|- S*Rnd}* {({I|IU}{NF})} ;[6]
{X|Y|XY}*SRsd* = EXPAND *BRm* {+|- *BRn}* {({I|IU}{NF})} ;[6]
{X|Y|XY} *SRsq* = EXPAND *BRmd* { + |- BAnd} {({I| IU} {NF})} ;[6]
{X|Y|XY}*SRs* = COMPACT *Rmd* { + |- *Rnd*}{ ({T|I|IS|ISU}{NF})} ;[6]
{X|Y|XY}BRs=COMPACT SRmd{+|-SRnd}{({T|I|IS|ISU}{NF})};[7]
{X|Y|XY} Rs = COMPACT LRmd {({U | IS | ISU} {NF})} ;

---

[1] Допустимые опции: ( ): знаковое, с насыщением, (X): расширенное для ABS с беззнаковым насыщением, (U): беззнаковое.
[2] Допустимые опции:( ): обычное знаковое сравнение, (U): сравнение беззнаковых чисел, (Z): результат равен нулю, если выбран Rn, в операциях MIN/MAX; иначе результат равен Rm, (UZ): беззнаковое сравнение с опцией (Z) описанной выше.
[3] Допустимые опции: ( ): без насыщения, (S): с насыщением, знаковое, (SU): с насыщением, беззнаковое.
[4] Допустимые опции: ( ): знаковое, (U):беззнаковое.
[5] Допустимые опции: ( ): дробное, (I): знаковое целое, (IU): беззнаковое целое.
[6] Допустимые опции: ( ): дробное округление, (I): целое, без насыщения, (T): дробное, с усечением, (IS): целое, с насыщением, знаковое, (ISU): целое, с насыщением, беззнаковое.

{X|Y|XY}LRsd = COMPACT QRmq {({U|IS|ISU}{NF})} ;

Листинг А-4 – Команды ALU операций с плавающей точкой

{X|Y|XY}*FRs = Rm* +|- *Rn* {({T}{NF})} ;[1]
{X|Y|XY}*FRsd = Rmd* +|- *Rnd* {({T}{NF})} ;[1]
{X|Y|XY}*FRs = (Rm* +|- *Rn)/2* {({T}{NF})} ;[1]
{X|Y|XY}F*Rsd = (Rmd* +|- *Rnd)/2* {({T}{NF})} ;[1]
{X|Y|XY}F*Rs* = MAX |MIN *(Rm, Rn)* { (NF) } ;[2]
{X|Y|XY}F*Rsd* = MAX | MIN *(Rmd, Rnd)* {(NF)} ;[2]
{X|Y|XY}*FRs* = ABS *Rm* {(NF)} ;
{X|Y|XY}*FRsd* = ABS *Rmd* {(NF)} ;
{X|Y|XY}F*Rs* = ABS (*Rm* + | - *Rn*){({T) {NF})} ;[1]
{X|Y|XY}F*Rsd* = ABS *(Rmd* +|- *Rnd)* {({T}{NF})} ;[1]
{X|Y|XY}*FRs = - Rm* {(NF)} ;
{X|Y|XY}*FRsd = - Rmd* {(NF)} ;
{X|Y|XY}FCOMP *(Rm, Rn) ;*
{X|Y|XY}FCOMP *(Rmd, Rnd)* ;
{X|Y|XY}*Rs* = FIX *FRm|FRmd* {BY Rn} {({T}{NF})} ;[1]
{X|Y|XY}*FRs|FRsd* = FLOAT *Rm* {BY Rn} {({T}{NF})} ; [1]
{X|Y|XY}*FRsd* = EXTD *Rm* {(NF)} ;
{X|Y|XY*}FRs* = SNGL *Rmd* {({T}{NF})} ;
{X|Y|XY}*FRs* = CLIP *Rm* BY *Rn* {(NF)) ;
{X|Y|XY}*FRsd* = CLIP *Rmd* BY *Rnd* {(NF)} ;
{X|Y|XY}FRs = *Rm* COPYSIGN *Rn* { (NF) } ;
{X|Y|XY}FRsd = *Rmd* COPYSIGN *Rnd* { (NF) } ;
{X|Y|XY}*FRs* = SCALB F*Rm* BY *Rn* {(NF)} ;
{X|Y|XY}*FRsd* = SCALB F*Rmd* BY *Rn* {(NF)} ;
{X|Y|XY}*FRs* = PASS *Rm* ;
{X|Y|XY}*FRsd* = PASS *Rmd* ;
{X|Y|XY}*FRs* = RECIPS *Rm* {(NF)} ;
{X|Y|XY}F*Rsd* = RECIPS *Rmd* {(NF)} ;
{X|Y|XY}*FRs* = RSQRTS *Rm* {(NF)} ;
{X|Y|XY}*FRsd* = RSQRTS *Rmd* {(NF)} ;
{X|Y|XY}*Rs* = MANT *FRm|FRmd* {(NF)} ;
{X|Y|XY}*Rs* = LOGB *FRm|FRmd* {({S}{NF})} ;[3]
{X|Y|XY}*FRs = Rm + Rn, FRa = Rm - Rn* { (NF) } ; *(двойная операция)*
{X|Y|XY}*FRsd = Rmd + Rnd,* F*Rad = Rmd - Rnd* { (NF) } ; *(двойная операция)*

[1] Допустимые опции: (): округление, (T): с усечением.
[2] Допустимые опции: (): округление, (T): с усечением (только MTN).
[3] Допустимые опции: (): без насыщения, (S): с насыщением.

A.2 Краткий справочник по CLU

Примеры использования этих команд можно найти в подразделе «Примеры использования CLU».

Листинг A-5 – Команды CLU

```
{X|Y|XY}{S}TR sd = TMAX (TRmd + Rmq_h, TRnd + Rmq_l) {(NF)} ;
{X|Y|XY}{S}TRsd = TMAX(T Rmd - Rmq_h, TRnd - Rmq_l ){(NF)} ;
{X|Y|XY}{S}Rs= TMAX ( TRm, TRn) {(NF)} ;
{X|Y|XY}{ S} TR sd = MAX(T Rmd + Rmq_h, TRnd + Rmq_l) {(NF)} ;
{X|Y|XY}{S}TR sd = MAX(T Rmd - Rmq_h, TRnd - Rmq_l) {(NF)} ;
/* здесь Rmq_h должен быть старшей частью квадрорегистра, а Rmq_l должен быть
младшей частью ТОГО ЖЕ квадрорегистра */
{X|Y|XY} Rs = T Rm ;
{X|Y|XY} Rsd = T Rmd ;
{X|Y|XY} Rsq = T Rmq ;
{X|Y|XY} Rs = TH Rm ;
{X|Y|XY} Rsd = TH Rmd ;
{X|Y|XY} Rsq = TH Rmq ;
{X|Y|XY} Rs = CMCTL ;
/* Синтаксис команд загрузки регистров TR,THR и CMCTL можно найти в разделе
«Краткий справочник по сдвиговому устройству» */
{X |Y|XY} TRs += DESPREAD (Rmq, THRd) ;
{X |Y|XY} Rs = TRs, TRs = DESPREAD (Rmq, THRd) { (NF) } ; (двойная операция)
{X |Y|XY} Rsd = TRsd, TRsd = DESPREAD (Rmq, THRd){ (NF)};(двойнаяоперация)
{X |Y|XY} TRsa = XCORRS (Rmq, THRnq) { (CUT<Imm> | R) } { (CLR)}{(EXT)){(NF) };
{X |Y|XY) Rsq = TRbq, TRsa = XCORRS (Rmq, THRnq)
{ (CUT <Imm> |R) } { (CLR) } { (EXT) } { (NF) } ; (двойная операция)
/* здесь TRsa = TR15:0 или TR31:16 */
{X|Y|XY}{S}TRsq = ACS (TRmd, TRnd, Rm) {(TMAX)}{(NF));
{X|Y|XY}Rsq = TRaq, {S}TRsq = ACS (TRmd, TRnd, Rm)
{(TMAX)} {(NF)} ; (двойная операция)
```

A.3 Краткий справочник по умножителю

Примеры использования этих команд можно найти в разделе «Примеры использования умножителя».

Листинг A-6 – Команды умножения 32-битных операндов с фиксированной точкой

```
{X|Y|XY}Rs = Rm * Rn {({U|nU}{I|T}{S}{NF})} ;[1]
{X|Y|XY}Rsd = Rm * Rn {({U|nU}{I}{NF})} ;
{X|Y|XY}MRa += Rm * Rn {({U}{I}{C|CR}(NF))} ;[2]
{X|Y|XY}MRa -= Rm * Rn {({I}{C|CR}{NF})} ;
{X|Y|XY}Rs = MRa, MRa += Rm * Rn {({U} {I} {C|CR}{NF})} ; (двойная операция)
{X|Y|XY}Rsd = MRa, MRa += Rm * Rn {({U}{I}{C}{ NF})} ; (двойная операция)
/* здесь MRa либо MR1:0, либо MR3:2 */
```

[1] Допустимые опции: (): дробное, знаковое, без насыщения, (S): с насыщением, знаковое, (SU): с насыщением, беззнаковое.

[2] Допустимые опции: (): знаковое, округление к ближайшему чётному, (T): знаковое, с усечением, (U): беззнаковое, округление к ближайшему чётному, (TU): беззнаковое, с усечением.

Листинг А-7 – Команды квадроумножения 16-битных операндов с фиксированной точкой

```
{X|Y|XY}Rsd = Rmd * Rnd {({U} {I|T}{S}{NF})} ;
{X|Y|XY}Rsq = Rmd * Rnd {({U}{I}(NF})} ;
{X|Y|XY}MR3:0 += Rmd * Rnd {({U}{I}{C|CR}{NF})} ;
{X|Y|XY}MRb += Rmd * Rnd {({U}{I}{C}{NF})} ;
{X|Y|XY}Rsd = MRb,MR3:0 += Rmd * Rnd {({I} {C|CR}{NF})}; (двойная операция)
{X|Y|XY}Rsd = MRb,MRb += Rmd * Rnd {({I}{C}{NF})}; (двойная операция)
/* здесь MRb либо MR1:0, либо MR3:2 */
```

Листинг А-8 – Команды комплексного умножения 16-битных операндов с фиксированной точкой

```
{X|Y|XY}MRa += Rm** Rn {({I}{C|CR}{J}{NF})} ;
{X|Y|XY}MRa -= Rm** Rn {({I}{C|CR}{J}{NF})} ;
{X|Y|XY}Rs = MRa, MRa += Rm ** Rn {({I}{C|CR}{J}{NF})} ; (двойная операция)
{X|Y|XY}Rsd = MRa, MRa += Rm ** Rn {({I}{C}{J}{NF})} ; (двойная операция)
/* здесь MRa либо MR1:0, либо MR3:2 */
```

Листинг А-9 – Команды умножения 32- и 40-битных операндов с плавающей точкой

```
{X|Y|XY}FRs = Rm * Rn {({T}{NF})} ;
{X|Y|XY}FRsd = Rmd * Rnd {({T}{NF})} ;
```

Листинг А-10 – Команды загрузки регистров умножителя

```
{X|Y|XY}{S|L}MRa = Rmd {({SE|ZE}{NF})} ;
{X|Y|XY}MR4 = Rm {(NF)} ;
{X|Y|XY}{S}Rsd = MRa {({U}{S}{NF})} ;
{X|Y|XY}Rsq = MR3:0 {({U}{S}{NF})} ;
{X|Y|XY}Rs = MR4 ;
/* здесь MRa либо MR1:0, либо MR3:2 */
{X|Y|XY}Rsd = SMRb {({U}{NF})} ; /* извлечь 2 коротких слова */
{X|Y|XY}LRsd = MRb {({U}{NF})} ; /* извлечь 1 обычные слово */
/* здесь MRb равен MR0, MR1, MR2 или MR3 */
{X|Y|XY}Rsq = SMRa {({U}{NF})} ; /* извлечь 4 коротких слова */
{X|Y|XY}LRsq = MRa {({U}{NF})} ; /* извлечь 2 обычных слова */
{X|Y|XY}QRsq = LMRa      {({U}{NF})} ; /* извлечь 1 длинное слово из MRa*/
/* здесь MRa либо MR1:0, либо MR3:2 */
{X|Y|XY}Rs = COMPACT MRa {({U}{I}{S}{NF})} ;
{X|Y|XY}SRsd = COMPACT MR3:0 {({U}{I}{S}{NF})} ;
/* здесь MRa либо MRI:0, либо MR3:2 */
```

A.4 Краткий справочник по сдвиговому устройству

Примеры использования этих команд можно найти в подразделе «Примеры использования сдвигового устройства».

Листинг A-11 – Команды сдвигового устройства

```
{X|Y|XY}{B|S}Rs = LSHIFT|ASHIFT Rm BY Rn|<Imm> {(NF)} ;[1, 2]
{X|Y|XY}{B|S|L}Rsd = LSHIFT | ASHIFT Rmd BY Rn|<Imm> { (NF) } ;[1, 2]
{X|Y|XY}Rs = ROT Rm BY Rn|<Imm6> {(NF)}
{X|Y|XY}{L}Rsd = ROT Rmd BY Rn|<Imm> {(NF)} ;[1, 2]
{X|Y|XY}Rs = FEXT Rm BY Rn|Rnd {({SE}{MF})} ;[3]
{X|Y|XY}LRsd = FEXT Rmd BY Rn|Rnd {({SE}{NF})} ;[3]
{X|Y|XY}Rs += FDEP Rm BY Rn|Rnd {({SE|ZF}{NF})} ;[3]
{X|Y|XY}LRsd += FDEP Rmd BY Rn|Rnd {({SE|ZF}{NF})} ;[3]
{X|Y|XY}Rs += MASK Rm BY Rn {(NF)} ;
{X|Y|XY}LRsd += MASK Rmd BY Rnd {(NF)} ;
{X|Y|XY}Rsd = GETBITS Rmq BY Rnd {({SE}{NF})} ;
{X|Y|XY}Rsd += PUTBITS Rmd BY Rnd {(NF)} ;
{X|Y|XY}BITEST Rm BY Rn|<Imm5> ;
{X|Y|XY}BITEST Rmd BY Rn|<Imm6> ;
{X|Y|XY}Rs = BCLR|BSET|BTGL Rm BY Rn|<Imm5> {(NF)} ;
{X|Y|XY}Rsd = BCLR|BSET|BTGL Rmd BY Rn|<Imm6> {(NF)} ;
{X|Y|XY}Rs = LD0|LD1 Rm|Rmd {(NF)} ;
{X|Y|XY}Rs = EXP Rm|Rmd {(NF)} ;
{X|Y}STAT = Rm ;
{X|Y}STAT_L = Rm ;
{X|Y}Rs = {X|Y}STAT ;
{X|Y|XY}BKFPT Rmd, Rnd ;
{X|Y|XY}Rsd = BFOTMP {(NF)} ;
{X|Y|XY}BFOTMP = Rmd {(NF)} ;
{X|Y|XY}TRs = Rm ;
{X|Y|XY}TRsd = Rmd ;
{X|Y|XY}TRsq = Rmq ;
{X|Y|XY}THRs = Rm ;
{X|Y|XY}THRsd = Rmd {(I)} ;
{X|Y|XY}THRsq = Rmq ;
{X|Y|XY}CMCTL = Rm ;
/* Синтаксис команд пересылки регистров TR, THR и CMCTL см. в подразделе «Краткий справочник по CLU».
```

A.5 Краткий справочник по IALU

Примеры использования этих команд можно найти в разделе «Примеры программирования IALU».

Листинг A-12 – Арифметические, логические и функциональные команды IALU

```
Js = Jm +|- Jn|<Imm8>|<Imm32> {({CJMP|CB|BR}{NF})} ;
JB0|JB1|JB2|JB3|JL0|JL1|JL2|JL3 = Jm +|-Jn|<Imm8>|<Imm32> {(NF)};
Js = Jm + Jn|<Imm8>|<Imm32> + JC {(NF)} ;
```

[1] Размер данных в регистре *Rn,* определяющих величину сдвига, зависит от типа выходного операнда: для однобайтного - 5 бит, для короткого - 6, для обычного - 7, для длинного - 8.

[2] Размер данных *Imm,* определяющих величину сдвига, зависит от типа выходного операнда: для однобайтного - 4 бита, для короткого - 5, для обычного - 6, для данного - 7.

[3] Расположение полей Pos8 и Len7 зависит от имени регистра - *Rn* или *Rnd.*

Js = *Jm* - Jn|<*Imm8*>|<*Imm32*> + JC - 1 {(NF)} ;
Js = (Jm +|- *Jn*|<*Imm8*>|<*Imm32*>)/2 {(NF)} ; COMP(Jm, *Jn*\<*Imm8*>|<*Imm32*>){(U)} ;
Js = MAX|MIN (Jm, *Jn*|<*Imm8*>|<*Imm32*>){(NF)} ; Js = ABS Jm {(NF)} ;
Js = *Jm* OR|AND|XOR|AND NOT *Jn*|<*Imm8*>|<*Imm32*> {(NF)} ;
Js = NOT *Jm* {(NF)} ;
Js = ASHIFTR|LSHIFTR *Jm* {(NF)} ;
Js = ROTR|ROTL *Jm* {(NF)} ;
Ks = *Km* +|- *Kn*|<*Imm8*>|<*Imm32*> {({CJMP|CB|BR}{NF})} ;
KB0|KB1|KB2|KB3|KL0|KL1|KL2|KL3 = *Km* +|-*Kn*|<*Imm8*>|<*Imm32*> {(NF)} ;
Ks = *Km* + *Kn*|<*Imm8*>|<*Imm32*> + KC {(NF)} ;
Ks = *Km* - *Kn* | <*Imm8*>|<*Imm32*> + KC - 1 {(NF)} ;
Ks = *(Km* + |- *Kn* |<*Imm8*>|<*Imm32*>)/2 {(NF)} ;
COMP(Km, *Kn*|<*Imm8*>|<*Imm32>)* {(U)} ;
Ks = MAX|MIN *(Km, Kn* |<*Imm8*>|<*Imm32>)* {(NF)} ;
Ks = ABS *Km* {(NF)} ;
Ks = *Km* OR|AND|XOR|AND NOT *Kn* |*Imm8*>|<*Imm32*> {(NF)} ;
Ks = NOT *Km* {(NF)} ;
Ks = ASHIFTR|LSHIFTR *Km* {(NF)} ;
Ks = ROTR|ROTL *Km* {(NF)} ;

Листинг А-13 – Команды IALU загрузки Ureg-регистров (адресация данных)

Ureg_s = {CB|BR}[Jm +|+= Jn |<Imm8>|<Imm32>] ;
Ureg_sd = {CB|BR} L [Jm +|+= Jn|<Imm8>|<Imm32>];
Ureg_sq = {CB|BR} Q [Jm +|+= Jn|<Imm8>|<Imm32>] ;
Ureg_s = {CB|BR}[Km +|+= Kn|<Imm8>|<Imm32>] ;
Ureg_sd = {CB|BR} L [Km +|+= Kn|<Imm8>|<Imm32>];
Ureg_sq = {CB|BR} Q [Km +|+= Kn |<Imm8>|<Imm32>] ;
/* суффикс при *Ureg* означает: _s=обычный, _sd=двойной, _sq=квадрорегистр */

Листинг А-14 – Команды IALU загрузки Dreg-регистров (адресация данных и операции с DAB)

{X|Y|XY}Rs = {CB|BR} [Jm += Jn |<Imm8>|<Imm32>] ;
{X|Y|XY}Rsd = {CB|BR} L [Jm += Jn\<Imm8>|<Imm32>] ;
{XY|YX}Rs = {CB|BR) L [Jm += Jn |<Imm8>|<ImmJ2>] ;
{X|Y|XY}Rsq = {CB|BR|DAB|SDAB} Q [Jm += Jn|<Imm8>|<Imm32>] ;
{XY|YX}Rsd = {CB|BR} Q [Jm += Jn |<Imm8>|<Imm32>];
{XY}Rsq = XDAB Q [Jm += Jn|<Imm8>|<Imm32>] ;
{XY}Rsq = XSDAB Q [Jm += Jn |<Imm8>|<Imm32>] ;
{XY}Rsq = YDAB Q [Jm += Jn|<Imm8>|<Imm32>] ;
{XY}Rsq = YSDAB Q [Jm += Jn |<Imm8>|<Imm32>] ;
{X|Y|XY}Rs = {CB|BR) [Km += Kn|<Imm8>\<Imm32>] ;
{X|Y|XY}Rsd = {CB|BR) L [Km += Kn|<Imm8>|<Imm32>] ;
{XY|YX}Rs = {CB|BR} L [Km += Kn|<Imm8>|<Imm32>] ;
{X|Y|XY}Rsq = {CB|BR|DAB|SDAB} Q [Km += Kn|<Imm8>|<Imm32>] ;
{XY|YX}Rsd = {CB|BR} Q [Km += Kn |<Imm8>|<Imm32>] ;
{XY}Rsq = XDAB Q [Km += Kn | < Imm8> | < Imm32>] ;
{XY}Rsq = XSDAB Q [Km += Kn | <Imm8> | < Imm32>] ;
{XY}Rsq = YDAB Q [Km += Kn |<Imm8>|<Imm32>] ;
{XY}Rsq = YSDAB Q [Km += Kn|<Imm8>|<Imm32>] ;
/* суффикс при R означает: _s=обычный, _sd=двойной, _sq=квадрорегистр */
/* m должно быть равно 0,1,2 или 3 для бит-реверсивных и кольцевых буферов */

Листинг А-15 – Команды IALU сохранения Ureg-регистров (адресация данных)

```
      [Jm +|+= Jn|<Imm8>|<Imm32>] = Ureg_s ;
L     [Jm +|+= Jn|<Imm8>|<Imm32>] = Ureg_sd ;
Q     [Jm +|+= Jn|<Imm8>|<Imm32>] = Ureg_sq ;
      [Km +|+= Kn |<Imm8>|<Imm32>] = Ureg_s ;
L     [Km +|+= Kn |<Imm8>|<Imm32>] = Ureg_sd ;
Q     [Km +|+= Kn |<Imm8>|<Imm32>] = Ureg_sq ;
```

Листинг А-16 – Команды IALU сохранения Dreg-регистров (адресация данных)

```
{CB | BR}      [Jm+= Jn |<Imm8> |<Imm32>]={X|Y}Rs ;
{CB | BR}  L   [Jm+= Jn | < Imm8>|<Imm32>]= {X|Y}Rsd ;
{CB | BR}  L   [Jm+= Jn | <Imm8>|<Imm32>]= {XY|YX}Rs ;
{CB | BR}  Q   [Jm+= Jn | <Imm8>|< Imm32>]={X|Y}Rsq ;
{CB | BR)  Q   [Jm+= Jn |<Imm8> | <Imm32>]={XY|YX}Rsd ;
{CB | BR}      [Km+= Kn|<Imm8>|<Imm32>]= {X|Y}Rs ;
{CB | BR}  L   [Km+= Kn|<Imm8>|<Imm32>] = {X|Y}Rsd ;
{CB | BR}  L   [Km+= Kn|<Imm8>|<Imm32>] = {XY|YX}Rs ;
{CB | BR}  Q   [Km+= Kn|<Imm8>|<Imm32>] = {X|Y}Rsq ;
{CB | BR}  Q   [Km+= Kn|<Imm8>|<Imm32>]= {XY|YX}Rsd ;
/* суффикс при R означает: _s=обычный, _sd=двойной, _sq=квадрорегистр */
/* m должно быть равно 0,1,2 или 3 для бит-реверсивных и кольцевых буферов */
```

Листинг А-17 – Команды IALU пересылки универсальных регистров

```
Ureg_s = <Imm15>|<Imm32> ;
Ureg_s = Ureg_m ;
Ureg_sd = Ureg_md ;
Ureg_sq = Ureg_mq ;
```

А.6 Краткий справочник по устройству управления

Примеры использования этих команд можно найти в разделе «Примеры программирования устройства управления».

Листинг А-18 – Команды устройства управления

{IF *Condition,}* JUMP|CALL <*Label*> {(NP)} {(ABS)} ;
{IF *Condition,}* CJMP|CJMP_CALL {(NP)} {(ABS)} ;
{IF *Condition,}* RETI|RTI {(NP)} {(ABS)};
{IF *Condition,}* RDS ; IF *Condition;*
DO, *команда;* DO, *команда;* DO, *команда;;*
/* Такой синтаксис позволяет подчинить одному предикату до трёх условных команд. Пропуск DO перед командой делает её безусловной. */
IF *Condition,* JUMP|CALL|CJMP|CJMP_CALL ; ELSE, *команда;* ELSE, *команда;* ELSE, *команда;;*
/* Такой синтаксис позволяет подчинить одному предикату до трёх условных команд. Пропуск ELSE перед командой делает её безусловной. */[1]

[1] В командных строках, начинающихся с команды устройства управления (таких, как условная команда, либо условная или безусловная команда устройства управления) команды могут использовать опцию NF (не-обновлять-флаги) тогда и только тогда, когда команды во ВСЕХ полях этой строки ее используют. Впрочем, в этой строке могут присутствовать и команды, опцию NF не поддерживающие (примером может служить команда COMP).

{X|Y|XY}SF1 | SF0 = *Выч. предикат* ;
{X|Y|XY}SF1 | SF0 += AND|OR|XOR *Выч.предикат* ;
ISF1|ISF0 = *IALU-предикат Выч.предикат* | *УУ-предикат* ;
ISF1|ISF0 += AND|OR|XOR *IALU-предикат Выч. предикат* | *УУ-предикат* ;
IDLE ;
BTBEN ; BTBDIS ;
BTBINV ;
TRAP (*<Imm5>)* ;;
EMUTRAP ;;
NOP ;

Приложение Б
Определения регистров и битов

При написании программ для DSP часто бывает необходимо устанавливать, сбрасывать или проверять биты регистров процессора. Эти битовые операции могут быть выполнены не только с использованием ссылок на позиции битов внутри регистра или (для ряда операций) на адрес регистра, задаваемых шестнадцатеричными числами, но и, что удобнее, с пользованием символических ссылок, соответствующих названиям битов или регистров. Ради удобства и согласованности имен используется заголовочный файл, который содержит определения битов и регистров процессора. Дополнительную информацию о регистрах процессора см. в Спецификации процессора, глава «Регистры и память».

Листинг В-1 – DEFTS201.H - файл определений регистров и битов

```
#if !defined(_DEFTS201_H_)
#define _DEFTS201_H_
//
// Макросы битовых масок
//
#if !defined(MAKE_BITMASK_)
#define MAKE_BITMASK_(x_) (l<<(x_)) // Получить битовую маску из позиции бита
#endif
#if !defined(MAKE_LL_BITMASK_)
#define MAKE_LL_BITMASK_(x_) (lLL<<(x_)) // Получить битовую маску из позиции
бита (используется только в C)
#endif
//*
//* Регистры, не отображенные в память *
//*
//* XSTAT *

// Позиции битов
#define XSTAT_AZ_P ( 0)
#define XSTAT_AN_P ( 1)
#define XSTAT_AV_P ( 2)
#define XSTAT_AC_P ( 3)
#define XSTAT_MZ_P ( 4)
#define XSTAT_MN_P ( 5)
#define XSTAT_MV_P ( 6)
#define XSTAT_MU_P ( 7)
#define XSTAT_SZ_P ( 8)
#define XSTAT_SN_P ( 9)
#define XSTAT_BF_P (10)
#define XSTAT_AI_P (12)
#define XSTAT_MI_P (13)
#define XSTAT_UEN_P (20)
#define XSTAT_OEN_P (21)
#define XSTAT_IVEN_P (22)
#define XSTAT_AUS_P (24)
```

```
#define XSTAT_AVS_P (25)
#define XSTAT_AOS_P (26)
#define XSTAT_AIS_P (27)
#define XSTAT_MUS_P (28)
#define XSTAT_MVS_P (29)
#define XSTAT_MOS_P (30)
#define XSTAT_MIS_P (31)

// Битовые маски
#define XSTAT_AZ MAKE_BITMASK_(XSTAT_AZ_P)
#define XSTAT_AN MAKE_BITMASK_(XSTAT_AN_P)
#define XSTAT_AV MAKE_BITMASK_(XSTAT_AV_P)
#define XSTAT_AC MAKE_BITMASK_(XSTAT_AC_P)
#define XSTAT_MZ MAKE_BITMASK_(XSTAT_MZ_P)
#define XSTAT_MN MAKE_BITMASK_(XSTAT_MN_P)
#define XSTAT_MV MAKE_BITMASK_(XSTAT_MV_P)
#define XSTAT_MU MAKE_BITMASK_(XSTAT_MU_P)
#define XSTAT_SZ MAKE_BITMASK_(XSTAT_SZ_P)
#define XSTAT_SN MAKE_BITMASK_(XSTAT_SN_P)
#define XSTAT_BF MAKE_BITMASK_(XSTAT_BF_P)
#define XSTAT_AI MAKE_BITMASK_(XSTAT_AI_P)
#define XSTAT_MI MAKE_BITMASK_(XSTAT_MI_P)
#define XSTAT_UEN MAKE_BITMASK_(XSTAT_UEN_P)
#define XSTAT_OEN MAKE_BITMASK_(XSTAT_OEN_P)
#define XSTAT_IVEN MAKE_BITMASK_(XSTAT_IVEN_P)
#define XSTAT_AUS MAKE_BITMASK_(XSTAT_AUS_P)
#define XSTAT_AVS MAKE_BITMASK_(XSTAT_AVS_P)
#define XSTAT_AOS MAKE_BITMASK_(XSTAT_AOS_P)
#define XSTAT_AIS MAKE_BITMASK_(XSTAT_AIS_P)
#define XSTAT_MUS MAKE_BITMASK_(XSTAT_MUS_P)
#define XSTAT_MVS MAKE_BITMASK_(XSTAT_MVS_P)
#define XSTAT_MOS MAKE_BITMASK_(XSTAT_MOS_P)
#define XSTAT_MIS MAKE_BITMASK_(XSTAT_MIS_P)
//* YSTAT *

// Позиции битов
#define YSTAT_AZ_P ( 0)
#define YSTAT_AN_P ( 1)
#define YSTAT_AV_P ( 2)
#define YSTAT_AC_P ( 3)
#define YSTAT_MZ_P ( 4)
#define YSTAT_MN_P ( 5)
#define YSTAT_MV_P ( 6)
#define YSTAT_MU_P ( 7)
#define YSTAT_SZ_P ( 8)
#define YSTAT_SN_P ( 9)
#define YSTAT_BF_P (10)
#define YSTAT_AI_P (12)
#define YSTAT_MI_P (13)
```

```
#define YSTAT_UEN_P (20)
#define YSTAT_OEN_P (21)
#define YSTAT_IVEN_P (22)
#define YSTAT_AUS_P (24)
#define YSTAT_AVS_P (25)
#define YSTAT_AOS_P (26)
#define YSTAT_AIS_P (27)
#define YSTAT_MUS_P (28)
#define YSTAT_MVS_P (29)
#define YSTAT_MOS_P (30)
#define YSTAT_MIS_P (31)

// Битовые маски
#define YSTAT_AZ MAKE_BITMASK_(YSTAT_AZ_P)
#define YSTAT_AN MAKE_BITMASK_(YSTAT_AN_P)
#define YSTAT_AV MAKE_BITMASK_(YSTAT_AV_P)
#define YSTAT_AC MAKE_BITMASK_(YSTAT_AC_P)
#define YSTAT_MZ MAKE_BITMASK_(YSTAT_MZ_P)
#define YSTAT_MN MAKE_BITMASK_(YSTAT_MN_P)
#define YSTAT_MV MAKE_BITMASK_(YSTAT_MV_P)
#define YSTAT_MU MAKE_BITMASK_(YSTAT_MU_P)
#define YSTAT_SZ MAKE_BITMASK_(YSTAT_SZ_P)
#define YSTAT_SN MAKE_BITMASK_(YSTAT_SN_P)
#define YSTAT_BF MAKE_BITMASK_(YSTAT_BF_P)
#define YSTAT_AI MAKE_BITMASK_(YSTAT_AI_P)
#define YSTAT_MI MAKE_BITMASK_(YSTAT_MI_P)
#define YSTAT_UEN MAKE_BITMASK_(YSTAT_UEN_P)
#define YSTAT_OEN MAKE_BITMASK_(YSTAT_OEN_P)
#define YSTAT_IVEN MAKE_BITMASK_(YSTAT_IVEN_P)
#define YSTAT_AUS MAKE_BITMASK_(YSTAT_AUS_P)
#define YSTAT_AVS MAKE_BITMASK_(YSTAT_AVS_P)
#define YSTAT_AOS MAKE_BITMASK_(YSTAT_AOS_P)
#define YSTAT_AIS MAKE_BITMASK_(YSTAT_AIS_P)
#define YSTAT_MUS MAKE_BITMASK_(YSTAT_MUS_P)
#define YSTAT_MVS MAKE_BITMASK_(YSTAT_MVS_P)
#define YSTAT_MOS MAKE_BITMASK_(YSTAT_MOS_P)
#define YSTAT_MIS MAKE_BITMASK_(YSTAT_MIS_P)
//*
//* Регистры, отображенные в память *
//*
//* Выч. блок X *
#define XR0_LOC (0x1E0000)
#define XR1_LOC (0x1E0001)
#define XR2_LOC (0x1E0002)
#define XR3_LOC (0x1E0003)
#define XR4_LOC (0x1E0004)
#define XR5_LOC (0x1E0005)
#define XR6_LOC (0x1E0006)
#define XR7_LOC (0x1E0007)
```

```
#define XR8_LOC (0x1E0008)
#define XR9_LOC (0x1E0009)
#define XR10_LOC (0x1E000A)
#define XR11_LOC (0x1E000B)
#define XR12_LOC (0x1E000C)
#define XR13_LOC (0x1E000D)
#define XR14_LOC (0x1E000E)
#define XR15_LOC (0x1E000F)
#define XR16_LOC (0x1E0010)
#define XR17_LOC (0x1E0011)
#define XR18_LOC (0x1E0012)
#define XR19_LOC (0x1E0013)
#define XR20_LOC (0x1E0014)
#define XR21_LOC (0x1E0015)
#define XR22_LOC (0x1E0016)
#define XR23_LOC (0x1E0017)
#define XR24_LOC (0x1E0018)
#define XR25_LOC (0x1E0019)
#define XR26_LOC (0x1E001A)
#define XR27_LOC (0x1E001B)
#define XR28_LOC (0x1E001C)
#define XR29_LOC (0x1E001D)
#define XR30_LOC (0x1E001E)
#define XR31_LOC (0x1E001F)

//* Выч. блок Y *
#define YR0_LOC (0x1E0040)
#define YR1_LOC (0x1E0041)
#define YR2_LOC (0x1E0042)
#define YR3_LOC (0x1E0043)
#define YR4_LOC (0x1E0044)
#define YR5_LOC (0x1E0045)
#define YR6_LOC (0x1E0046)
#define YR7_LOC (0x1E0047)
#define YR8_LOC (0x1E0048)
#define YR9_LOC (0x1E0049)
#define YR10_LOC (0x1E004A)
#define YR11_LOC (0x1E004B)
#define YR12_LOC (0x1E004C)
#define YR13_LOC (0x1E004D)
#define YR14_LOC (0x1E004E)
#define YR15_LOC (0x1E004F)
#define YR16_LOC (0x1E0050)
#define YR17_LOC (0x1E0051)
#define YR18_LOC (0x1E0052)
#define YR19_LOC (0x1E0053)
#define YR20_LOC (0x1E0054)
#define YR21_LOC (0x1E0055)
#define YR22_LOC (0x1E0056)
#define YR23_LOC (0x1E0057)
#define YR24 _LOC (0x1E0058)
```

```
#define YR25_LOC (0x1E0059)
#define YR26_LOC (0x1E005A)
#define YR27_LOC (0x1E005B)
#define YR28_LOC (0x1E005C)
#define YR29_LOC (0x1E005D)
#define YR30_LOC (0x1E005E)
#define YR31_LOC (0x1E005F)

// Блок XY для совмещенных операций
#define XYR0_LOC (0x1E0080)
#define XYR1_LOC (0x1E0081)
#define XYR2_LOC (0x1E0082)
#define XYR3_LOC (0x1E0083)
#define XYR4_LOC (0x1E0084)
#define XYR5_LOC (0x1E0085)
#define XYR6_LOC (0x1E0086)
#define XYR7_LOC (0x1E0087)
#define XYR8_LOC (0x1E0088)
#define XYR9_LOC (0x1E0089)
#define XYR10_LOC (0x1E008A)
#define XYR11_LOC (0x1E008B)
#define XYR12_LOC (0x1E008C)
#define XYR13_LOC (0x1E008D)
#define XYR14_LOC (0x1E008E)
#define XYR15_LOC (0x1E008F)
#define XYR16_LOC (0x1E0090)
#define XYR17_LOC (0x1E0091)
#define XYR18_LOC (0x1E0092)
#define XYR19_LOC (0x1E0093)
#define XYR20_LOC (0x1E0094)
#define XYR21_LOC (0x1E0095)
#define XYR22_LOC (0x1E0096)
#define XYR23_LOC (0x1E0097)
#define XYR24_LOC (0x1E0098)
#define XYR25_LOC (0x1E0099)
#define XYR26_LOC (0x1E009A)
#define XYR27_LOC (0x1E009B)
#define XYR28_LOC (0x1E009C)
#define XYR29_LOC (0x1E009D)
#define XYR30_LOC (0x1E009E)
#define XYR31_LOC (0x1E009F)

// Блок YX для совмещенных операций
#define YXR0_LOC (0x1E00C0)
#define YXR1_LOC (0x1E00C1)
#define YXR2_LOC (0x1E00C2)
#define YXR3_LOC (0x1E00C3)
#define YXR4_LOC (0x1E00C4)
#define YXR5_LOC (0x1E00C5)
```

```
#define YXR6_LOC (0x1E00C6)
#define YXR7_LOC (0x1E00C7)
#define YXR8_LOC (0x1E00C8)
#define YXR9_LOC (0x1E00C9)
#define YXR10_LOC (0x1E00CA)
#define YXR11_LOC (0x1E00CB)
#define YXR12_LOC (0x1E00CC)
#define YXR13_LOC (0x1E00CD)
#define YXR14_LOC (0x1E00CE)
#define YXR15_LOC (0x1E00CF)
#define YXR16_LOC (0x1E00D0)
#define YXR17_LOC (0x1E00D1)
#define YXR18_LOC (0x1E00D2)
#define YXR19_LOC (0x1E00D3)
#define YXR20_LOC (0x1E00D4)
#define YXR21_LOC (0x1E00D5)
#define YXR22_LOC (0x1E00D6)
#define YXR23_LOC (0x1E00D7)
#define YXR24_LOC (0x1E00D8)
#define YXR25_LOC (0x1E00D9)
#define YXR26_LOC (0x1E00DA)
#define YXR27_LOC (0x1E00DB)
#define YXR28_LOC (0x1E00DC)
#define YXR29_LOC (0x1E00DD)
#define YXR30_LOC (0x1E00DE)
#define YXR31_LOC (0x1E00DF)

  //* Блок ХУ для широковещательных операций *
#define XYBR0_LOC (0x1E0100)
#define XYBR1_LOC (0x1E0101)
#define XYBR2_LOC (0x1E0102)
#define XYBR3_LOC (0x1E0103)
#define XYBR4_LOC (0x1E0104)
#define XYBR5_LOC (0x1E0105)
#define XYBR6_LOC (0x1E0106)
#define XYBR7_LOC (0x1E0107)
#define XYBR8_LOC (0x1E0108)
#define XYBR9_LOC (0x1E0109)
#define XYBR10_LOC (0x1E010A)
#define XYBR11_LOC (0x1E010B)
#define XYBR12_LOC (0x1E010C)
#define XYBR13_LOC (0x1E010D)
#define XYBR14_LOC (0x1E010E)
#define XYBR15_LOC (0x1E010F)
#define XYBR16_LOC (0x1E0110)
#define XYBR17_LOC (0x1E0111)
#define XYBR18_LOC (0x1E0112)
#define XYBR19_LOC (0x1E0113)
#define XYBR20_LOC (0x1E0114)
```

```
#define XYBR21_LOC (0x1E0115)
#define XYBR22_LOC (0x1E0116)
#define XYBR23_LOC (0x1E0117)
#define XYBR24_LOC (0x1E0118)
#define XYBR25_LOC (0x1E0119)
#define XYBR26_LOC (0x1E011A)
#define XYBR27_LOC (0x1E011B)
#define XYBR28_LOC (0x1E011C)
#define XYBR29_LOC (0x1E01ID)
#define XYBR30_LOC (0x1E01IE)
#define XYBR31_LOC (0x1E011F)


  //* JALU *
#define J0_LOC (0x1E0180)
#define J1_LOC (0x1E0181)
#define J2_LOC (0x1E0182)
#define J3_LOC (0x1E0183)
#define J4_LOC (0x1E0184)
#define J5_LOC (0x1E0185)
#define J6_LOC (0x1E0186)
#define J7_LOC (0x1E0187)
#define J8_LOC (0x1E0188)
#define J9_LOC (0x1E0189)
#define J10_LOC (0x1E018A)
#define J11_LOC (0x1E018B)
#define J12_LOC (0x1E018C)
#define J13_LOC (0x1E018D)
#define J14_LOC (0x1E018E)
#define J15_LOC (0X1E018F)
#define J16_LOC (0x1E0190)
#define J17_LOC (0x1E0191)
#define J18_LOC (0x1E0192)
#define J19_LOC (0x1E0193)
#define J20_LOC (0x1E0194)
#define J21_LOC (0x1E0195)
#define J22_LOC (0x1E0196)
#define J23_LOC (0x1E0197)
#define J24_LOC (0x1E0198)
#define J25_LOC (0x1E0199)
#define J26_LOC (0x1E019A)
#define J27_LOC (0x1E019B)
#define J28_LOC (0x1E019C)
#define J29_LOC (0x1E019D)
#define J30_LOC (0x1E019E)
#define J31_LOC (0x1E019F)

//* KALU *
#define K0_LOC (0x1E01A0)
#define K1_LOC (0x1E01A1)
```

```
#define K2_LOC (0x1E01A2)
#define K3_LOC (0x1E01A3)
#define K4_LOC (0x1E01A4)
#define K5_LOC (0x1E01A5)
#define K6_LOC (0x1E01A6)
#define K7_LOC (0x1E01A7)
#define K8_LOC (0x1E01A8)
#define K9_LOC (0x1E01A9)
#define K10_LOC (0x1E01AA)
#define K11_LOC (0x1E01AB)
#define K12_LOC (0x1E01AC)
#define K13_LOC (0x1E01AD)
#define K14_LOC (0x1E01AE)
#define K15_LOC (0x1E01AF)
#define K16_LOC (0x1E01B0)
#define K17 LOC (0x1E01B1)
#define K18_LOC (0x1E01B2)
#define K19_LOC (0x1E01B3)
#define K20_LOC (0x1E01B4)
#define K21_LOC (0x1E01B5)
#define K22_LOC (0x1E01B6)
#define K23_LOC (0x1E01B7)
#define K24_LOC (0x1E01B8)
#define K25_LOC (0x1E01B9)
#define K26_LOC (0x1E01BA)
#define K27_LOC (0x1E01BB)
#define K28_LOC (0x1E01BC)
#define K29_LOC (0x1E01BD)
#define K30_LOC (0x1E01BE)
#define K31_LOC (0x1E01BF)

//* Регистры JALU для адресации кольцевого буфера *
#define JB0_LOC (0x1E01C0)
#define JB1_LOC (0x1E01C1)
#define JB2_LOC (0x1E01C2)
#define JB3_LOC (0x1E01C3)
#define JL0_LOC (0x1E01C4)
#define JL1_LOC (0x1E01C5)
#define JL2_LOC (0x1E01C6)
#define JL3_LOC (0x1E01C7)

//* Регистры KALU для адресации кольцевого буфера *
#define KB0_LOC (0x1E01E0)
#define KB1_LOC (0x1E0IE1)
#define KB2_LOC (0x1E01E2)
#define KB3_LOC (0x1E01E3)
#define KL0_LOC (0x1E01E4)
#define KL1_LOC (0x1E01E5)
#define KL2_LOC (0x1E01E6)
```

```
#define KL3_LOC (0x1E01E7)

//* Регистры устройства управления *
#define CJMP_LOC (0x1E0340)
#define RETI_LOC (0x1E0342)
#define RETS_LOC (0x1E0344)
#define DBGE_LOC (0x1E0345)
#define LC0_LOC (0x1E0348)
#define LC1_LOC (0x1E0349)
#define IVSW_LOC (0x1E0350)

//* Регистры управления, установки и сброса флагов с определениями битов
#define FLAGREG_LOC (0x1E0354)
#define FLAGREGST_LOC (0x1E0355)
#define FLAGREGCL_LOC (0x1E0356)

// Позиции битов
#define FLAGREG_FLAGO_EN_P (0)
//направление дискретного сигнала FLAG0 определяется битом 0 во FLAGREG
#define FLAGREG_FLAG1_0UT_P (1) //-" "- FLAG1-" "- 1-" "-
#define FLAGREG_FLAG1_0UT_P (2) //-" "- FLAG2-" "- 2-" "-
#define FLAGREG_FLAG1_0UT_P (3) //-" "- FLAG3-" "- 3-" "-
#define FLAGREG_FLAG1_0UT_P (4) //-" "
// значение выходного сигнала FLAG0 определяется битом 4 во FLAGREG
#define FLAGREG_FLAG1_0UT_P (5) //-" "- FLAG1-" "- 5-" "-
#define FLAGREG_FLAG1_0UT_P (6) //-" "- FLAG2-" "- 6-" "-
#define FLAGREG_FLAG1_0UT_P (7) //-" "- FLAG3-" "- 7-" "-

// Битовые маски
#define FLAGREG_FLAG0_EN MAKE_BITMASK_(FLAGREG_FLAG0_EN_P)
#define FLAGREG_FLAG1_EN MAKE_BITMASK_(FLAGREG_FLAG1_EN_P)
#define FLAGREG_FLAG2_EN MAKE_BITMASK_(FLAGREG_FLAG2_EN_P)
#define FLAGREG_FLAG3_EN MAKE_BITMASK_(FLAGREG_FLAG3_EN_P)
#define FLAGREG_FLAG0_0UT MAKE_BITMASK_(FLAGREG_FLAG0_OUT_P)
#define FLAGREG_FLAG1_0UT MAKE_BITMASK_(FLAGREG_FLAG1_OUT_P)
#define FLAGREG_FLAG2_0UT MAKE_BITMASK_(FLAGREG_FLAG2_OUT_P)
#define FLAGREG_FLAG3_0UT MAKE_BITMASK_(FLAGREG_FLAG3_OUT_P)

//* SQCTL с определениями битов *
#define SQCTL_LOC (0x1E0358)
#define SQCTLST_LOC (0x1E0359)
#define SQCTLCL_LOC (0x1E035A)

// Позиции битов
#define SQCTL_GIE_P (2)
#define SQCTL_SW_P (3)
#define SQCTL_DBGDSBL_P (8)
#define SQCTL_NMOD_P (9)
#define SQCTL_TRCBEN_P (10)
#define SQCTL_TRCBEXEN_P (11)

// Битовые маски
#define SQCTL_GIE MAKE_BITMASK_(SQCTL_GIE_P)
```

```
#define SQCTL_SW MAKE_BITMASK_(SQCTL_SW_P)
#defineSQCTL_DBGDSBL MAKE_BITMASK_(SQCTL_DBGDSBL_P)
#define SQCTL_NMOD MAKE_BITMASK_(SQCTL_NMOD_P)
#define SQCTL_TRCBEN MAKE_BITMASK_(SQCTL_TRCBEN_P)
#define SQCTL_TRCBEXEN MAKE_BITMASK_(SQCTL_TRCBEXEN_P)

//* SQSTAT с определениями битов *
#define SQSTAT_LOC (0x1E035B)

// Позиции битов (масок)
#define SQSTAT_MODE_P (0)
#define SQSTAT_IDLE_P (2)
#define SQSTAT SPVCMD_P (3)
#define SQSTAT_EXCAUSE_P (8)
#define SQSTAT_EMCAUSE_P (12)
#define SQSTAT_FLG_P (16)
#define SQSTAT_GIE_P (20)
#define SQSTAT_SW_P (21)
#define SQSTAT_EMU_P (22)
#define SQSTAT_BTBEN_P (24)
#define SQSTAT_BTBLK_P (25)

// Битовые маски
#define SQSTAT_MODE (0x00000003)
#define SQSTAT_IDLE (0x00000004)
#define SQSTAT_SPVCMD (0x000000F8)
#define SQSTAT_EXCAUSE (0x00000F00)
#define SQSTAT_EMCAUSE (0x0000F000)
#define SQSTAT_FLG (0x000F0000)
//* SFREG с определениями битов *
#define SFREG_LOC (0x1E035C)

// Позиции битов
#define SFREG_GSCF0_P (0)
#define SFREG_GSCF1_P (1)
#define SFREG_XSCF0_P (2)
#define SFREG_XSCF1_P (3)
#define SFREG_YSCF0_P (4)
#define SFREG_YSCF1_P (5)

// Битовые маски
#define SFREG_GSCF0 MAKE_BITMASK_(SFREG_GSCF0_P)
#define SFREG_GSCF1 MAKE_BITMASK_(SFREG_GSCF1_P)
#define SFREG_XSCF0 MAKE_BITMASK_(SFREG_XSCF0_P)
#define SFREG_XSCF1 MAKE_BITMASK_(SFREG_XSCF1_P)
#define SFREG_YSCF0 MAKE_BITMASK_(SFREG_YSCF0_P)
#define SFREG_YSCF1 MAKE_BITMASK_(SFREG_YSCF1_P)

//* Регистры эмуляции I *

//* Регистры точек наблюдения *
#define WP0CTL_LOC (0x1E0360)
#define WP1CTL_LOC (0x1E0361)
#define WP2CTL_LOC (0x1E0362)
```

```
// Битовые маски
// OPMODE
#define WPCTL_DSBL (0x00000000)
#define WPCTL_ADDRESS (0x00000001)
#define WPCTL_RANGE (0x00000002)
#define WPCTL_NOTRANGE (0x00000003)

// EXTYPE
#define WPCTL_NOEXCEPT (0x0000000)
#define WPCTL_EXCEPT (0x00000004)
#define WPCTL_EMUTRAP (0x00000008)

// SSTP, WP0CTL
#define WPCTL_SSTP (0x00000010)
// R/W, WP1CTL and WP2CTL
#define WPCTL_READ (0x00000010)
#define WPCTL_WRITE (0x00000020)

// JK, WP1CTL
#define      WPCTL_JK_JBUS    (0x00000040)
#define      WPCTL_JK_KBUS    (0x00000080)
#define WPCTL_JK_BOTH (0x000000C0)
#define WP0STAT_LOC (0x1E0364)
#define WP1STAT_LOC (0x1E0365)
#define WP2STAT_LOC (0x1E0366)

// Позиции битов (масок)
#define WPSTAT_VALUE_P (0)
#define WPSTAT_EX P_(16)

// Битовые маски
#define WPSTAT_VALUE (0x0000FFFF)
#define WPSTAT_EX (0x00030000)
#define WP0L_LOC (0x1E0368)
#define WP0H_LOC (0x1E0369)
#define WP1L_LOC (0x1E036A)
#define WP1H_LOC (0x1E036B)
#define WP2L_LOC (0x1E036C)
#define WP2H_LOC (0x1E036D)
#define CCNT0_LOC (0x1E0370)
#define CCNT1_LOC (0x1E0371)
#define PRFM_LOC (0x1E0372)

// Битовые маски
// Неудовлетворенные запросы
#define PRFM_NGR_SEQ (0)
#define PRFM_NGR_JALU (1)
#define PRFM_NGR_KALU (2)
#define PRFM_NGR_SOC (3)

// Удовлетворенные запросы
#define PRFM_GR_SEQ (4)
#define PRFM_GR_JALU (5)
```

```
#define PRFM_GR_KALU (6)
#define PRFM_GR_SOC (7)

// Учитываемые модули
#define PRFM_MODULE_JALU (8)
#define PRFM_MODULE_KALU (9)
#define PRFM_MODULE_CBX (10)
#define PRFM_MODULE_CBY (11)
#define PRFM_MODULE_CTRL (12)
#define PRFM_SCYCLE (16)
#define PRFM_BTBPR (17)
#define PRFM_ISL (18)
#define PRFM_CCYCLE (19)
#define PRFM_SUMEN (31)
#define PRFCNT_LOC (0x1E0373)
#define TRCBMASK_LOC (0x1E0374)
#define TRCBPTR_LOC (0x1E0375)

//* TCB с определениями битов
#define DCS0_LOC (0x1F0000)
#define DCD0_LOC (0x1F0004)
#define DCS1_LOC (0x1F0008)
#define DCD1_LOC (0x1F000c)
#define DCS2_LOC (0x1F0010)
#define DCD2_LOC (0x1F0014)
#define DCS3_LOC (0x1F0018)
#define DCD3_LOC (0x1F001C)
#define DC4_LOC (0x1F0020)
#define DC5_LOC (0x1F0024)
#define DC6_LOC (0x1F0028)
#define DC7_LOC (0x1F002C)
#define DC8_LOC (0x1F0040)
#define DC9_LOC (0x1F0044)
#define DC10_LOC (0x1F0048)
#define DC11_LOC (0x1F004C)
#define DC12_LOC (0x1F0058)
#define DC13_LOC (0x1F005C)

// ТИПЫ (TY)
#define TCB_EPROM (0xC0000000)
#define TCB_FLYBY (0xA0000000)
#define TCB_EXTMEM (0x80000000)
#define TCB_INTMEM (0x40000000)
#define TCB_LINK (0x20000000)
#define TCB_DISABLE (0x00000000)

// ПРИОРИТЕТ (PR)
#define TCB_HPRIORITY (0x10000000)

// Двумерный DMA
#define TCB_TWODIM (0x08000000)

// ДЛИНА ОПЕРАНДА (LEN)
#define TCB_QUAD (0x06000000)
```

```
#define TCB_LONG (0x04000000)
#define TCB_NORMAL (0x02000000)

// ПРЕРЫВАНИЕ (INT)
#define TCB_INT (0x01000000)

// ЗАПРОС DMA (DRQ)
#define TCB_DMAR (0x00800000)

// ВКЛЮЧЕНИЕ ЗАЦЕПЛЕНИЯ (CHEN)
#define TCB_CHAIN (0x00400000)

// ЗАЦЕПЛЯЕМЫЙ КАНАЛ (CHTG)
#define TCB_DMA8DEST (0x00000000)
#define TCB_DMA9DEST (0x00080000)
#define TCB_DMA10DEST (0x00100000)
#define TCB_DMA11DEST (0x00180000)
#define TCB_DMA4DEST (0x00200000)
#define TCB_DMA5DEST (0x00280000)
#define TCB_DMA6DEST (0x00300000)
#define TCB_DMA7DEST (0x00380000)

//* Регистры управления DMA с определениями битов *
#define DCNT_LOC (0x1F0060)
#define DCNTST_LOC (0x1F0064)
#define DCNTCL_LOC (0x1F0068)

// Позиции битов
#define DCNT_DMA0_P (0)
#define DCNT_DMA1_P (1)
#define DCNT_DMA2_P (2)
#define DCNT_DMA3_P (3)
#define DCNT_DMA4_P (4)
#define DCNT_DMA5_P (5)
#define DCNT_DMA6_P (6)
#define DCNT_DMA7_P (7)
#define DCNT_DMA8_P (10)
#define DCNT_DMA9_P (11)
#define DCNT_DMA10_P (12)
#define DCNT_DMA11_P (13)
#define DCNT_DMA12_P (16)
#define DCNT_DMA13_P (17)

// Битовые маски
#define DCNT_DMA0 MAKE_BITMASK_(DCNT_DMA0_P)
#define DCNT_DMA1 MAKE_BITMASK_(DCNT_DMA1_P)
#define DCNT_DMA2 MAKE_BITMASK_(DCNT_DMA2_P)
#define DCNT_DMA3 MAKE_BITMASK_(DCNT_DMA3_P)
#define DCNT_DMA4 MAKE_BITMASK_(DCNT_DMA4_P)
#define DCNT_DMA5 MAKE_BITMASK_(DCNT_DMA5_P)
#define DCNT_DMA6 MAKE_BITMASK_(DCNT_DMA6_P)
#define DCNT_DMA7 MAKE_BITMASK_(DCNT_DMA7_P)
#define DCNT_DMA8 MAKE_BITMASK_(DCNT_DMA8_P)
#define DCNT_DMA9 MAKE_BITMASK_(DCNT_DMA9_P)
```

```
#define DCNT_DMA10 MAKE_BITMASK_ (DCNT_DMA10_P)
#define DCNT_DMA11 MAKE_BITMASK_ (DCNT_DMA11_P)
#define DCNT_DMA12 MAKE_BITMASK_ (DCNT_DMA12_P)
#define DCNT_DMA13 MAKE_BITMASK_ (DCNT_DMA13_P)

//* Регистры статуса DMA с определениями битов *
#define DSTATL_LOC (0x1F006C)
#define DSTATCL_LOC (0x1F0070)

// Битовые маски
#define DSTAT_IDLE (0x00000000)
#define DSTAT_ACT (0x00000001)
#define DSTAT_DONE (0x00000002)
#define DSTAT_ACT_ERR (0x00000004)
#define DSTAT_CFG_ERR (0x00000005)
#define DSTAT_ADD_ERR (0x00000007)

// Параметры полей статуса для извлечения командой FEXT
#define DSTATL0 (0x0003) // 0-ая позиция длины 3
#define DSTATL1 (0x0303) // 3-я позиция длины 3
#define DSTATL2 (0x0603) // 6-ая позиция длины 3
#define DSTATL3 (0x0903) // 9-ая позиция длины 3
#define DSTATL4 (0x0C03) // 12-ая позиция длины 3
#define DSTATL5 (0x0F03) // 15-ая позиция длины 3
#define DSTATL6 (0x1203) // 18-ая позиция длины 3
#define DSTATL7 (0x1503) // 21-ая позиция длины 3
#define DSTATH_LOC (0x1F006D)
#define DSTATCH_LOC (0x1F0071)
#define DSTATH8 (0x0003) // 0-ая позиция длины 3
#define DSTATH9 (0x0303) // 3-я позиция длины 3
#define DSTATH10 (0x0603) // 6-ая позиция длины 3
#define DSTATH11 (0x0903) // 9-ая позиция длины 3
#define DSTATH12 (0x1203) // 18-ая позиция длины 3
#define DSTATH13 (0x1503) // 21-ая позиция длины 3

//* Регистр SYSCON с масками битов *
#define SYSCON_LOC (0x1F0080)

// Битовые маски
#define SYSCON_MS0_IDLE (0x00000001)
#define SYSCON_MS0_WT0 (0x00000000)
#define SYSCON_MS0_WT1 (0x00000002)
#define SYSCON_MS0_WT2 (0x00000004)
#define SYSCON_MS0_WT3 (0x00000006)
#define SYSCON_MS0_PIPE1 (0x00000000)
#define SYSCON_MS0_PIPE2 (0x00000008)
#define SYSCON_MS0_PIPE3 (0x00000010)
#define SYSCON_MS0_PIPE4 (0x00000018)
#define SYSCON_MS0_SLOW (0x00000020)
#define SYSCON_MS1_IDLE (SYSCON_MS0_IDLE << 6)
#define SYSCON_MS1_WT0 (SYSCON_MS0_WT0<< 6)
#define SYSCON_MS1_WT1 (SYSCON_MS0_WT1<< 6)
#define SYSCON_MS1_WT2 (SYSCON_MS0_WT2<< 6)
#define SYSCON_MS1_WT3  (SYSCON_MS0_WT3 << 6)
```

```
#define SYSCON_MS1_PIPE1 (SYSCON_MS0_PIPE1<< 6)
#define SYSCON_MS1_PIPE2 (SYSCON_MS0_PIPE2<< 6)
#define SYSCON_MS1_PIPE3 (SYSCON_MS0_PIPE3<< 6)
#define SYSCON_MS1_PIPE4 (SYSCON_MS0_IPE4<< 6)
#define SYSCON_MS1_SLOW (SYSCON_MS0_SLOW << 6)
#define SYSCON_MS1_IDLE (SYSCON_MS0_IDLE << 12)
#define SYSCON_MSH_WT0 (SYSCON_MS0_WT0 << 12)
#define SYSCON_MSH_WT1 (SYSCON_MS0_WT1 << 12)
#define SYSCON_MSH_WT2 (SYSCON_MS0_WT2 << 12)
#define SYSCON_MSH_WT3 (SYSCON_MS0_WT3 << 12)
#define SYSCON_MSH_PIPE1 (SYSCON_MS0_PIPE1<< 12)
#define SYSCON_MSH_PIPE2 (SYSCON_MS0_PIPE2<< 12)
#define SYSCON_MSH_PIPE3 (SYSCON_MS0_PIPE3<< 12)
#define SYSCON_MSH_PIPE4 (SYSCON_MS0_IPE4<< 12)
#define SYSCON_MSH_SLOW (SYSCON_MS0_SLOW << 12)
#define SYSCON_MEM_WID64 (0x00080000)
#define SYSCON_MP_WID64 (0x00100000)
#define SYSCON_HOST_WID64 (0x00200000)

//* Регистр BUSLOCK *
#define BUSLOCK_LOC (0x1F0083)

//* Регистр SDRCON с масками битов *
#define SDRCON_LOC (0x1F0084)

// Битовые маски
#define SDRCON_ENBL (0x00000001)
#define SDRCON_CLAT1 (0x00000000)
#define SDRCON_CLAT2 (0x00000002)
#define SDRCON_CLAT3 (0x00000004)
#define SDRCON_PIPE1 (0x00000008)
#define SDRCON_PG256 (0x00000010)
#define SDRCON_PG512 (0x00000000)
#define SDRC0N_PG1K (0x00000020)
#define SDRCON_REF1100 (0x00000000)
#define SDRCON_REF1850 (0x00000080)
#define SDRCON_REF2200 (0x00000100)
#define SDRCON_REF3700 (0x00000180)
#define SDRCON_PC2RAS2 (0x00000000)
#define SDRCON_PC2RAS3 (0x00000200)
#define SDRCON_PC2RAS4 (0x00000400)
#define SDRCON_PC2RAS5 (0x00000600)
#define SDRCON_RAS2PC2 (0x00000000)
#define SDRCON_RAS2PC3 (0x00000800)
#define SDRCON_RAS2PC4 (0x00001000)
#define SDRCON_RAS2PC5 (0x00001800)
#define SDRCON_RAS2PC6 (0x00002000)
#define SDRCON_RAS2PC7 (0x00002800)
#define SDRCON_RAS2PC8 (0x00003000)
#define SDRCON_INIT (0x00004000)
#define SDRCON_EMRS (0x00008000)

//* Регистры SYSTAT *
#define SYSTAT_LOC (0x1F0086)
```

```
#define SYSTATCL_LOC (0x1F0087)

//* Регистры BMAX *
#define BMAX_LOC (0x1F008C)
#define BMAXC_LOC (0x1F008D)

//* Регистры буферов линков - изменено 13.07.04 под новые спецификации
#define LBUFTX0_LOC (0x1F04A0)
#define LBUFRX0_LOC (0x1F04A4)
#define LBUFTX1_LOC (0x1F04A8)
#define LBUFRX1_LOC (0x1F04AC)
#define LBUFTX2_LOC (0x1F04B0)
#define LBUFRX2_LOC (0x1F04B4)
#define LBUFTX3_LOC (0x1F04B8)
#define LBUFRX3_LOC (0x1F04BC)

//* Регистры управления приема по линкам с масками битов *
#define LRCTL0_LOC (0x1F00E0)
#define LRCTL1_LOC (0x1F00E1)
#define LRCTL2_LOC (0x1F00E2)
#define LRCTL3_LOC (0x1F00E3)

// Позиции битов
#define LRCTL_REN_P (0)
#define LRCTL_RVERE_P (1)
#define LRCTL_RTOE_P (2)
#define LRCTL_RBCMPE_P (3)
#define LRCTL_RDSIZE_P (4)
#define LRCTL_ROVRE_P (5)
#define LRCTL_RINIF_P (6)
#define LRCTL_RINIV_P (7)

// Битовые маски
#define LRCTL_REN MAKE_BITMASK_(LRCTL_REN_P)
#define LRCTL_RVERE MAKE_BITMASK_(LRCTL_RVERE_P)
#define LRCTL_RTOE MAKE_BITMASK_(LRCTL_RTOE_P)
#define LRCTL_RBCMPE MAKE_BITMASK_(LRCTL_RBCMPE_P)
#define LRCTL_RDSIZE MAKE_BITMASK_(LRCTL_RDSIZE_P)
#define LRCTL_ROVRE MAKE_BITMASK_(LRCTL_ROVRE_P)

//* Регистры управления передачей по линкам с масками битов *
#define LTCTL0_LOC (0x1F00E4)
#define LTCTL1_LOC (0x1F00E5)
#define LTCTL2_LOC (0x1F00E6)
#define LTCTL3_LOC (0x1F00E7)

// Позиции битов
#define LTCTL_TEN_P (0)
#define LTCTL_TVERE_P (1)
#define LTCTL_TTOE_P (2)
#define LTCTL_TBCMPE_P (3)
#define LTCTL_TDSIZE_P (4)

// Битовые маски
```

```
#define LTCTL_TEN MAKE_BITMASK_(LTCTL_TEN_P)
#define LTCTL_TVERE MAKE_BITMASK_(LTCTL_TVERE_P)
#define LTCTL_TTOE MAKE_BITMASK_(LTCTL _TTOE_P)
#define LTCTL_TBCMPE MAKE_BITMASK_(LTCTL_TBCMPE_P)
#define LTCTL_TDSIZE MAKE_BITMASK_(LTCTL_TDSIZE_P)
#define LTCTL_TCLKDIV1 (0x00000000)
#define LTCTL_TCLKDIV1P5 (0x00000020)
#define LTCTL_TCLKDIV2 (0x00000040)
#define LTCTL_TCLKDIV4 (0x00000080)

//* Регистры статуса линков с масками битов *
#define LRSTAT0_LOC (0x1F00F0)
#define LRSTAT1_LOC (0x1F00F1)
#define LRSTAT2_LOC (0x1F00F2)
#define LRSTAT3_LOC (0x1F00F3)
#define LTSTAT0_LOC (0x1F00F4)
#define LTSTAT1_LOC (0x1F00F5)
#define LTSTAT2_LOC (0x1F00F6)
#define LTSTAT3_LOC (0x1F00F7)
#define LRSTATC0_LOC (0x1F00F8)
#define LRSTATC1_LOC (0x1F00F9)
#define LRSTATC2_LOC (0x1F00FA)
#define LRSTATC3_LOC (0x1F00FB)
#define LTSTATC0_LOC (0x1F00FC)
#define LTSTATC1_LOC (0x1F00FD)
#define LTSTATC2_LOC (0x1F00FE)
#define LTSTATC3_LOC (0x1F00FF)

// Позиции битов - статус приема
#define LRSTAT_RTER_P (2)
#define LRSTAT_RWER_P (3)
#define LRSTAT_RCSER_P (4)
#define LRSTAT_ROVER_P (5)

// Позиции битов - статус передачи
#define LTSTAT_TVACANT_P (0)
#define LTSTAT_TEMPTY_P (1)
#define LTSTAT_TTER_P (2)
#define LTSTAT_TWER_P (3)

// Битовые маски - статус приема
#define LRSTAT_RSTAT (0x00000003)
#define LRSTAT_RTER MAKE_BITMASK_(LRSTAT_RTER_P)
#define LRSTAT_RWER MAKE_BITMASK_(LRSTAT_RWER_P)
#define LRSTAT_RCSER MAKE_BITMASK_(LRSTAT_RCSER_P)
#define LRSTAT_ROVER MAKE_BITMASK_(LRSTAT_ROVER_P)

// Битовые маски - статус передачи
#define LTSTAT_TVACANT MAKE_BITMASK_(LTSTAT_TVACANT_P)
#define LTSTAT_TEMPTY MAKE_BITMASK_(LTSTAT_TEMPTY_P)
#define LTSTAT_TTER MAKE_BITMASK_(LTSTAT_TTER_P)
#define LTSTAT_TWER MAKE_BITMASK_(LTSTAT_TWER_P)

//* Вектора прерываний (за исключением программного (SW) прерывания) *
```

```
#define IVKERNEL_LOC (0x1F0300)
#define IVTIMER0LP_LOC (0x1F0302)
#define IVTIMER1LP_LOC (0x1F0303)
#define IVLINK0_LOC (0x1F0306)
#define IVLINK1_LOC (0x1F0307)
#define IVLINK2_LOC (0x1F0308)
#define IVLINK3_LOC (0x1F0309)
#define IVDMA0_LOC (0x1F030E)
#define IVDMA1_LOC (0x1F030F)
#define IVDMA2_LOC (0x1F0310)
#define IVDMA3_LOC (0x1F0311)
#define IVDMA4_LOC (0x1F0316)
#define IVDMA5_LOC (0x1F0317)
#define IVDMA6_LOC (0x1F0318)
#define IVDMA7_LOC (0x1F0319)
#define IVDMA8_LOC (0x1F03ID)
#define IVDMA9_LOC (0x1F031E)
#define IVDMA10_LOC (0x1F031F)
#define IVDMA11_LOC (0x1F0320)
#define IVDMA12_LOC (0x1F0325)
#define IVDMA13_LOC (0x1F0326)
#define IVIRQ0_LOC (0x1F0329)
#define IVIRQ1_LOC (0x1F032A)
#define IVIRQ2_LOC (0x1F032B)
#define IVIRQ3_LOC (0x1F032C)
#define VIRPT_LOC (0x1F0330)
#define IVBUSLK_LOC (0x1F0332)
#define IVBUSLOCK_LOC (0x1F0332) // доб. 06.05.04 по запросу от VDK
                                 // для поддержки совместимости с TS101
#define IVTIMER0HP_LOC (0x1F0334)
#define IVTIMER1HP_L0C (0x1F0335)
#define IVHW_LOC (0x1F0339)

//* ILAT, IMASK и PMASK с определениями битов *
#define ILATL_LOC (0x1F0340)
#define ILATH_LOC (0x1F0341)
#define ILATSTL_LOC (0x1F0342)
#define ILATSTH_LOC (0x1F0343)
#define ILATCLL_LOC (0x1F0344)
#define ILATCLH_LOC (0x1F0345)
#define PMASKL_LOC (0x1F0346)
#define PMASKH_LOC (0x1F0347)
#define IMASKL_LOC (0x1F0348)
#define IMASKH_LOC (0x1F0349)

// Позиции битов
#define INT_KERNEL_P (0)
#define INT_RES1_P (1)
#define INT_TIMER0L_P (2)
#define INT_TIMER1L_P (3)
#define INT_RES4_P (4)
#define INT_RES5_P (5)
#define INT_LINK0_P (6)
#define INT_LINK1_P (7)
```

```
#define INT_LINK2_P (8)
#define INT_LINK3_P (9)
#define INT_RES_10_P (10)
#define INT_RES_11_P (11)
#define INT_RES_12_P (12)
#define INT_RES_13_P (13)
#define INT_DMA0_P (14)
#define INT_DMA1_P (15)
#define INT_DMA2_P (16)
#define INT_DMA3_P (17)
#define INT_RES18_P (18)
#define INT_RES19_P (19)
#define INT_RES20_P (20)
#define INT_RES21_P (21)
#define INT_DMA4_P (22)
#define INT_DMA5_P (23)
#define INT_DMA6_P (24)
#define INT_DMA7_P (25)
#define INT_RES26_P (26)
#define INT_RES27_P (27)
#define INT_RES28_P (28)
#define INT_DMA8_P (29)
#define INT_DMA9_P (30)
#define INT_DMA10_P (31)
#define INT_DMA11_P (0)
#define INT_RES33_P (1)
#define INT_RES34_P (2)
#define INT_RES35_P (3)
#define INT_RES36_P (4)
#define INT_DMA12_P (5)
#define INT_DMA13_P (6)
#define INT_RES39_P (7)
#define INT_RES40_P (8)
#define INT_IRQ0_P (9)
#define INT_IRQ1_P (10)
#define INT_IRQ2_P (11)
#define INT_IRQ3_P (12)
#define INT_RES45_P (13)
#define INT_RES46_P (14)
#define INT_RES47_P (15)
#define INT_VIRPT_P (16)
#define INT_RES49_P (17)
#define INT_BUSLOCK_P (18)
#define INT_RES51_P (19)
#define INT_TIMER0H_P (20)
#define INT_TIMER1H_P (21)
#define INT_RES54_P (22)
#define INT_RES55_P (23)
#define INT_RES56_P (24)
#define INT_HWERR_P (25)
#define INT_RES58_P (26)
#define INT_RES59_P (27)
#define INT_RES61_P (29)
```

```
// Битовые маски только для C
#define INT_KERNEL_64 MAKE_LL_BITMASK_(INT_KERNEL_P + 0)
#define INT_RES1_64 MAKE_LL_BITMASK_(INT_RES1_P + 0)
#define INT_TIMER0L_64 MAKE_LL_BITMASK_(INT_TIMER0L_P + 0)
#define INT_TIMER1L_64 MAKE_LL_BITMASK_(INT_TIMER1L_P+0)
#define INT_RES4_64 MAKE_LL_BITMASK_(INT_RES4_P+0)
#define INT_RES5_64 MAKE_LL_BITMASK_(INT_RES5_P+0)
#define INT_LINK0_64 MAKE_LL_BITMASK_(INT_LINK0_P+0)
#define INT_LINK1_64 MAKE_LL_BITMASK_(INT_LINK1_P+0)
#define INT_LINK2_64 MAKE_LL_BITMASK_(INT_LINK2_P +0)
#define INT_LINK3_64 MAKE_LL_BITMASK_(INT_LINK3_P+0)
#define INT_RES_10_64 MAKE_LL_BITMASK_(INT_RES_10_P +0)
#define INT_RES_11_64 MAKE_LL_BITMASK_(INT_RES_11_P+0)
#define INT_RES_12_64 MAKE_LL_BITMASK_(INT_RES_12_P+0)
#define INT_RES_13_64 MAKE_LL_BITMASK_(INT_RES_13_P +0)
#define INT_DMA0_64 MAKE_LL_BITMASK_(INT_DMA0_P+0)
#define INT_DMA1_64 MAKE_LL_BITMASK_(INT_DMA1_P+0)
#define INT_DMA2_64 MAKE_LL_BITMASK_(INT_DMA2_P+0)
#define INT_DMA3_64 MAKE_LL_BITMASK_(INT_DMA3_P+0)
#define INT_RES18_64 MAKE_LL_BITMASK_(INT_RES18_P+0)
#define INT_RES19_64 MAKE_LL_BITMASK_(INT_RES19_P+0)
#define INT_RES20_64 MAKE_LL_BITMASK_(INT_RES2 0_P+0)
#define INT_RES21_64 MAKE_LL_BITMASK_(INT_RES2 1_P+0)
#define INT_DMA4_64 MAKE_LL_BITMASK_(INT_DMA4_P+0)
#define INT_DMA5_64 MAKE_LL_BITMASK_(INT_DMA5_P+0)
#define INT_DMA6_64 MAKE_LL_BITMASK_(INT_DMA6_P+0)
#define INT_DMA7_64 MAKE_LL_BITMASK_(INT_DMA7_P+0)
#define INT_RES26_64 MAKE_LL_BITMASK_(INT_RES26_P+0)
#define INT_RES27_64 MAKE_LL_BITMASK_(INT_RES27_P+0)
#define INT_RES28_64 MAKE_LL_BITMASK_(INT_RES28_P+0)
#define INT_DMA8_64 MAKE_LL_BITMASK_(INT_DMA8_P+0)
#define INT_DMA9_64 MAKE_LL_BITMASK_(INT_DMA9_P+0)
#define INT_DMA10_64 MAKE_LL_BITMASK_(INT_DMA10_P+0)
#define INT_DMA11_64 MAKE_LL_BITMASK_(INT_DMA11_P+32)
#define INT_RES33_64 MAKE_LL_BITMASK_(INT_RES33_P+32)
#define INT_RES34_64 MAKE_LL_BITMASK_(INT_RES34_P+32)
#define INT_RES35_64 MAKE_LL_BITMASK_(INT_RES35_P + 32)
#define INT_RES36_64 MAKE_LL_BITMASK_(INT_RES36_P+32)
#define INT_DMA12_64 MAKE_LL_BITMASK_(INT_DMA12_P+32)
#define INT_DMA13_64 MAKE_LL_BITMASK_(INT_DMA13_P + 32)
#define INT_RES39_64 MAKE_LL_BITMASK_(INT_RES39_P+32)
#define INT_RES40_64 MAKE_LL_BITMASK_(INT_RES40_P+32)
#define INT_IRQ0_64 MAKE_LL_BITMASK_(INT_IRQ0_P+32)
#define INT_IRQ1_64 MAKE_LL_BITMASK_(INT_IRQ1_P+32)
#define INT_IRQ2_64 MAKE_LL_BITMASK_(INT_IRQ2_P+32)
#define INT_IRQ3_64 MAKE_LL_BITMASK_(INT_IRQ3_P+32)
#define INT_RES45_64 MAKE_LL_BITMASK_(INT_RES45_P+32)
#define INT_RES46_64 MAKE_LL_BITMASK_(INT_RES46_P+32)
#define INT_RES47_64 MAKE_LL_BITMASK_(INT_RES47_P+32)
#define INT_VIRPT_64 MAKE_LL_BITMASK_(INT_VIRPT_P+32)
#define INT_RES49_64 MAKE_LL_BITMASK_(INT_RES49_P+32)
#define INT_BUSLOCK_64 MAKE_LL_BITMASK_(INT_BUSLOCK_P+32)
#define INT_RES51_64 MAKE_LL_BITMASK_(INT_RES51_P+32)
#define INT_TIMER0H_64 MAKE_LL_BITMASK_(INT_TIMER0H_P+32)
```

```
#define INT_TIMER1H_64 MAKE_LL_BITMASK_(INT_TIMER1H_P+32)
#define INT_RES54_64 MAKE_LL_BITMASK_(INT_RES54_P+32)
#define INT_RES55_64 MAKE_LL_BITMASK_(INT_RES55_P+32)
#define INT_RES56_64 MAKE_LL_BITMASK_(INT_RES56_P+32)
#define INT_HWERR_64 MAKE_LL_BITMASK_(INT_HWERR_P + 32)
#define INT_RES58_64 MAKE_LL_BITMASK_(INT_RES58_P + 32)
#define INT_RES59_64 MAKE_LL_BITMASK_(INT_RES59_P + 32)
#define INT_RES61_64 MAKE_LL_BITMASK_(INT_RES61_P + 32)

// Битовые маски
#define INT_KERNEL MAKE_BITMASK_(INT_KERNEL_P)
#define INT_RES1 MAKE_BITMASK_(INT_RES1_P)
#define INT_TIMER0L MAKE_BITMASK_(INT_TIMER0L_P)
#define INT_TIMER1L MAKE_BITMASK_(INT_TIMER1L_P)
#define INT_RES4 MAKE_BITMASK_(INT_RES4_P)
#define INT_RES5 MAKE_BITMASK_(INT_RES5_P)
#define INT_LINK0 MAKE_BITMASK_(INT_LINK0_P)
#define INT_LINK1 MAKE_BITMASK_(INT_LINK1_P)
#define INT_LINK2 MAKE_BITMASK_(INT_LINK2_P)
#define INT_LINK3 MAKE_BITMASK_(INT_LINK3_P)
#define INT_RES_10 MAKE_BITMASK_(INT_RES_10_P)
#define INT_RES_11 MAKE_BITMASK_(INT_RES_11_P)
#define INT_RES_12 MAKE_BITMASK_(INT_RES_12_P)
#define INT_RES_13 MAKE_BITMASK_(INT_RES_13_P)
#define INT_DMA0 MAKE_BITMASK_(INT_DMA0_P)
#define INT_DMA1 MAKE_BITMASK_(INT_DMA1_P)
#define INT_DMA2 MAKE_BITMASK_(INT_DMA2_P)
#define INT_DMA3 MAKE_BITMASK_(INT_DMA3_P)
#define INT_RES18 MAKE_BITMASK_(INT_RES18_P)
#define INT_RES19 MAKE_BITMASK_(INT_RES19_P)
#define INT_RES20 MAKE_BITMASK_(INT_RES20_P)
#define INT_RES21 MAKE_BITMASK_(INT_RES21_P)
#define INT_DMA4 MAKE_BITMASK_(INT_DMA4_P)
#define INT_DMA5 MAKE_BITMASK_(INT_DMA5_P)
#define INT_DMA6 MAKE_BITMASK_(INT_DMA6_P)
#define INT_DMA7 MAKE_BITMASK_(INT_DMA7_P)
#define INT_RES26 MAKE_BITMASK_(INT_RES26_P)
#define INT_RES27 MAKE_BITMASK_(INT_RES27_P)
#define INT_RES28 MAKE_BITMASK_(INT_RES28_P)
#define INT_DMA8 MAKE_BITMASK_(INT_DMA8_P)
#define INT_DMA9 MAKE_BITMASK_(INT_DMA9_P)
#define INT_DMA10 MAKE_BITMASK_(INT_DMA10_P)
#define INT_DMA11 MAKE_BITMASK_(INT_DMA11_P)
#define INT_RES33 MAKE_BITMASK_(INT_RES33_P)
#define INT_RES34 MAKE_BITMASK_(INT_RES34_P)
#define INT_RES35 MAKE_BITMASK_(INT_RES35_P)
#define INT_RES36 MAKE_BITMASK_(INT_RES36_P)
#define INT_DMA12 MAKE_BITMASK_(INT_DMA12_P)
#define INT_DMA13 MAKE_BITMASK_(INT_DMA13_P)
#define INT_RES39 MAKE_BITMASK_(INT_RES39_P)
#define INT_RES40 MAKE_BITMASK_(INT_RES40_P)
#define INT_IRQ0 MAKE_BITMASK_(INT_IRQ0_P)
#define INT_IRQ1 MAKE_BITMASK_(INT_IRQ1_P)
#define INT_IRQ2 MAKE_BITMASK_(INT_IRQ2_P)
```

```
#define INT_IRQ3 MAKE_BITMASK_(INT_IRQ3_P)
#define INT_RES45 MAKE_BITMASK_(INT_RES45_P)
#define INT_RES46 MAKE_BITMASK_(INT_RES46_P)
#define INT_RES47 MAKE_BITMASK_(INT_RES47_P)
#define INT_VIRPT MAKE_BITMASK_(INT_VIRPT_P)
#define INT_RES49 MAKE_BITMASK_(INT_RES49_P)
#define INT_BUSLOCK MAKE_BITMASK_(INT_BUSLOCK_P)
#define INT_RES51 MAKE_BITMASK_(INT_RES51_P)
#define INT_TIMER0H MAKE_BITMASK_(INT_TIMER0H_P)
#define INT_TIMER1H MAKE_BITMASK_(INT_TIMER1H_P)
#define INT_RES54 MAKE_BITMASK_(INT_RES54_P)
#define INT_RES55 MAKE_BITMASK_(INT_RES55_P)
#define INT_RES56 MAKE_BITMASK_(INT_RES56_P)
#define INT_HWERR MAKE_BITMASK_(INT_HWERR_P)
#define INT_RES58 MAKE_BITMASK_(INT_RES58_P)
#define INT_RES59 MAKE_BITMASK_(INT_RES59_P)
#define INT_RES61 MAKE_BITMASK_(INT_RES61_P)

//*Регистр управления прерываниями с определениями битов *
#define INTCTL_LOC (0x1F034E)

// Позиции битов
#define INTCTL_IRQ0_EDGE_P (0)
#define INTCTL_IRQ1_EDGE_P (1)
#define INTCTL_IRQ2_EDGE_P (2)
#define INTCTL_IRQ3_EDGE_P (3)
#define INTCTL_TMR0RN_P (4)
#define INTCTL_TMR1RN_P (5)

// Битовые маски
#define INTCTL_IRQ0_EDGE MAKE_BITMASK_(INTCTL_IRQ0_EDGE_P)
#define INTCTL_IRQ1_EDGE MAKE_BITMASK_(INTCTL_IRQ1_EDGE_P)
#define INTCTL_IRQ2_EDGE MAKE_BITMASK_(INTCTL_IRQ2_EDGE_P)
#define INTCTL_IRQ3_EDGE MAKE_BITMASK_(INTCTL_IRQ3_EDGE_P)
#define INTCTL_TMR0RN MAKE_BITMASK_(INTCTL_TMR0RN_P )
#define INTCTL_TMR1RN MAKE_BITMASK_(INTCTL_TMR1RN_P )

//* Регистры таймеров *
#define TIMER0L_LOC (0x1F0350)
#define TIMER0H_LOC (0x1F0351)
#define TIMER1L_LOC (0x1F0352)
#define TIMER1H_LOC (0x1F0353)
#define TMRIN0L_LOC (0x1F0354)
#define TMRIN0H_LOC (0x1F0355)
#define TMRIN1L_LOC (0x1F0356)
#define TMRIN1H_LOC (0x1F0357)

//* Регистры эмуляции II *
//* EMUCTL с определениями битов *
#define EMUCTL_LOC (0x1F03A0)

// Позиции битов
#define EMUCTL_EMEN_P (0)
#define EMUCTL_TEME_P (1)
```

```
#define EMUCTL_EMUOE_P (2)
#define EMUCTL_SPFDIS_P (3)
#define EMUCTL_SWRST_P (4)
#define EMUCTL_BOOTDSBL_P (5)

// Битовые маски
#define EMUCTL_EMEN MAKE_BITMASK_(EMUCTL_EMEN_P)
#define EMUCTL_TEME MAKE_BITMASK_(EMUCTL_TEME_P)
#define EMUCTL_EMUOE MAKE_BITMASK_(EMUCTL EMUOE_P)
#define EMUCTL_SPFDIS MAKE_BITMASK_(EMUCTL_SPFDIS_P)
#define EMUCTL_SWRST MAKE_BITMASK_(EMUCTL_SWRST_P)
#define EMUCTL_BOOTDSBL MAKE_BITMASK_(EMUCTL_BOOTDSBL_P)

//* EMUSTAT с определениями битов *
#define EMUSTAT_LOC (0x1F03A1)

// Позиции битов
#define EMUSTAT_EMUMOD_P (0)
#define EMUSTAT_IRFREE_P (1)
#define EMUSTAT_INRESET_P (2)

// Битовые маски
#define EMUSTAT_EMUMOD MAKE_BITMASK_(EMUSTAT_EMUMOD_P)
#define EMUSTAT_IRFREE MAKE_BITMASK_(EMUSTAT_IRFREE_P)
#define EMUSTAT_INRESET MAKE_BITMASK_(EMUSTAT_INRESET_P)
//*
#define EMUDAT_LOC (0x1F03A2)
#define IDCODE_LOC (0x1F03A3)
#define EMUIR_LOC (0x1F03A4)

//* Регистры буфера трассировки *
#define TRCB0_LOC (0xlE0140)
#define TRCB1_LOC (0xlE0141)
#define TRCB2_LOC (0xlE0142)
#define TRCB3_LOC (0xlE0143)
#define TRCB4_LOC (0xlE0144)
#define TRCB5_LOC (0xlE0145)
#define TRCB6_LOC (0xlE0146)
#define TRCB7_LOC (0xlE0147)
#define TRCB8_LOC (0xlE0148)
#define TRCB9_LOC (0xlE0149)
#define TRCB10_LOC (0xlE014A)
#define TRCB11_LOC (0xlE014B)
#define TRCB12_LOC (0xlE014C)
#define TRCB13_LOC (0xlE014D)
#define TRCB14_LOC (0xlE014E)
#define TRCB15_LOC (0xlE014F)
#define TRCB16_LOC (0xlE0150)
#define TRCB17_LOC (0xlE0151)
#define TRCB18_LOC (0xlE0152)
#define TRCB19_LOC (0xlE0153)
#define TRCB20_LOC (0xlE0154)
#define TRCB21_LOC (0xlE0155)
#define TRCB22_LOC (0xlE0156)
```

```
#define TRCB23_LOC (0xlE0157)
#define TRCB24_LOC (0xlE0158)
#define TRCB25_LOC (0xlE0159)
#define TRCB26_LOC (0xlE015A)
#define TRCB27_LOC (0xlE015B)
#define TRCB28_LOC (0xlE015C)
#define TRCB29_LOC (0xlE015D)
#define TRCB30_LOC (0xlE015E)
#define TRCB31_LOC (0x1E015F)
//*
#define AUTODMA0_LOC (0x1F03E0)
#define AUTODMA1_LOC (0x1F03E4)

//* Глобальная память *
#define BLOCK0_LOC(0x00000000) //Блок 0 внутренней памяти
#define BLOCK2_LOC (0x00040000) //Блок 2 внутренней памяти
#define BLOCK4_LOC (0x00080000) //Блок 4 внутренней памяти
#define BLOCK6_LOC (0x000C0000) //Блок 6 внутренней памяти
#define BLOCK8_LOC (0x00100000) //Блок 8 внутренней памяти
#define BLOCK10_LOC (0x00140000) //Блок 10 внутренней памяти

#define BCST_OFFSET_LOC (0x0C000000) // Смещ.области широковещательных
                                        адресов
                                        // в пространстве мульти- процессорной
                                        (MP-) памяти
#define P0_OFFSET_LOC (0x10000000) // Смещ.области адресов процессора ID0
                                        в MP-памяти
#define P1_OFFSET_LOC (0x14000000) // Смещ.области адресов процессора ID1
                                        в MP-памяти
#define P2_OFFSET_LOC (0x18000000) // Смещ.области адресов процессора ID2
                                        в MP-памяти
#define P3_OFFSET_LOC (0x10000000) // Смещ.области адресов процессора ID3
                                        в MP-памяти
#endif // ! defined ( DEFTS201_H_)
```

Приложение В
Декодирование команд

Настоящее приложение описывает коды операций команд, а также формирование этих кодов.

В.1 Структура команды

Все команды процессора представляют собой 32-битные слова, при этом старшие биты имеют одинаковый смысл для всех команд, как это показано на рисунке В-1.

Поле ITYPE определяет группу исполнения, которой принадлежит команда. Его длина варьируется в зависимости от типа команды. Формат строго определяется несколькими старшими битами (MSB) этого поля. Декодирование нескольких старших битов поля ITYPE представлено в таблице В-1.

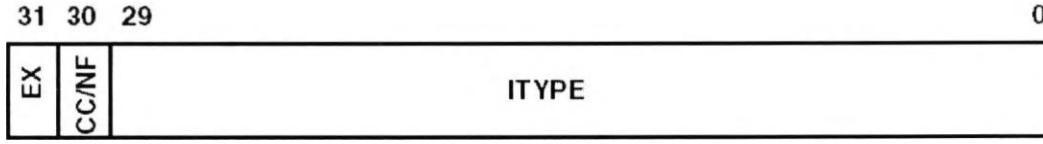


Рисунок В-1 – Структура команды

Таблица В-1 – Поле ITYPE

Биты	Поле	Описание
29-28	0<JK>	Целочисленная операция ALU (вычисления и/или загрузка/сохранение). Поле <JK> в командах IALU определяет, относится ли команда к J- IALU или K-IALU и интерпретируется следующим образом: 0 = J-IALU; 1 = K-IALU
29-25	1000<JK>	Операции целочисленного IALU, загрузка и сохранение байт или коротких числе. Бит <JK> интерпретируется как описанно выше
29-26	10<XY>	Команда ALU вычислительного блока. Поле <XY> в командах вычислительного блока является ненулевым двухбитным полем, которое определяет обращается ли команда к вычислительному блоку X, вычислительному блоку Y или к обоим. Интерпретируется следующим образом: 01 = вычислительный блок X; 10 = вычислительный блок Y; 11 = оба вычислительных блока X и Y
29-25	10<XY>0	Команда ALU вычислительного блока
29-24	10<XY>10	Команда сдвигового устройства вычислительного блока
29-24	10<XY>11	Команда умножителя вычислительного блока
29-26	11<XY>	Команда CLU вычислительного блока; где 1101 – CLU блока Y; 1110 – CLU блока X; 1111 – CLU блоков X и Y
29-26	1100	Команды управления потоком исполнения, непосредственные расширения и др.

Биты	Поле	Описание
30	CC/NF	Для командных строк, начинающихся с условных команд устройства управления (if <предикат>...), этот бит интерпретируется как бит CC. В противном случае этот бит интерпретируется как бит NF. В случае командной строки, начинающейся с условной команды, действие бита NF, присутствующего в этой команде, распространяется на всю командную строку. Применимо к командам вычислительного блока, IALU и к командам загрузки/сохранения. Неприменимо к условным командам, непосредственным расширениям и др. (см. пункт «Формат команды косвенного перехода (УУ)»). Если бит 30 является битом CC (признаком условности), то если он установлен – обуславливает исполнение команды значением предиката в командной строке. Если бит 30 является битом NF (не обновлять флаги), то если он установлен – текущая команда не обновляет значения флагов по завершению исполнения этой команды
31	EX	Установленный бит EX определяет команду как последнюю в командной строке

B.2 Формат команды вычислительного блока

Формат всех команд ALU, умножителя, а также некоторых команд сдвигового устройства описывается рисунком B-2 и таблицей B-2. Поле XY не должно быть нулевым.

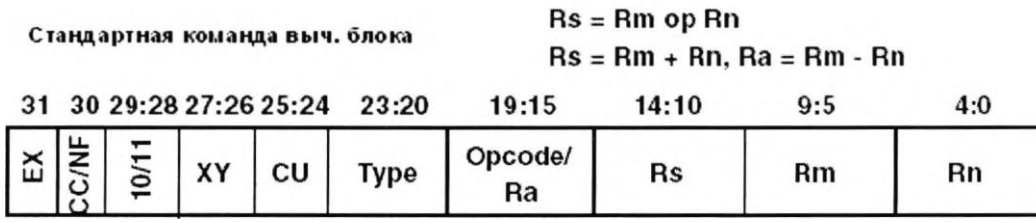


Рисунок B-2 – Формат стандартной команды вычислительного блока

Таблица B-2 – Поля кода операции команды вычислительного блока

Биты	Поле	Описание
4-0	RN	Определяет второй операнд команды
9-5	RM	Определяет первый операнд команды
14-10	RS	Определяет регистры результата
19-15	OPCODE / RA	См. таблицы B-3, B-4, B-5, B-7, B-8, B-9, B-10, B-11, B-12, B-23, B-24
23-20	TYPE	См. таблицы B-3, B-4, B-5, B-7, B-8, B-9, B-10, B-11, B-12, B-23, B-24
25-24	CU	Определяет тип команды вычислительного блока: 00 = команда ALU фиксированной точки; 01 = команда ALU фиксированной и плавающей точки; 10 = Команда сдвигового устройства; 11 = Команда умножителя

Биты	Поле	Описание
27-26	XY	Задает вычислительный блок, который должен исполнять операцию: 01 = исполняющим устройством назначается вычислительный блок X; 10 = исполняющим устройством назначается вычислительный блок Y; 11 = исполняющими устройствами назначаются оба вычислительных блока
29-28	10	Определяет, что операция должна исполняться: устройством ALU вычислительного блока, умножителем или сдвиговым устройством
29-28	11	Определяет, что операция должна исполняться устройством CLU вычислительного блока
30	CC/NF	Для командных строк, начинающихся с условных команд устройства управления (if <предикат>...), этот бит интерпретируется как бит CC. В противном случае этот бит интерпретируется как бит NF. В случае командной строки, начинающейся с условной команды, действие бита NF, присутствующего в этой команде, распространяется на всю командную строку. Применимо к командам вычислительного блока. IALU и к командам загрузки/сохранения. Неприменимо к условным командам, непосредственным расширениям и др. (см. пункт «Формат команды косвенного перехода (УУ)»). Если бит 30 является битом CC (признаком условности), то он, будучи установлен, обуславливает исполнение команды значением предиката в командной строке. Если бит 30 является битом NF (не обновлять флаги), то когда он установлен, текущая команда не обновляет значения флагов по завершению исполнения этой команды
31	EX	Установленный бит EX определяет команду как последнюю в командной строке

B.3 Команды ALU

Синтакис и коды операции команд ALU описываются в следующих разделах:

- ALU фиксированной точки: арифметические и логические команды (CU=00);
- ALU фиксированной точки: команды преобразования данных (CU=01);
- ALU плавающей точки: арифметические и логические команды.

ALU фиксированной точки: арифметические и логические команды (CU=00)

В таблице B-3 приведен синтаксис команд ALU фиксированной точки вместе со значениями полей типа (TYPE) и кода операции. Операнды и результаты этих команд имеют одинаковые типы и размеры данных.

Таблица В-3 – Синтаксис и коды операции арифметических и логических команд ALU фиксированной точки

Синтаксис	Тип	Код операции	
(B/S/L)Rs(d) = Rm + Rn	0000-Rs	000	S1 S0[3]
(B/S/L)Rs(d) = Rm – Rn	0011-(L)Rsd[1]	001	S1 S0[3]
(B/S/L)Rs(d) = SETMASK[4] Rm {} Rn	0110-SRs	001	00
(B/S/L)Rs(d) = ABS (Rm + Rn)	1001-BRs	010	OX[5]
(B/S/L)Rs(d) = ABS (Rm - Rn)[5]	1100-B/SRsd[2]	011	UX
(B/S/L)Rs(d) = (Rm + Rn) / 2		100	UT[6]
(B/S/L)Rs(d) = (Rm - Rn) / 2		101	UT[6]
(B/S/L)Rs(d) = ROUNDUP (Rm + Rn) / 2		100	UT[6]
(B/S/L)Rs(d) = ROUNDUP (Rm - Rn) / 2		101	UT[6]
(B/S/L)Rs(d) = ROUNDUP[7, 8] (Rm(d) + Rn(d)) / 2		100	UT[6]
(B/S/L)Rs(d) = ROUNDUP[7, 8] (Rm(d) - Rn(d)) / 2		101	UT[6]
(B/S/L)Rs(d) = MAX (Rm, Rn)		110	UZ[9]
(B/S/L)Rs(d) = MIN (Rm, Rn)		111	UZ[9]
(B/S/L)Rs = CLIPZ (Rm, Rn)[7]		111	00
(L)Rs(d) = Rm + Rn + CI[10]	0001-Rs	000	S1 S0[3]
(L)Rs(d) = Rm - Rn + CI – I[10]	0100-(L)Rsd[1]	001	S1 S0[3]
(B/S/L)Rs(d) = INC Rm	0111-SRs	010	S1 S0[3]
(B/S/L)Rs(d) = DEC Rm	1010-BRs	011	S1 S0[3]
(B/S/L)COMP (Rm, Rn) – знаковое	1101-B/SRsd[2]	100	00
(B/S/L)COMP (Rm, Rn) – беззнаковое		100	10
(B/S/L)Rs = CLIP Rm by Rn		101	00
(L)Rs(d) = PASS Rm[10]		101	01
(B/S/L)Rs(d) = ABS Rm		101	10
(B/S/L)Rs(d) = -Rm		101	11
(L)Rs(d) = Rm AND Rn[10]		110	00
(L)Rs(d) = Rm OR Rn[10]		110	01
(L)Rs(d) = Rm XOR Rn[10]		110	10
(L)Rs(d) = NOT Rm[10]		110	11
S/BRsd = VMAX (Rmd, Rnd)[11]		110	00
S/BRsd = VMIN (Rmd, Rnd)[11]		111	00
(L)Rs(d) = Rm + CI[10]		111	00
(L)Rs(d) = Rm + CI - I[10]		111	01
(L)Rs(d) = Rm AND NOT Rn[10]		111	11

[1] LSB регистра Rn (бит 0 команды) используется для декодирования размера операнда, где:
0 = определяет операнд как двойной обычный;
1 = определяет операнд как длинное слово.

[2] LSB регистра Rn (бит 0 команды) используется для декодирования размера операнда, где:
0 = определяет операнд как восьмерку байтов;
1 = определяет операнд как четверку коротких слов.

[3] Значения S1 S0: без насыщения () – 00, насыщение знаковое (S) – 01, насыщение беззнаковое (SU) – 11.

[4] Поэлементное векторное сравнение. Выполняется только без опции насыщения и с одни из префиксов: 0x33C0001C (EQ), 0x33C0001D (GE), 0x33C0001C (GT).

[5] Опция X – расширенный диапазон для результата ABS.

[6] Для команды Rs = (Rm +/- Rn)/2 поле (TU) декодируется следующим образом (биты [16:15], UT):
11 = беззнаковое с усечением (TU);
10 = беззнаковое с округлением к ближайшему четному (U);
01 = знаковое с усечением (T);
00 = знаковое с округлением к ближайшему четному ().

[7] Выполняется только с префиксом 0x33C0001A.

[8] С округлением до бесконечности.

[9] Для команд MIN и MAX опции кодируются так: () = 00, (U) = 10, (Z) = 01 и (UZ) = 11.

[10] Команда реализована только для одиночных обычных или длинных данных.

[11] Команды VMIN и VMAX используют тот же код операции, что и другие команды, но только для коротких и байтовых данных, в то время как другие команды реализованы только для словных и длинных.

Синтаксис	Тип	Код операции
(B/S/L)Rs(d) = Rm + Rn, (B/S/L)Ra(d) = Rm - Rn	0010-Rs 0101-(L)Rsd[1] 1000-SRs 1011-BRs 1110-B/SRsd[2]	-Ra-

ALU фиксированной точки: команды преобразования данных (CU=01)

В таблице B-4 приведен синтаксис, а также значения полей типа (TYPE) и кода операции для команд ALU фиксированной точки, имеющих короткие, байтовые и смешанные операнды.

Таблица B-4 – Синтаксис и коды операции команд ALU фиксированной точки, осуществляющих преобразование данных

Синтаксис	CU = 01 фикс.	Тип	Код операции	
Rsd = EXPAND SRm + SRn		0000	000	T1 T0[1]
Rsq = EXPAND SRmd +/- SRnd[2]		0000	001	T1 T0[1]
Rsq = EXPAND SRmd - SRnd (ZF)[3]		0000	001	T1 T0[1]
SRsd = EXPAND BRm + BRn		0000	010	T1 T0[1]
SRsq = EXPAND BRmd +/- B Rnd[2]		0000	011	T1 T0[1]
SRsq = EXPAND BRmd - B Rnd (ZF)[3]		0000	011	T1 T0[1]
Rsd = EXPAND SRm – SRn		0000	100	T1 T0[1]
Rsd = EXPAND SRm – SRn (ZF)[3]		0000	100	T1 T0[1]
SRsd = EXPAND BRm – BRn		0000	110	T1 T0[1]
SRsd = EXPAND BRm - BRn(ZF)[3]		0000	110	T1 T0[1]
Rsd = EXPAND SRm	Rn = 00000	0000	111	T1 T0[1]
Rsq = EXPAND SRmd	Rn = 00001	0000	111	T1 T0[1]
SRsd = EXPAND BRm	Rn = 00010	0000	111	T1 T0[1]
SRsq = EXPAND BRmd	Rn = 00011	0000	111	T1 T0[1]
SRs = COMPACT Rmd +/- Rnd[2]		0001	00	C2 C1 C0[4]
BRs = COMPACT SRmd +/- SRnd[2]		0001	01	C2 C1 C0[4]
SRs = COMPACT Rmd		0001	10	C2 C1 C0[4]
BRs = COMPACT SRmd		0001	11	C2 C1 C0[4]
Rs =COMPACT LRmd		0011	01	100
LRs =COMPACT QRmd		0011	01	100
Rs = COMPACT LRmd	Rn = 00000	0011	011	00
Rs = COMPACT LRmd (U)	Rn = 00001	0011	011	00
Rs = COMPACT LRmd (IS)	Rn = 00010	0011	011	00
Rs = COMPACT LRmd (ISU)	Rn = 00011	0011	011	00
LRsd = COMPACT QRmq	Rn = 00100	0011	011	00
LRsd = COMPACT QRmq (U)	Rn = 00101	0011	011	00
LRsd = COMPACT QRmq (IS)	Rn = 00110	0011	011	00
LRsd = COMPACT QRmq (ISU)	Rn = 00111	0011	011	00
BRsd = MERGE Rm, Rn		0010	000	00
BRsq = MERGE Rmd, Rnd		0010	000	01
SRsd = MERGE Rm, Rn		0010	000	10
SRsq = MERGE Rmd, Rnd		0010	000	11
Rs = SUM SRm[5]	Rn = 00000	0010	001	S0 0[6]
Rs = SUM SRmd[5]	Rn = 00001	0010	001	S0 0[6]
Rs = SUM BRm[5]	Rn = 00010	0010	001	S0 0[6]

Синтаксис	CU = 01 фикс.	Тип	Код операции	
Rs = SUM BRmd[5]	Rn = 00011	0010	001	S0 0[6]
Rs = ONES Rm	Rn = 00100	0010	001	00
Rs = ONES Rmd	Rn = 00101	0010	001	00
Rsd = PR1:0	Rn = 00110	0010	001	00
Rs = BFOINC Rmd		0010	010	00
PR0 += ABS (SRmd - SRnd)		0011	000	S0 0[6]
PR0 += ABS (BRmd - BRnd)		0011	001	S0 0[6]
PR1 += ABS (SRmd - SRnd)		0011	000	S0 1[6]
PR1 += ABS (BRmd - BRnd)		0011	001	S0 1[6]
PR0+= SUM SRm	Rn = 00000	0011	010	S0 0[6]
PR0 += SUM SRmd	Rn = 00001	0011	010	S0 0[6]
PR0 += SUM BRm	Rn = 00010	0011	010	S0 0[6]
PR0 += SUM BRmd	Rn = 00011	0011	010	S0 0[6]
PR1 += SUM SRm	Rn = 00100	0011	010	S0 0[6]
PR1 += SUM SRmd	Rn = 00101	0011	010	S0 0[6]
PR1 += SUM BRm	Rn = 00110	0011	010	S0 0[6]
PR1 += SUM BRmd	Rn = 00111	0011	010	S0 0[6]
PR1:0 = Rmd	Rn = 01000	0011	010	00
Reserved	Rn > 01000	0011	010	XX
Rsq = Permute (Rmd, -Rmd, Rn)		1100	101	00
Rsd = Permute (Rmd, Rn)		1100	101	01

---

1 Значения T1 T0: дробные ( ) – 00, целые знаковые (I) – 01, целые беззнаковые (IU) – 11.

2 LSB регистра Rn в командах EXPAND и COMPACT используется для кодирования +/-, причем LSB = 0 определяет сложение, а LSB = 1 – вычитание.

3 Знаковое расширение с вычитанием. Выполняется только после префикса 0x33C0001A.

4 Компактное кодирование – C2 C1 C0 – используется для указания комбинаций опций, которые могут употребляться с командами. Задействованы следующие комбинации:
000 = дробное с округлением;
001 = целое знаковое, без насыщения (I);
100 = дробное с усечением (T);
101 = целое знаковое, с насыщением (IS);
111 = целое беззнаковое, с насыщением (1SU).

5 SUM – продольная сумма. Например, BR5 = SUM R1:0 складывает 8 байтов в двойном регистре R1:0 и сохраняет результат в R5. Команды SUM с участием PR0/1 накапливают результаты в регистрах PR – двух регистрах-аккумуляторах, используемых главным образом при сравнении блоков данных (block matching).

6 SO равно 0 для операций над знаковыми операндами и 1 – над беззнаковыми.

ALU плавающей точки: арифметические и логические команды

В таблице B-5 приведен синтаксис, а также значения полей типа (TYPE) и кода операции для команд ALU плавающей точки. Следует отметить, что здесь поле CU = 01, а от команд фиксированной точки, перечисленных в таблице B-4, у которых также CU = 01, команды плавающей точки отличаются иными кодами типа. В таблице B-5 приведены коды команд для ПЗ одинарной (d==0), расширенной (d==1) и двойной точности (d==1). Поскольку коды расширенной и двойной точности совпадают, то дополнительный выбор формата определяется соответствующим битом регистра управления SQCTL.

Таблица B-5 – Синтаксис и коды операции арифметических команд ALU плавающей точки

Синтаксис	(CU = 01 плав.)	Тип	Код операции
FRs(d) = Rm(d) + Rn(d)		0100	000 T d[1, 2]
FRs(d) = Rm(d) - Rn(d)		0100	001 T d
FRs(d) = (Rm(d) + Rn(d)) / 2		0100	010 T d
FRs(d) = (Rm(d) - Rn(d)) / 2		0100	011 T d
FRs(d) = ABS (Rm(d)+ Rn(d))		0100	100 T d
FRs(d) = ABS (Rm(d) - Rn(d))		0100	101 T d
FRs(d) = FLOAT Rm by Rn		0100	110Td
Rs = FIX FRm(d) by Rn		0100	111 T d
FRs(d) = CLIP Rm(d) by Rn(d)		0101	000 0 d
FRs(d) = Rm(d) COPYSIGN Rn(d)		0101	000 1 d
FRs(d) = SCALB FRm(d) by Rn		0101	001 0 d
FRs(d) = FLOAT Rm	Rn =00000	0101	010 T d
FRsd = FLOAT LRmd	10000	0101	010 0 1[3]
FRs(d) = ABS Rm(d)	Rn =00001	0101	010 0d
Rs = MANT FRm(d)	Rn = 00010	0101	010 0d
LRsd = MANT FRmd	01110	0101	010 01[3]
FRs(d) = PASS Rm(d)	Rn =00011	0101	010 0d
FRs(d) = - Rm(d)	Rn =00100	0101	010 0d
FRs(d) = RECIPS Rm(d)	Rn = 00101	0101	010 0d
FRs(d) = RSQRTS Rm(d)	Rn =00110	0101	010 0d
Rs = FIX FRm(d)	Rn =00111	0101	010 Td
LRsd = FIX FRmd	01111	0101	010 T1[3]
Rs = LOGB FRm(d)	Rn = 01000	0101	010 Sd[4]
FRsd = EXTD Rm - рез. с повышенной точн.	Rn = 01001	0101	010 0 1
FRs = SNGL Rmd - рез. с одинарной точн.	Rn =01010	0101	010T0
FRs(d) = MAX (Rm(d), Rn(d))		0101	011 0 d
FRs(d) = MIN (Rm(d), Rn(d))		0101	011 1 d
FCOMP (Rm(d), Rn(d))		0101	100 0 d
FRs = Rm + Rn, FRa = Rm - Rn - всегда с округлением		0110	-Ra-
FRsd = Rmd + Rnd, FRad = Rmd - Rnd - всегда с округлением		0111	-Rad-

[1] T: с округлением (T=0); с усечением (T=1) для всех команд ALU с плавающей точкой.

[2] d: формат повышенной точности неявно определяется размером операнда. Например, в команде FR1:0 = R3:2 + R5:4 бит d установлен (d=I), что означает размещение операндов и результата с плавающей точкой в двойных регистрах, а следовательно, использование 64(40)-битного формата повышенной точности. В команде FR0 = R2 + R3 бит d сброшен (d=0), что означает использование обычного одиночного регистра в 32-битном формате плавающей точки стандарта IEEE. Это справедливо для всех команд ALU плавающей точки.

[3] Коды команд, выделенные красным цветом, относятся только к вычислениям с двойной точностью.

[4] S задает насыщение: 0 – без насыщения, 1 – с насыщением.

В таблице В-6 преведены коды векторных команд для формата одинарной точности. Назначение бит T и S аналогично стандартному (таблица В-5) формату.

Таблица В-6 – Синтаксис и коды операции векторных команд

Синтаксис	Rn	Тип	Код операции
FRsd = RmH + RnH, RmL + RnL		1000	000T0
FRsd = RmH - RnH, RmL - RnL		1000	001T0
FRsd = (RmH + RnH) / 2, (RmL + RnL) / 2		1000	010T0
FRsd = (RmH - RnH) / 2, (RmL - RnL) / 2		1000	011T0
FRsd = ABS (RmH + RnH), ABS (RmL + RnL)		1000	100T0
FRsd = ABS (RmH - RnH), ABS (RmL - RnL)		1000	101T0
FRsd = FLOAT RmH by RnH, FLOAT RmL by RnL		1000	110T0
Rsd = FIX RmH by RnH, FIX RmL by RnL		1000	111T0
FRsd = CLIP RmH by RnH, CLIP RmL by RnL		1001	00000
FRsd = RmH COPYSIGN RnH, RmL COPYSIGN RnL		1001	00010
FRsd = SCALB FRmL BY RnL, SCALB FRmH BY RnH		1001	00100
FRsd = FLOAT RnL, FLOAT RmH	00000	1001	010T0
FRsd = ABS (RmH), ABS(RmL)	00001	1001	01000
Rsd = MANT FRmH, MANT FRmL	00010	1001	01000
FRsd = PASS RmH, PASS RmL	00011	1001	01000
FRsd = -RmH, -RmL	00100	1001	01000
FRsd = RECIPS RmH, RECIPS RmL	00101	1001	01000
FRsd = RSQRTS RmH, RSQRTS RmL	00110	1001	01000
Fsd = FIX FRmH, FIX FRmL	00111	1001	010T0
Rsd = LOGB FRmH, LOGB FRmL	01000	1001	010S0
FRsd = MAX RmH by RnH, MAX RmL by RnL		1001	0110 0
FRsd = MIN RmH by RnH, MIN RmL by RnL		1001	01110
FCOMP (RmH, RnH), COMP(RmL, RnL)		1001	10000
FRsd = RmH + RnH, RmL + RnL, FRad = RmH - RnH, RmL – RnL всегда с округлением		1010	-Ra-

B.4 Команды CLU

Команды коммуникационно-логического устройства (CLU) являются командами вычислительного блока и похожи на команды ALU или сдвигового устройства. Смысл битов 25:0 кода операции поясняется в таблице B-7, в то время как биты 31:26 имеют тот же смысл, что и все команды вычислительного блока.

Таблица B-7 – Синтаксис и коды операции команд комм-логического устройства (CLU)

25:20	19	18:17	16:15	14:12 11 10	9: 7	6:5	4:3 2 1 0
CU и тип	**Код операции**						
Rs(d)(q) = TRm(d)(q)							
00 0000	00000			Rs(d)(q)	000	NLQ[1]	TRm(d)(q)
Rs(d)(q) = THRm(d)(q)							
00 0000	00001			Rs(d)(q)	000	NLQ[1]	THRm(d)(q)
Rs = CMCTL							
00 0000	00001			Rs	00000		11111
Коды операции команд загрузки TR, THR и CMCTL см. в таблице B-12							
(S)TRsd = MAX(TRmd + Rmq_h, TRnd + Rmq_l)							
00 0010	0	TRsd [4:1]		TRnd [4:1] 0	Rmq [4:2]	00	TRmd [4:1] S[2]
(S)TRsd = MAX(TRmd - Rmq_h, TRnd - Rmq_1)							
00 0010	0	TRsd [4:1]		TRnd [4:1] 1	Rmq [4:2]	00	TRmd [4:1] S[2]
(S)TRsd = TMAX(TRmd + Rmq_h, TRnd + Rmq_l)							
00 0010	1	TRsd [4:1]		TRnd [4:1] 0	Rmq [4:2]	00	TRmd [4:1] S[2]
(S)TRsd = TMAX(TRmd - Rmq_h, TRnd - Rmq_l)							
00 0010	1	TRsd [4:1]		TRnd [4:1] 1	Rmq [4:2]	00	TRmd [4:1] S[2]
(S)Rs = TMAX(TRm, TRn)							
00 0011	0000S			Rs	TRn		TRm
(S)TRsq = ACS (TRmd, TRnd, Rm) (TMAX)[3]							
00 0100	0	TRsq [4:2] 0		TRnd [4:1] TMAX	Rm		TRmd [4:1] S[2]
Rsq = TRaq, (S)TRsq = ACS (TRmd, TRnd, Rm) (TMAX)[3]							
01 Rsq[4:2] TRAQ[4:1]	TRsq [4:2]		TRAQ [3:2]	TRnd [4:1] TMAX	Rm		TRmd [4:1] S[2]
TRs += DESPREAD (Rmq, THRmd)							
00 1000	TRs [0]	TRs [4:1]		00000	Rmq [4:2]	00	THRmd [4:1] 0
Rs = TRs, TRs = DESPREAD (Rmq, THRmd)							
00 1001	TRs [0]	TRs [4:1]		Rs	Rmq [4:2]	00	THRmd [4:1] 0
Rsd = TRsd, TRsd = DESPREAD (Rmq, THRmd)							
00 1011	0	TRsd [4:1]		Rsd [4:1] 0	Rmq [4:2]	00	THRmd [4:1] 0
TR31:16/15:0 = XCORRS (Rmq. THRmq)(cut#) (clr)(ext)							
00 11E0[4]	THL	00	CUT [5:4]	000 CLR R[5]	Rmq [4:2]	CUT [3:2]	THRmq [4:2] CUT [1:0]
Rsq = TRsq, TR31:16/15:0 = XCORRS (Rmq,THRmq) (cut Imm) (clr) (ext)							
00 11E1[4]	THL	TRsq[6] [3:2]	CUT [5:4]	Rsq [4:2] CLR R[5]	Rmq [4:2]	CUT [3:2]	THRmq [4:2] CUT [1:0]
Коды операции команд перестановок (permute) см. в таблице B-4							

[1] Определяет размер данных: 00 – для обычного слова (32 бита), 10 – для двойного слова (64 бита) и 11 – для квадрослова (128 бит).

[2] Бит S равен 0 при коротком слове, 1 – при обычном.

[3] Хотя имеются и другие регистры THR, команда ACS допускает ссылки только на THR 1:0.

[4] При указании опции повышенной точности (ext) бит E (бит 21) равен 1, в противном случае – 0.

[5] Бит R определяет, задается ли величина отсечения (cut) регистром (если R = 1) или непосредственным значением (если R = 0). Если бит R установлен, поле отсечения (биты 16, 15, 6, 5, 1 и 0) обнулено.

[6] TRsq [4] = THL, THL = 0 для TR31:16 или 1 для TR15:0.

B.5 Команды умножителя

Для умножителя код операции определяется набором опций. Опции кодируются следующим образом:

Биты 'xy' определяют опции (U) и (NU):
00: оба операнда знаковые – ();
10: оба операнда беззнаковые – (U);
01: Rm знаковый, а Rn беззнаковый – (NU).

Бит 'U' указывает на то, что данные беззнаковые (если бит установлен) или знаковые (если бит сброшен).

Бит 'I' указывает на то, что данные целые (если бит установлен) или дробные (если бит сброшен). В ассемблерной записи команды, если не указана опция (I), данные подразумеваются дробными.

Бит 'S' указывает на то, что команда выполняется с насыщением (если бит установлен) или без насыщения (если бит сброшен).

Бит 'T' указывает на то, что команда выполняется с усечением (если бит установлен) или с округлением (если бит сброшен). Этот бит значим только для данных дробного формата.

Биты 'T' и 'S' для команды умножения с накоплением означают следующее:
C = 0, R = 0 – обычное умножение с накоплением;
C = 1, R = 0 – очистка регистров MR перед умножением с накоплением;
C = 1, R = 1 – очистка регистров MR и установка битов округления перед умножением с накоплением.

Биты 'ab' для команды квадроумножения с накоплением и пересылкой означают следующее:
Rsd = MR3:0, MR3:0 += Rmd * Rnd => ab = 00
Rsd = MR3:2, MR3:2 += Rmd * Rnd => ab = 01
Rsd = MR1:0, MR1:0 += Rmd * Rnd => ab = 10

В таблице B-8 приведен синтаксис, а также значения полей типа (TYPE) и кода операции для команд умножителя.

Таблица B-8 – Синтаксис и коды операции команд умножителя

Синтаксис	Rs = ?	Тип	Код операции
Rs = Rm * Rn		0000	xISTy[1]
Rsd = Rm * Rn		0001	xI10y
Rsd = Rmd * Rnd		0010	UIST0
Rsq = Rmd * Rnd		0011	UI100
FRs = Rm * Rn[2]		1111	100T0
FRsd = RmH * RnH, RmL * RnL[3]		1111	000T0
FRsd = RmH * RnL, RmL * RnH[3]		1111	000T0
FRsd = RmL * RnH, RmH * RnL[3]		1111	000T0
FRsd = RmL * RnL, RmH * RnH[3]		1111	000T0
FRsd = RmH * Rn, RmL * Rn[4]		1111	001T0
FRsd = RmH * -Rn, RmL * Rn[4]		1111	001T0

Синтаксис	Rs = ?	Тип	Код операции
FRsd = RmH * Rn, RmL * -Rn[4]		1111	001T0
FRsd = Rmd * Rnd		1111	100T1
Rs = MRa, MRa += Rm * Rn		0100	UICRa[5]
Rsd = MRa, MRa += Rm * Rn		0101	UIC0a
Rs = MRa, MRa += Rm ** Rn		1000	0ICRa
Rsd = MRa, MRa += Rm ** Rn		1001	0IC0a
Rs = MRa, MRa += Rm ** Rn (J)		1010	0ICRa
Rsd = MRa, MRa += Rm ** Rn (J)		1011	0IC0a
Rsd = MRa, MRa += Rmd * Rnd		1100	UIC0a
Rsd = MR3:0, MR3:0 += Rmd * Rnd (CR)		1100	U0111
Rsd = MR3:0, MR3:0 += Rmd * Rnd (C)		1100	U0110
Rsd = MR3:0, MR3:0 += Rmd * Rnd		1100	U0010
Rsd = MR3:0, MR3:0 += Rmd * Rnd (I)		1100	U1010
Rsd = MR3:0, MR3:0 += Rmd * Rnd (C)(I)		1100	U1110
MRa += Rm * Rn	Rs = 00000	1101	UICRa
MRa -= Rm * Rn	Rs = 00001	1101	UICRa
MRa += Rmd * Rnd	Rs = 00010	1101	UIC0a
MR3:0 += Rmd * Rnd	Rs = 00011	1101	UICR0
MRa += Rm ** Rn	Rs = 00100	1101	0ICRa
MRa -= Rm ** Rn	Rs = 00101	1101	0ICRa
MRa += Rm ** Rn(J)	Rs = 00110	1101	0ICRa
MRa -= Rm ** Rn(J)	Rs = 00111	1101	0ICRa
MRa = Rmd	Rs = 01000	1101	0000a
MR4 = Rm	Rs = 01100	1101	00000
MRa = Rmd(se/ze)[6]	Rs = 01001	1101	U000a
SMRa = Rmd(se/ze)	Rs = 01010	1101	U000a
LMRa = Rmd(se/ze)	Rs = 01011	1101	U000a
Rsd = MRa	Rm = 00000	1110	U0S0a
SRsd = MRa	Rm = 00001	1110	U0S0a
Rsq = MR3:0	Rm = 00010	1110	U0S00
Rs = MR4	Rm = 00011	1110	00001
Rs = COMPACT MRa	Rm = 00100	1110	UIS0a
SRsd = COMPACT MR3:0	Rm = 00100	1110	UIS10
Rsq = SMRa	Rm = 01000	1110	U000a
LRsq = MRa	Rm = 01001	1110	U000a
QRsq = LMRa	Rm = 01010	1110	U000a
Rsd = SMR3[7]	Rm = 01100	1110	00011
Rsd = SMR2	Rm = 01100	1110	U0010
Rsd = SMR1	Rm = 01100	1110	U0001
Rsd = SMR0	Rm = 01100	1110	U0000

Синтаксис	Rs = ?	Тип	Код операции
LRsd = MR3	Rm = 01101	1110	U0011
LRsd = MR2	Rm = 01101	1110	U0010
LRsd = MR1	Rm = 01101	1110	U0001
LRsd = MR0	Rm = 01101	1110	U0000

---

[1] Бит S (от названия опции Saturation – насыщение) – это всегда бит 2 поля кода операции.
Биты 'x' и 'y' кодируют опции (U) и (nU) следующим образом:
если xy = 00, то оба операнда знаковые, что соответствует синтаксису ();
если xy = 01, то Rm – знаковый, а Rn – беззнаковый, что соответствует синтаксису (nU);
если xy = 10, то оба операнда беззнаковые, что соответствует синтаксису (U).

[2] Для команд плавающей точки бит 4 кода операции равен 1.

[3] Векторная операция умножения ПЗ одинарной точности. Поэлементное умножение. Две пары операндов перемножаются и формируют две результата. Если значение регистра в поле Rm или Rn нечетное, то это вызывает перестановку операндов.

[4] Векторная операция множения ПЗ одинарной точности. В зависимости от того, четное значение регистра Rm или нет, возможна инверсия знака.

Rm	Rn	функция
четное	четное	Умножение каждого операнда регистровой пары Rmd на значение регистра Rn
четное	нечетное	Умножение каждого операнда регистровой пары Rmd на значение регистра Rn
нечетное	четное	Умножение старшего(левого) операнда регистровой пары Rmd на значение регистра -Rn. Умножение младшего(правого) операнда регистровой пары Rmd на значение регистра Rn
нечетное	нечетное	Умножение старшего(левого) операнда регистровой пары Rmd на значение регистра Rn. Умножение младшего(правого) операнда регистровой пары Rmd на значение регистра -Rn

[5] Бит 'a' определяет пару MR для команды следующим образом:
если a = 0, то используется MR1:0;
если a = 1, то MR3:2

[6] Загрузка регистров MR осуществляется с распространением знака или нуля в регистр MR4, при этом что именно распространяется, определяет бит U:
если данные знаковые (U = 0), то распространяется знак;
если беззнаковые (U = 1), то распространяется ноль.
Тип данных (обычные/короткие/длинные) определяется битами 1:0 поля Rs.

[7] В командах, которые оперируют одиночным регистром MR, выбор регистра MR определяется значениями битов 1:0 кода операции: MR3 (11), MR2 (10). MR1 (01) и MR0 (00).

B.6 Команды сдвигового устройства

Синтаксис и коды операции команд сдвигового устройства описывается в следующих разделах:

- Операнды – одиночные обычные слова, результаты – одиночные регистры;
- Операнды – одиночные длинные или двойные обычные слова, результаты – двойные регистры;
- Операнды – короткие или байтовые, результаты - одиночные или двойные регистры;
- Одиночный операнд.

Почти все команды сдвигового устройства используют формат, альтернативный стандартному формату команд вычислительного блока. Альтернативный формат изображен на рисунке B-3, стандартный – на рисунке B-2. Команды, использующие стандартный формат, помечены как таковые в таблицах B-9, B-10, B-11, B-12 и B-13; остальные команды сдвигового устройства используют альтернативный формат, изображенный на рисунке B-3.

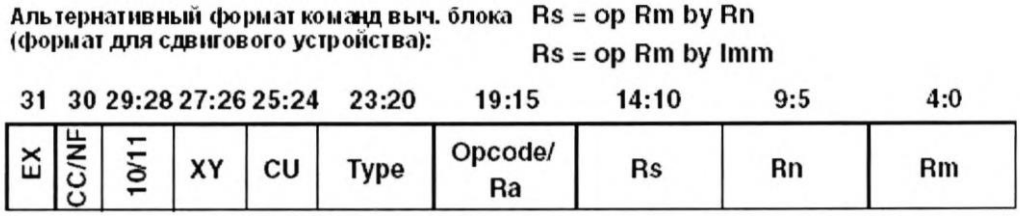


Рисунок B-3 – Альтернативный формат команд вычислительного блока (формат команд сдвигового устройства)

Операнды – одиночные обычные слова, результаты – одиночные регистры

В таблице B-9 перечислены команды сдвигового устройства, использующие в качестве операндов одиночные обычные слова и помещающие результат в одиночные регистры. Здесь приведен синтаксис команд, а также значения полей типа и кода операции для них.

Таблица B-9 – Операнды – одиночные обычные слова, результаты – одиночные регистры

Синтаксис	Тип	Код операции		Комментарии
Rs = LSHIFT Rm BY Rn	0000	000	xx	См.[1]
Rs = LSHIFT Rm BY <Imm6>	0000	001	xi5[2]	
Rs = ASHIFT Rm BY Rn	0000	010	xx	Нет опций
Rs = ASHIFT Rm BY <Imm6>	0000	011	xi5[2]	
Rs = ROT Rm BY Rn	0001	000	xx	Нет опций
Rs = ROT Rm BY <Imm6>	0001	001	0i5[2]	
Rs = FEXT Rm BY Rn	0001	010	x S0	S0 = 1 распр. знака (SE)
Rs = FEXT Rm BY Rnd	0001	011	x S0	
Rs = FDEP Rm BY Rn	0010	000	S1 S0	S0 = 1 распр. знака
Rs += FDEP Rm BY Rnd	0010	001	S1 S0	S1 = 1 заполн. нулями ZF
Rs += MASK Rm BY Rn	0010	010	xx	Нет опций

[1] Величина выполняемого операцией сдвига [определяется размером данных результата и] задается в следующих битах регистра Rn:

Байт: [4:0];
Короткое слово: [5:0];
Обычное слово: [6:0];
Длинное слово: [7:0].

[2] LSB поля кода операции используется как бит 5 шестибитного непосредственного значения.

Операнды – одиночные длинные или двойные обычные слова, результаты – двойные регистры

В таблице В-10 перечислены команды сдвигового устройства, использующие в качестве операндов одиночные длинные слова или двойные обычные слова и помещающие результат в двойные регистры. Здесь приведен синтаксис команд, а также значения полей типа и кода операции для них.

Примечания:

1 Одиночные длиннословные операнды имеют префикс L, как в команде LRsd = Rmd + Rnd.

2 Двойные слоеные операнды не имеют префиксов, как Rsd = Rmd + Rnd.

3 LSB поля Rn* (бит[0] команды) используется для декодирования размера операнда, причем LSB = 0 определяет размер данных как размер двойных обычных слов, а LSB = 1 определяет размер данных как размер длинных слов.

Таблица В-10 – Операнды – одиночные длинные или двойные обычные слова, результаты – двойные регистры

Синтаксис	Тип	Код операции	
(L)Rsd = LSHIFT Rmd BY Rn	0100	000	xx
(L)Rsd = LSHIFT Rmd BY <Imm7>	0100	001	i6 i5[1]
SRsd = COMPACT Rmd BY <Imm7>[2]	0100	001	i6 i5[1]
(L)Rsd = ASHIFT Rmd BY Fin	0100	010	Xx
(L)Rsd = ASHIFT Rmd BY <Imm7>	0100	011	i6 i5[1]
SRsd = COMPACT Rmd BY <Imm7>[2]	0100	001	i6 i5[1]
(L)Rsd = ROT Rmd BY Rn	0101	000	Xx
(L)Rsd = ROT Rmd BY <Imm7>	0101	001	i6 i5[1]
LRsd = FEXT Rmd BY Rn	0101	010	x S0
LRsd = FEXT Rmd BY Rnd	0101	011	x S0
Rsd = GETBITS Rmq BY Rnd[3]	0101	100	x S0
LRsd += FDEP Rmd BY Rn	0110	000	S1 S0
LRsd += FDEP Rmd BY Rnd	0110	001	S1 S0
LRsd += MASK Rmd BY Rnd	0110	010	xx
Rsd += PUTBITS Rmd BY Rnd[3]	0110	100	xx

---

1 Два бита LSB кода операции используются как биты 6:5 семибитного непосредственного значения.

2 Сдвиг вправо с упаковкой в меньший формат представления. 30 бит команды всегда должен быть 1 (NF). Выполняется только после префикса 0x33C00018 для операции без округления, либо префикса 0x33C00019 для операции с округлением.

3 Использует стандартный формат вычислительного блока, изображенный на рисунке В-2.

Операнды – короткие или байтовые, результаты - одиночные или двойные регистры

В таблице В-11 перечислены команды сдвигового устройства, использующие в качестве операндов короткие слова или байты и помещающие результат в одиночные либо двойные регистры. Здесь приведен синтаксис команд, а также значения полей типа и кода операции для них.

Примечания:

1 Двойные короткословные операнды имеют префикс S, как в команде SRs = Rm+ Rn.

2 Квадро-короткословные операнды имеют префикс S и суффикс d, как в команде SRsd = Rmd + Rnd.
3 Байтовые операнды имеют префикс B, как в команде BRs = Rm + Rn.
4 Октобайтовые операнды имеют префикс B и суффикс d, как в команде BRsd = Rmd + Rnd.
5 LSB поля Rn* (бит[0] команды) используется для декодирования размера операнда, причем LSB = 0 определяет размер данных как размер байтов, а LSB = 1 определяет размер данных как размер коротких слов.

Таблица B-11 – Операнды – короткие или байтовые, результаты – одиночные или двойные регистры

Синтаксис	Тип	Код операции	S1 S0
SRs = LSHIFT Rm BY Rn	1000	000	xx
SRs = LSHIFT Rm BY <Imm5>	1000	001	
SRs = ASHIFT Rm BY Rn	1000	010	xx
SRs = ASHIFT Rm BY <Imm5>	1000	011	
BRs = LSHIFT Rm BY Rn	1001	000	xx
BRs = LSHIFT Rm BY <Imm4>	1001	001	
BRs = ASHIFT Rm BY Rn	1001	010	xx
BRs = ASHIFT Rm BY <Imm4>	1001	011	
(S/B)Rsd = LSHIFT Rmd BY Rn	1010	000	xx
(S/B)Rsd = LSHIFT Rmd BY <Imm5>	1010	001	
BRsd = COMPACT SRmd BY <Imm5>[1]	1010	001	
(S/B)Rsd = ASHIFT Rmd BY Rn	1010	010	xx
(S/B)Rsd = ASHIFT Rmd BY <Imm5>	1010	011	
BRsd = COMPACT SRmd BY <Imm5>[1]	1010	011	

[1] Сдвиг вправо с упаковкой в меньший формат представляения. 30 бит команды всегда должен быть 1 (NF). Выполняется только после префикса 0x33C00018 для операции без округления, либо префикса 0x33C00019 для операции с округлением.

Одиночный операнд

Формирование кода операции для следующих команд с одиночным операндом отличается от формирования кода операции для остальных команд сдвигового устройства и определяется следующими правилами:

- бит [0] – используется как бит i5 непосредственного значения <Imm6 >;
- бит [1] – устанавливает, выполняется ли операция над одиночным или двойным регистром;
- бит [2] – использование регистрового файла или непосредственного значения Биты [4:3] – проверка, установка, сброс и переключение [бита].

В таблице B-12 приведен синтаксис, а также значения полей типа и кода операции для команд сдвигового устройства с одиночным операндом.

Таблица B-12 – Синтаксис и коды операции для команд сдвигового устройства (одиночный операнд)

Синтаксис	Тип	Код операции
BITEST Rm BY Rn	1011	1100x
BITEST Rm BY <Imm5>	1011	1110x
BITEST Rmd BY Rn	1011	1101 x
BITEST Rmd BY <Imm6>	1011	1111i5
Rs = BCLR Rm BY Rn	1011	0000x

Синтаксис	Тип	Код операции
Rs = BCLR Rm BY<Imm5>	1011	0010x
Rs = BSET Rm BY Rn	1011	0100x
Rs = BSET Rm BY <Imm5>	1011	0110x
Rs = BTGL Rm BY Rn	1011	1000x
Rs = BTGL Rm BY <Imm5>	1011	1010x
Rsd = BCLR Rmd BY Rn	1011	0001x
Rsd = BCLR Rmd BY <Imm6>	1011	0011i5
Rsd = BSET Rmd BY Rn	1011	0101x
Rsd = BSET Rmd BY <Imm6>	1011	0111i5
Rsd = BTGL Rmd BY Rn	1011	1001x
Rsd = BTGL Rmd BY <Imm6>	1011	1011i5
Rs = LDO Rm[1]	1100	00000
Rs = LDO Rmd[1]	1100	00010
Rs = LD1 Rm[1]	1100	00001
Rs = LD1 Rmd[1]	1100	00011
Rs = EXP Rm[1]	1100	00100
Rs = EXP Rmd[1]	1100	00110
X/YSTAT = Rm[1]	1100	01000
Rs = X/YSTAT[1]	1100	01001
X/YSTATL = Rm[1]	1100	01110
BKFPT Rmd, Rnd[1]	1100	01111
Rsd = BFOTMP[1]	1100	01010
BFOTMP = Rmd[1]	1100	01011

[1] Использует стандартный формат команд вычислительного блока, изображенный на рисунке В-2.

Регистры CLU

Формирование кода операции для следующих команд загрузки регистров CLU отличается от формирования кода операции для остальных команд сдвигового устройства и определяется следующими правилами:

- биты [2:0] = 000;
- биты [4:3] = NLQ; определяет размер данных:
  00 – обычные слова (32 бита);
  10 – длинные слова (64 бита);
  11 – квадрослова (128 бит);
- биты [9:7] = Rm(d)(q);
- биты [14:10] = Коды операции;
- биты [18:15] = TRs(d)(q) [4:1], THRs(d)(q) [4:1] с битами [4:3] для регистров THRx всегда равными 00 или 1111;
- бит [19] = TRs(d)(q) [0], THRs(d)(q) [0] или 1

В таблице В-13 приведен синтаксис, а также значения полей типа и кода операции для команд сдвигового устройства загрузки регистров CLU.

Таблица В-13 – Синтаксис и коды операции для команд сдвигового устройства (загрузка регистров CLU)

Синтаксис	Тип	Код операции
TRs(d)(q) = Rm(d)(q)1	1111	00000
THRs(d)(q) = Rm(d)(q)(I)[1, 2]	1111	00011
CMCTL = Rm[1]	1111	00010

[1] Использует стандартный формат команд вычислительного блока, изображенный на рисунке В-2

[2] I – признак чередования.

B.7 Команды IALU

Синтакис и коды операции команд IALU описываются в следующих разделах:
- Формат (целочисленной) команды IALU;
- Формат команды пересылки IALU;
- Формат команды загрузки данных IALU;
- Формат команды загрузки/сохранения IALU;
- Формат непосредственного расширения для IALU.

Формат (целочисленной) команды IALU

Формат обычных команд IALU характеризуется рисунком B-4 и таблицей B-14.

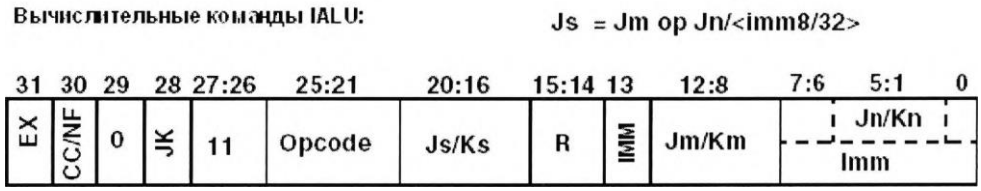


Рисунок B-4 – Формат команды IALU (целочисленной)

Таблица B-14 – Поля кода операции команд IALU (целочисленной)

Биты	Поле	Описание
7-0	Jn/Kn/Imm	Jn/Kn или непосредственное значение является вторым операндом команды. Какой именно вариант используется: Jn/Kn или непосредственное значение – определяется битом IMM: при IMM = 0 биты [5:1] указывают номер регистра; при IMM = 1 операнд является выровненным вправо 8-битным непосредственным значением в дополнительном коде. Однако, если в той же командной строке имеется непосредственное расширение для того же IALU, то операнд является 32-битным непосредственным значением в дополнительном коде
12-8	Jm/Km	Первый операнд команды
13	IMM	Определяет второй операнд: 0 = устанавливает, что второй операнд – регистр; 1 = устанавливает, что второй операнд – непосредственное значение
15-14	Xcode	Дополнительное поле кода операции
20-16	Js/Ks	Результирующий регистр: номер одного из 32-х регистров в регистровом файле IALU, куда должен быть помещен результат
25-21	Opcode	Код операции: см. таблицу B-14
27-26	11	Определяет, что это обычная команда IALU
28	JK	Определяет исполнительное устройство IALU: 0 = устанавливает, что исполнительное устройство – J-IALU; 1 = устанавливает, что исполнительное устройство – K-LALU
29	0	Определяет, что операция должна исполняться IALU

Биты	Поле	Описание
30	CC/NF	Для командных строк, начинающихся с условных команд устройства управления (if <предикат>...), этот бит интерпретируется как бит CC. В противном случае этот бит интерпретируется как бит NF. В случае командной строки, начинающейся с условной команды, действие бита NF, присутствующего в этой команде, распространяется на всю командную строку. Применимо к командам вычислительного блока, IALU и к командам загрузки/сохранения. Неприменимо к условным командам, непосредственным расширениям и др. (см. подраздел «Формат команды косвенного перехода (УУ)»). Если бит 30 является битом CC (признаком условности), то он, будучи установлен, обуславливает исполнение команды значением предиката в командной строке. Если бит 30 является битом NF (не обновлять флаги), то когда он установлен, текущая команда не обновляет значения флагов по завершению исполнения этой команды
31	EX	Установленный бит EX определяет команду как последнюю в командной строке

Таблица В-15 – Коды целочисленных операций IALU

Синтаксис	Opcode	Xcode
Js = Jm + Jn	00000	00
Js = Jm - Jn	00001	00
Js = Jm + Jn (CJMP)	00010	00
Js = Jm - Jn (CJMP)	00011	00
Js = Jm + Jn (CB)	00100	00
Js = Jm - Jn (CB)	00101	00
Js = Jm + Jn (BR)	00110	00
Js = BITREV Jm (поле Jn должно быть равно 31)	00110	01
Js = Jm - Jn (BR)	00111	00
JB3:0/JL3:0 = Jm + Jn	01000	00
JB3:0/JL3:0 = Jm - Jn	01001	00
Js = (Jm + Jn)\2	01010	00
Js = (jm - jn)/2	01011	00
COMP(Jm,Jn) signed	01100	00
COMP(Jm,Jn) unsigned	01101	00
Js MIN(Jm,Jn)	01110	00
Js = MAX(Jm,Jn)	01111	00
Js = Jm + Jn +Jc	10000	00
Js = Jm - Jn +Jc -1	10001	00
Js = Jm OR Jn	10010	00
Js = Jm AND Jn	10011	00
Js = Jm XOR Jn	10100	00
Js = Jm AND NOT Jn	10101	00
Js = NOT Jm	11000	00
Js = ABS Jm	11001	00
ASHIFTR = Jm	11010	00
LSHIFTR - Jm	11011	00

Синтаксис	Opcode	Xcode
ROTR = Jm	11100	00
ROTL = Jm	11101	00
Js = LSHIFTL Jm by Imm5	11111	01
Js = ASHIFTR Jm by Imm5	11111	10
Js = LSHIFTR Jm by Imm5	11111	11

Формат команды пересылки IALU

Формат команд пересылок регистров IALU характеризуется рисунком В-5 и таблицей В-16.

Рисунок В-5 – Формат команд пересылок регистров IALU

Таблица В-16 – Поля кода операции команд пересылок регистров IALU

Биты	Поле	Описание
0	Q	Указание на размер данных: длинные или квадрослова. Если LQ сброшен, т.е. данные - словные, бит[0] не используется; в противном случае: 1 = указывает на то, что данные должны быть квадрословами (128 бит); 0 = указывает на то, что данные должны быть длинными словами (64 bits)
5-1	UREG-приемник	Определяет регистр-приемник
6	R	Зарезервировано. Бит должен быть равен нулю
7	SG5	Старший бит (MSB) группы источника (Source Group)
13-8	Группа прием.	Определяет группу приемника
14	LQ	Указание на размер данных. Размер регистра должен соответствовать типу данных. Например, если данные квадрослова, то размер транзакции должен быть кратен четырем, а если данные – длинные слова, то размер транзакции должен быть кратен двум: 1 = указывает на то, что данные должны быть длинными или квадрословами; 0 = указывает на то, что данные должны быть обычными словами
15	0	Указывает на то, что данная команда является пересылкой регистров Ureg.
20-16	Ureg-источник	Определяет регистр-источник
25-21	Группа источ.	Определяет пять младших битов ([4:0]) поля, которое определяет группу источника. Бит[5] поля определяется битом SG5 (бит [7] команды)
27-26	10	Определяет тип команды как операции пересылки регистров

Биты	Поле	Описание
28	JK	Определяет исполнительное устройство IALU: 0 = устанавливает, что исполнительное устройство – J-IALU; 1 = устанавливает, что исполнительное устройство – K-IALU
29	0	Определяет, что данная операция должна исполняться устройством IALU
30	CC/NF	Для командных строк, начинающихся с условных команд устройства управления (if <предикат>...), этот бит интерпретируется как бит CC. В противном случае этот битинтерпретируется как бит NF. В случае командной строки, начинающейся с условной команды, действие бита NF, присутствующего в этой команде, распространяется на всю командную строку. Применимо к командам вычислительного блока, IALU и к командам загрузки/сохранения. Неприменимо к условным командам, непосредственным расширениям и др. (см. подраздел «Формат команды косвенного перехода (УУ)»). Если бит 30 является битом CC (признаком условности), то он, будучи установлен, обуславливает исполнение команды значением предиката в командной строке. Если бит 30 является битом NF (не обновлять флаги), то когда он установлен, текущая команда не обновляет значения флагов по завершению исполнения этой команды
31	EX	Установленный бит EX определяет команду как последнюю в командной строке

Формат команды загрузки данных IALU

Формат команд загрузки регистров характеризуется рисунком В-6 и таблицей В-17. Данный формат достоверен только для определенных значений в поле «Группа». Код группы не должен быть равен 1, 3, 5 или 7.

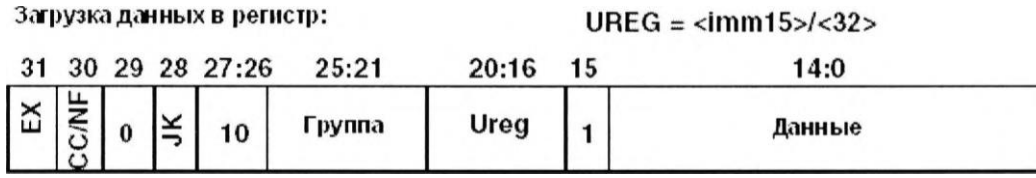


Рисунок В-6 – Формат команд загрузки данных IALU

Таблица В-17 – Формат команд загрузки данных IALU

Биты	Поле	Описание
14-0	Данные	Определяет 15 битов знаковых данных, подлежащих загрузке
15	1	Указывает на то, что данная команда является командой загрузки данных в Ureg-регистр
20-16	Ureg	Определяет регистр-приемник
25-21	Группа	Определяет группу регистра-приемника. Из всего набора кодов только ограниченное количество кодов является допустимым. Недопустимыми являются коды 1, 3, 5, 7
27-26	10	Определяет тип команды как команды загрузки константы в регистр
28	JK	Определяет исполнительное устройство IALU:

Биты	Поле	Описание
		0 = устанавливает, что исполнительное устройство – J-IALU; 1 = устанавливает, что исполнительное устройство – K-IALU
29	0	Определяет, что данная операция должна исполняться устройством IALU
30	CC/NF	Для командных строк, начинающихся с условных команд устройства управления (if <предикат>...), этот бит интерпретируется как бит CC. В противном случае этот бит интерпретируется как бит NF. В случае командной строки, начинающейся с условной команды, действие бита NF, присутствующего в этой команде, распространяется на всю командную строку. Применимо к командам вычислительного блока, IALU и к командам загрузки/сохранения. Неприменимо к условным командам, непосредственным расширениям и др. (см. подраздел «Формат команды косвенного перехода (УУ)»). Если бит 30 является битом CC (признаком условности), то он, будучи установлен, обуславливает исполнение команды значением предиката в командной строке. Если бит 30 является битом NF (не обновлять флаги), то когда он установлен, текущая команда не обновляет значения флагов по завершению исполнения этой команды
31	EX	Установленный бит EX определяет команду как последнюю в командной строке

В зависимости от поля «Группа» выполняется загрузка в регистр модуля:
0 – X
2 – Y
8 – X и Y
10 и 27 – дебаггера
12 – J
13 – K
14– JB, JL
15 – KB, KL
26 – секвенсера
30 и 31 – защиты

Все другие коды группы являются резервными и не должны использоваться в командах загрузки констант.

Формат команды константы вычислительного модуля XY

Данный тип команды выполняется на конвейере целочисленнооого АЛУ и формирует константу которая передается в вычислительный модуль ХУ для окончательного выполнения операции. Формат команд характеризуется таблицей В-18.

Таблица В-18 – Формат команды константы

Биты	Поле	Описание
14-0 18-16 30	K[18:0]	Определяет 19 битов данных для формирования константы

Биты	Поле	Описание
15	1	Должен быть равен 1
20-19	YY	Определяют к какой ветви конвейера в вычислительном модуле Y относится константа
21	1	Должен быть равен 1
23-22	XX	Определяют к какой ветви конвейера в вычислительном модуле X относится константа
27:24	1 0 0 0	Код команды
28	JK	Определяет исполнительное устройство IALU на конвейере которого формируется константа для передачи в вычислительный модуль: 0 = - J-IALU 1 = - K-IALU
29	0	Должен быть равен 0
31	EX	Установленный бит EX определяет команду как последнюю в командной строке

Формат команды загрузки/сохранения IALU

Данный формат команд предназначен для работы с 32-, 64- и 128-разрядными словами.

Формат команд загрузки с использованием регистров IALU характеризуется рисунком В-7 и таблицей В-19.

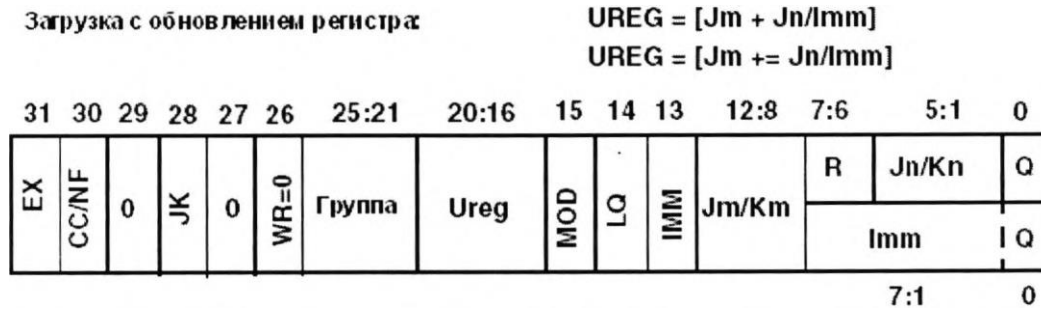


Рисунок В-7 – Формат команд загрузки IALU

Формат команд сохранения с использованием регистров IALU характеризуется рисунком В-8 и таблицей В-19.

Рисунок В-8 – Формат команд сохранения IALU

Таблица В-19 – Поля кода операции команд загрузки/сохранения IALU

Биты	Поле	Описание
0	Q	Указание на размер данных: длинные или квадрослова. Если LQ сброшен, [т.е. данные - словные,] бит[0] не используется тогда и только тогда, когда в команде употреблена операци над индексами [Jm +|+= Jn]. Если же употреблена операция [Jm +|+= Imm], то бит[0] используется как LSB непосредственного значения. Если же LQ установлен, то бит [0] непосредственного значения предполагается равным 0; в противном случае: 1 = указывает на то, что данные должны быть квадрословами (128 бит); 0 = указывает на то, что данные должны быть длинными словами (64 бит)
5-1	JN/KN	Определяет второй адрес в J-/K-IALU
7-6	R	Зарезервировано
12-8	JM/KM	Определяет первый адрес в J-/K-IALU. Для команд альтернативного доступа биты 12-10 определяет группу доступа (см. описание в таблице С-20)
13	Imm	Определяет, используется ли в команде обновление регистра (0) или непосредственное значение (1)
14	LQ	Указание на размер данных. См. также описание альтернативного доступа: 1 = указывает на то, что данные должны быть длинными или квадрословами; 0 = указывает на то, что данные должны быть обычными словами
15	MOD	Указание на режим модификации: 0 = определяет, что операция должна быть предмодификацией без обновления (+); 1 = определяет, что операция должна быть постмодификацией с обновлением (+=)
20-16	Ureg	Определяет регистр приемника/источника
25-21	Группа	Определяет группу регистров для приемника (для загрузки) или источника (для сохранения)
26	WR	Различает операции загрузки и сохранения: 0 = определяет операцию загрузки - пересылки данных из памяти в регистр-приемник; 1 = определяет операцию сохранения - пересылки данных из регистра-источника в память
27	0	Определяет тип команды как команды загрузки/сохранения с операцией обновления регистра
28	JK	Определяет исполнительное устройство IALU: 0 = устанавливает, что исполнительное устройство – J-IALU; 1 = устанавливает, что исполнительное устройство – K-IALU
29	0	Определяет, что данная операция должна исполняться устройством IALU

Биты	Поле	Описание
30	CC/NF	Для командных строк, начинающихся с условных команд устройства управления (if <предикат>...), этот бит интерпретируется как бит CC. В противном случае этот бит интерпретируется как бит NF. В случае командной строки, начинающейся с условной команды, действие бита NF, присутствующего в этой команде, распространяется на всю командную строку. Применимо к командам вычислительного блока, IALU и к командам загрузки/сохранения. Неприменимо к условным командам, непосредственным расширениям и др. (см. подраздел «Формат команды косвенного перехода (УУ)»). Если бит 30 является битом CC (признаком условности), то он, будучи установлен, обуславливает исполнение команды значением предиката в командной строке. Если бит 30 является битом NF (не обновлять флаги), то когда он установлен, текущая команда не обновляет значения флагов по завершению исполнения этой команды
31	EX	Установленный бит EX определяет команду как последнюю в командной строке

Запрос альтернативного доступа операциями загрузки/сохранения идентифицируется в поле группы (пять битов) и может быть выполнен только для регистров Jm/Km0:3 и регистров *Uregs* вычислительного блока. Это освобождает три бита в поле операнда Jm/Km для кода альтернативного доступа. Группы регистров альтернативного доступа вычислительного блока таковы:

- группа 1 = группа 0 с альтернативным доступом (к вычислительному блоку X);
- группа 3 = группа 2 с альтернативным доступом (к вычислительному блоку Y);
- группа 5 = группа 4 с альтернативным доступом (к вычислительному блоку X и Y – совмещенным);
- группа 7 = группа 6 с альтернативным доступом (к вычислительному блоку Y и X – совмещенным);
- группа 9 = группа 8 с альтернативным доступом (к вычислительному блокуX и Y – широковещательным).

При альтернативном доступе (с использованием кольцевого буфера, бит-реверсивной адресацией, DAB или SDAB) поле операнда Jm/Km (биты [12:8]) декодируется по-другому и дополнительно определяет тип доступа. Декодирование поля описано в таблице B-20, а синтаксис ассемблера для различных типов доступа описаны в таблице B-21.

Таблица B-20 – Поля кода операции команд загрузки/сохранения IALU с альтернативными типами доступа (с использованием кольцевого буфера, бит-реверсивной адресацией, DAB или SDAB)

Биты	Поле	Описание
9-8	Jm/Km	Определяет регистры Jm/Km в диапазоне между 0 и 3
12-10	Группа доступа	Операции с кольцевым буфером могут быть с использованы только с постмодификацией адреса. Дополнительную информацию см. в разделе «Адресация кольцевых буферов». Группы доступа таковы:

		000 = определяет обычный доступ к кольцевому буферу (CB); 001 = устанавливает опцию бит-реверсивной; 010 = устанавливает использование DAB со смещением на обычное слово и доступом к кольцевому буферу; 011 = устанавливает использование DAB со смещением на короткое слово и доступом к кольцевому буферу; 100 = устанавливает использование DAB со смещением для X на одно слово большим, чем смещение на обычное слово, и со смещением для Y, равным смещению на обычное слово, а также с доступом к кольцевому буферу; 101 = устанавливает использование SDAB со смещением X на одно короткое слово больше, чем смещение Y; 110 = устанавливает использование DAB со смещением для Y на одно слово большим, чемсмещение на обычное слово, и со смещением для X, равным смещению на обычное слово, а также с доступом к кольцевому буферу; 111= устанавливает использование SDAB со смещением Y на одно короткое слово больше, чем смещение X

Таблица B-21 – Синтаксис ассемблера команд загрузки/сохранения IALU с альтернативными типами доступа (с использованием кольцевого буфера, бит-реверсивной адресацией, DAB или SDAB)

Опция	Синтаксис ассемблера
Кольцевой буфер	xyRsq = CB q[Jm += Jn]
DAB и кольцевой буфер	xyRsq = DAB q[Jm += Jn]
DAB с коротким выравниванием и кольцевой буфер	xyRsq = SDAB q[Jm += Jn]
Бит-реверсивная адресация	xyRsq = BR q[Jm += Jn]
Загрузка слова или двойного слова по указателю на байт (быстрый вариант) (префикс PI8 и код команды загрузки)	xyRs(d) = BDAB (L)[Jm += Jn]
Загрузка слова или двойного слова по указателю на байт (стандартный вариант) (префикс PI8U и код команды загрузки)	xyRs(d) = (L)PB [Jm += Jn]
Загрузка слова или двойного слова по указателю на короткое (быстрый вариант) (префикс PI16 и код команды загрузки)	xyRs(d) = SDAB (L)[Jm += Jn]
Загрузка слова или двойного слова по указателю на короткое (стандартный вариант) (префикс PI16U и код команды загрузки)	xyRs(d) = (L)PS [Jm += Jn]
Сохранение слова по указателю на короткое	PS [Jm += Jn] = Ureg

Формат команды IALU для загрузки/сохранения байт и коротких чисел

Данный формат команд предназначен для работы с байтами и короткими (16 бит) словами.

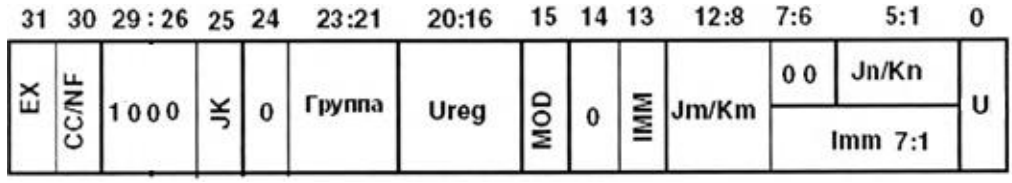


Рисунок В-9 – Загрузка байта

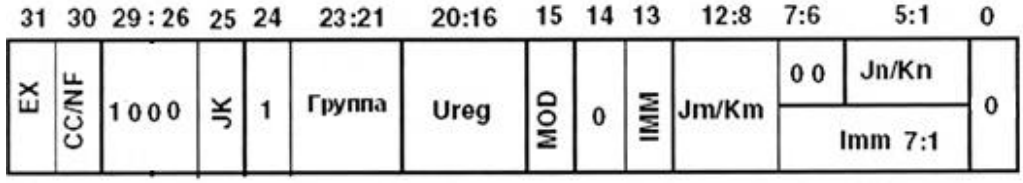


Рисунок В-10 – Сохранение байта

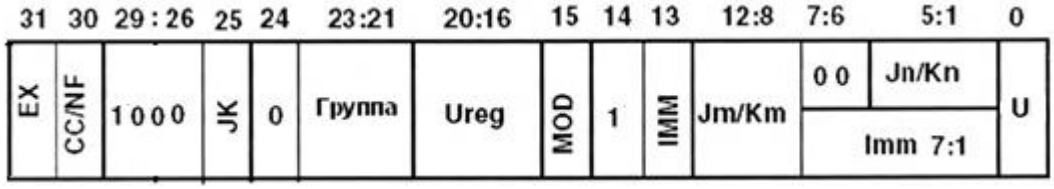


Рисунок В-11 – Загрузка короткого

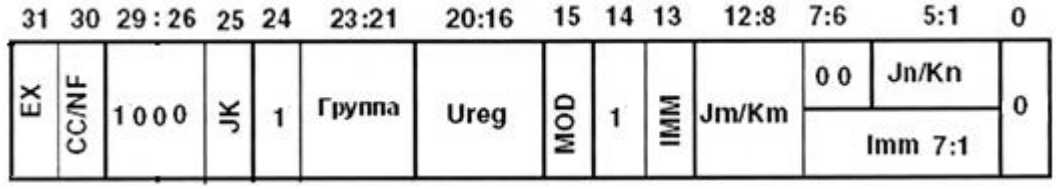


Рисунок В-12 – Сохранение короткого

Биты MOD и IMM аналогичны описанным в таблице В-19. Допустимые значения группы регистров описаны в отдельном документе.

Формат непосредственного расширения для IALU

Формат непосредственного расширения IALU характеризуется рисунками В-13, В-14 и таблицей В-22.

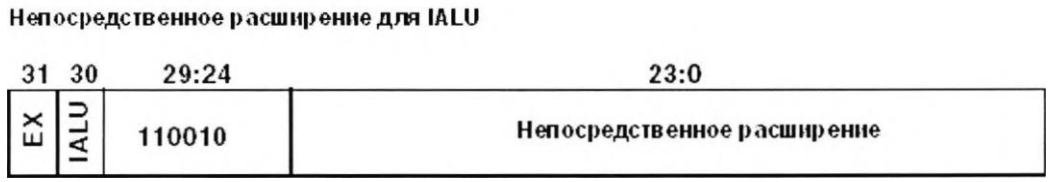


Рисунок В-13 – Формат непосредственного расширения IALU

Таблица В-22 – Поля кода операции непосредственного расширения IALU

Биты	Поле	Описание
23-0	Неп.расш.	Задает значение непосредственного расширения
29-24	110010	Определяет, что данная команда является непосредственным расширением IALU
30	IALU	Указывает исполнительное IALU:

		0 = J-IALU 1 = K-IALU
30	Reserved	Зарезервировано
31	EX	Установленный бит EX определяет команду как последнюю в командной строке

Рисунок B-14

B.8 Команды устройства управления

Синтаксис и коды операции команд устройства управления описываются в следующих разделах:

- Формат команды прямого перехода/вызова (УУ);
- Формат команды косвенного перехода (УУ).

Формат команды прямого перехода/вызова (УУ)

Формат команды прямого перехода/вызова характеризуется рисунком B-14 и таблицей B-23.

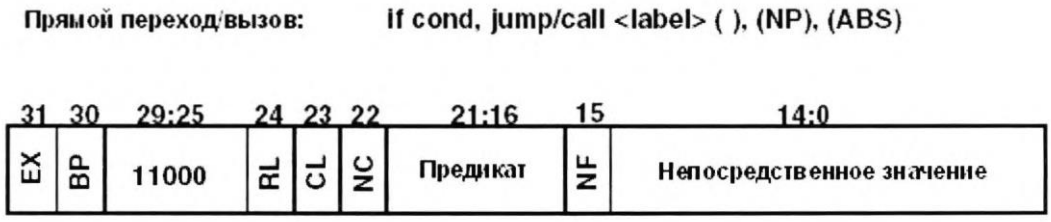


Рисунок B-14 – Формат команды прямого перехода/вызова (УУ)

Таблица B-23 – Поля кода операции команд прямого перехода/вызова (УУ)

Биты	Поле	Описание
14-0	Непосредств. знач.	Задает непосредственное значение
15	NF	Запрещает обновление флагов (для всей командной строки)
21-16	Предикат	Задает предикат для перехода. См. подраздел «Коды предикатов»
22	NC	Определяет отрицание предиката: 1 = означает, что фактический предикат перехода = НЕ указанный предикат
23	CL	Указывает тип команды: 1 = определяет, что команда является вызовом: значение следующего PC записывается в регистр CJMP, задавая тем самым адрес возврата по завершению вызова; 0 = команда является переходом

Биты	Поле	Описание
24	RL	Определяет, является ли адрес абсолютным (0) или относительным (1). 0 = целевой адрес является абсолютным адресом, равным непосредственному значению (биты 14-0), возможно, конкатенированному с непосредственным расширением (если существует); такой способ указания адреса задается опцией (ABS). См. подраздел «Операции устройства управления»; 1 = переход/вызов осуществляется по PC-относительному адресу: целевой адрес равен значению программного счетчика + непосредственное значение
29-25	11000	Определяет, что данная команда является прямым переходом/вызовом
30	BP	Указывает прогнозируемость перехода (Branch Prediction): 1 = бит BP установлен в 1 по умолчанию; при BP = 1 элемент помещается в память BTB. Прогноз заключается в том. что переход предпринимается – прогнозируемый переход; 0 = бит BP сброшен, когда в ассемблерном представлении команда включает опцию NP, т.е. прогноз заключается в том, что переход не предпринимается – непрогнозируемый переход. См. подраздел «Операции устройства управления»
31	EX	Установленный бит EX определяет команду как последнюю в командной строке. Если в командной строке больше, чем одна команда, то бит EX должен быть равен 0, поскольку переходы всегда являются первыми командами в строке

Формат команды косвенного перехода (УУ)

Формат команды косвенного перехода характеризуется рисунком В-15 и таблицей В-24.

Условный косвенный переход **if cond. cjmp/cjmp_call/rti/reti (np).(abs)**
Условное исполнение **if coild; do <любая команда>**
Установка SF: **sfb = psfb mod scond**

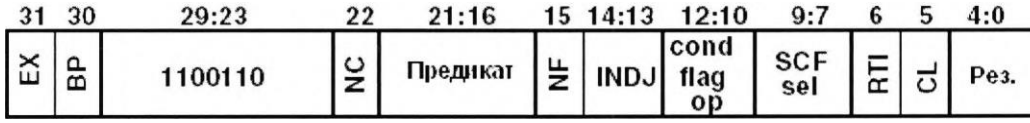


Рисунок В-15 – Формат команды косвенного перехода устройства управления

Таблица В-24 – Поля кода операции команд косвенного перехода (УУ)

Биты	Поле	Описание
4-0	PFX	Поле должно быть установлено в 00000. В случае использования поля в качестве кода префикса, бит 4 должен быть равен 1, а биты 3-0 содержат код. Коды префикса (биты 4-0) приведены в таблице В-29

Биты	Поле	Описание
5	CL	Задает тип переход/вызов: 1 = указывает, что это команда CJMP_CALL, вычисляемый вызов, где следующее значение программного счетчика PC записывается в регистр CJMP, таким образом, указывая адрес возврата по завершению вызова; 0 = Обычный вычисляемый переход
6	RTI	Указывает, что команда является возвратом из прерывания. В этом случае значение регистра PMASK корректируется необходимым образом. Задает предикат перехода см. раздел «Коды предикатов». Примечание – Код операции команды RDS есть OxB3080040, что свидетельствует о том, что RDS обязана быть единственной командой в командной строке. Она действует подобно команде RTI с условием перехода FALSE
9-7	SCF SELECT	Выбирает флаг условия как источник и приемник операции SCF: 000 = управляет флагом SF0; 001 = управляет флагом SF1; <xy>0 = вычислительный блок (X, Y или оба) и SF0; <xy>1 = вычислительный блок (X, Y или оба) и SF1
12-10	COND FLAG OP	Вместе с полями SF select определяет значение результирующего флага условия: SF1/0 += логич_операция знач_предиката. Логич операции таковы: 000 = нет операции; 001 = SF = знач предиката; присваивает знач_предиката флагу SF; 100 = SF += AND знач_предиката; 101 = SF += OR3Нач_предиката 110 = SF += XOR знач_предиката
14-13	INDJ	Определяет тип косвенного перехода: 00 = условная команда; не переход; 01 = RTI/RETI – переход по адресу, задаваемому значением в регистре RETI. Если это команда RTI, то значение регистра PMASK корректируется необходимым образом. 10 = CJMP – вычисляемый переход по абсолютному адресу; PC = CJMP 11 = CJMP – относительный адрес перехода; PC+=CJMP
15	NF	Определяет запрет на обновление флагов (для всей командной строки)
21-16	CONDI TION	Определяет предикат перехода. См. раздел «Коды предикатов»
22	NC	Определяет отрицание предиката: 1 = Фактический предикат = NOT (указанный предикат)
29-23	1100110	Определяет, что это команда косвенного перехода
30	BP	Указывает прогнозируемость перехода (Branch Prediction): 1 = бит BP установлен в 1 по умолчанию; при BP = 1 элемент помещается в память BTB
31	EX	Установленный бит EX определяет команду как последнюю в командной строке. Если в командной строке больше, чем одна команда, то бит EX должен быть равен 0, поскольку переходы всегда являются первыми командами в строке

Базовой условной командой является:

if cond, jump... ; else, <любая команда>

Команда if cond; do <любая команда> использует тот же формат, что и команда перехода. Другие команды в командной строке, которые являются условными в силу того, что условным является переход, выполняются при истинности предиката, являющегося отрицанием предиката, управляющего переходом. Чтобы сохранить читаемость ассемблерного текста при использовании команды «if cond; do <любая команда>;;», предикат, кодируемый в машинном коде, является отрицанием предиката, фигурирующего в ассемблерном тексте.

Два прогнозируемых перехода не могут располагаться в одном квадрослове, выровненном на границу.

B.9 Коды предикатов

Коды предикатов для условных команд описываются в следующих разделах:

- Предикаты вычислительного блока;
- Предикаты IALU;
- Предикаты устройства управления и внешние предикаты.

Предикаты вычислительного блока

Предикат идентифицируется как предикат вычислительного блока, если по крайней мере один из двух MSB его кода ≠ 0, а четверка LSB меньше, чем 1011. Декодирование предикатов приведено в таблице B-25.

Таблица B-25 – Коды предикатов ALU, умножителя и сдвигового устройства

Код	Предикат	
<XY>0000	AEQ	результат ALU равен нулю
<XY>0001	ALT	результат ALU меньше нуля
<XY>0010	ALE	результат ALU меньше или равен нулю
<XY>0011	MEQ	результат умножителя равен нулю
<XY>0100	MLT	результат умножителя меньше нуля
<XY>0101	MLE	результат умножителя меньше или равен нулю
<XY>0110	SEQ	результат сдвигового устройства равен нулю
<XY>0111	SLT	результат сдвигового устройства меньше нуля
<XY>1001	SF0	предикат статического флага #0
<XY>1010	SF1	предикат статического флага # 1
1111XX	предикаты не вычислительного блока	

Поле <XY> декодируется аналогично такому же полю в командах вычислительного блока:

00 = не является предикатом никакого вычислительного блока;
01 = предикат вычислительного блока X;
10 = предикат вычислительного блока Y;
11 = предикат обоих блоков X и Y (исключая 1111XX).

Предикаты IALU

Предикат идентифицируется как предикат IALU, если четыре MSB содержат 000<JK>. Бит <JK> аналогичен такому же биту в командах IALU:

0 = J-IALU;
1 = K-IALU.

Коды предикатов перечислены в таблице B-26.

Таблица B-26 – Коды предикатов IALU

Код	Предикат	
000<JK>00	JEQ/KEQ	результат J-/K-IALU равен нулю
000<JK>01	JLT/KLT	результат J-/K-IALU меньше нуля
000<JK>10	JLE/KLE	результат J-/K-IALU меньше или равен нулю

Предикаты устройства управления и внешние предикаты

Внешними предикатами именуют предикаты, вычисляемые на основании признаков (флагов), не зависящих от ядра. Их идентифицирует значение 001 трех MSB кода предиката.

Таблица В-27 – Предикаты устройства управления и внешние предикаты

Код	Предикат
001000	TRUE
001001	ISF0 – статический флаг #0 IALU
001010	ISF1 – статический флаг #1 IALU
001011	BM – ведущий на шине (Bus Master)
001100	LC0E – счетчик циклов #0 равен нулю
001101	LC1E – счетчик циклов # 1 равен нулю
1111<FLG>	FLAGx_IN (где x = 0..3) – флаг-индикатор входной линии дискретных сигналов 0..3

Формат непосредственного расширения для устройства управления

Формат команды для непосредственных расширений команд перехода (JUMP) характеризуется рисунком В-16 и таблицей В-28.

Рисунок В-16 – Формат непосредственного расширения (для команды JUMP)

Таблица В-28 – Поля кода операции для непосредственного расширения команд JUMP

Биты	Поле	Описание
16-0	Непосредств. расширение	Задает значение непосредственного расширения
21-17	R	Зарезервировано
29-22	11001110	Определяет, что это непосредственное расширение команды перехода
31	EX	Установленный бит EX определяет команду как последнюю в командной строке. Помните, что если в командную строку включено более одной команды, данная команда должна быть второй

Формат неклассифицированных команд

Формат неклассифицированных команд характеризуется рисунком В-17 и таблицей В-29.

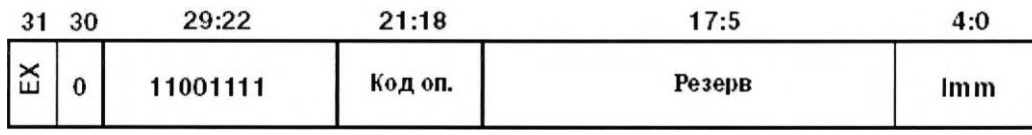


Рисунок В-17 – Формат неклассифицированных команд

Таблица В-29 – Поля кода операции неклассифицированных команд

Биты	Поле	Описание
4-0	IMM	Непосредственное значение для команды TRAP

Биты	Поле	Описание
17-5	Резерв.	Зарезервировано
21-18	Код оп.	Определяет код операции. См. таблицу B-30
29-22	11001111	Определяет, что это команда неклассифицированного типа
30	0	Постоянный 0 – эта группа не может быть условной
31	EX	Установленный бит EX определяет команду как последнюю в командной строке

Коды операции для команд этого типа перечислены в таблице B-30.

Таблица B-30 – Синтаксис и коды операции для неклассифицированных команд

Синтаксис команлы	Тип	Код операции	Комментарии
NOP	11001111	0000	–
IDLE	11001111	0001	–
команда BTB	11001111	0010	Команда BTB, где биты 4-0 таковы: 00000 – BTBINV (объявить недействительным); 00010 – BTBEN (подключить) 00100 - BTBDIS (отключить)
TRAP (spvcmd)	11001111	0100	“spvcmd” есть непосредственное значение, указанное в битах 4:0
EMUTRAP	11001111	1000	–

Код команды NOP используется для задания кодов расширения системы команд – кодов префикса. Допустимые значения кодов приведены в таблице B-31.

Таблица B-31 – Коды префиксов при использовании команды NOP

Название	Код	Код команды	Примечание
PI8	20	0x33C00014	Используется для указания байтового типа адреса при работе со словами данных в памяти
PI8U	21	0x33C00015	Используется для указания байтового типа адреса при работе со словами данных в памяти по невыровненному адресу
PI16	22	0x33C00016	Используется для указания короткого типа адреса при работе со словами данных в памяти
PI16U	23	0x33C00017	Используется для указания короткого типа адреса при работе со словами данных в памяти по невыровненному адресу
to_U8 to_U16	24	0x33C00018	Используется в АЛУ вычислительных модулей
to_U8R to_U16R	25	0x33C00019	Используется в АЛУ вычислительных модулей
to_Sign ZifNeg BIAS	26	0x33C0001A	Используется в АЛУ вычислительных модулей
IF_EQ	28	0x33C0001c	Используются в АЛУ вычислительных модулей
IF_GE	29	0x33C0001d	
IF_GT	30	0x33C0001e	

Приложение Г
Глоссарий

Здесь приводятся определения терминов, важных для понимания архитектуры процессора.

ALU (арифметико-логическое устройство)

Эта часть вычислительного блока выполняет арифметические и логические операции над данными с фиксированной и плавающей точкой.

CLU (коммуникационно-логическое устройство)

Процессор включает в себя специализированное вычислительное устройство, называемое коммуникационно-логическим устройством (CLU). Команды CLU предназначены для поддержки различных алгоритмов, используемых в коммуникационных приложениях.

DMA (прямой доступ к памяти)

Система В/В процессора и сам процессор поддерживают прямой обмен данными между памятью процессора и внешней памятью, хостом или периферийными устройствами через внешний порт или порты линков. При всякой операции DMA передается целый блок данных.

IALU (целочисленные арифметико-логические устройства)

IALU генерируют адреса памяти при пересылках данных между памятью и регистрами.

PC-относительный адрес

Режим исполнения команд call и jump по умолчанию заключается в использовании PC- относительного адреса. Адрес в команде комбинируется с адресом в программном счетчике (PC).

Абсолютный адрес

Возможный режим исполнения команд call и jump - с применением опции (abs) - использование абсолютного адреса. 32-битный адрес в команде позволяет адресовать любую позицию в адресном пространстве.

Адресация относительно регистра

Большинство ссылок на данные использует адресацию относительно регистра, т.е. позволяет обращаться программам к блокам данных, базируясь по регистру.

Адресация с модификацией

Целочисленное ALU (IALU) генерирует адрес, который инкрементируется непосредственным значением или значением регистра.

Адресация с постмодификацией и обновлением

Это вид косвенной адресации – постмодификация с обновлением (используется оператор [+=]). Команды этого вида обеспечивают доступ к памяти по индексированному адресу. Такие команды могут загружать значение в регистр-приемник или сохранять значение регистра-источника. После доступа к памяти индексный регистр изменяется на значение модификатора.

Адресация с предмодификацией без обновления

Это вид косвенной адресации – предмодификация без обновления (используется оператор [+]). Команды этого вида обеспечивают доступ к памяти по адресу значение индексного регистра + модификатор без изменения содержимого индексного регистра. Такие команды могут загружать значение в регистр-приемник или сохранять значение регистра-источника.

Архитектура многопроцессорного кластера

Это архитектура многопроцессорной системы, в которой процессор использует внешний порт для межпроцессорного взаимодействия.

Архитектура многопроцессорной обработки потока данных

Это архитектура многопроцессорной системы, в которой процессор использует порты линков для межпроцессорного взаимодействия.

Архитектура статической суперскалярности

Процессор может исполнять до четырех команд за такт, выбираемых из одного блока памяти со скоростью 128 бит/такт. Способность исполнять несколько команд в единственном такте есть следствие архитектуры статической суперскалярности. Она не является суперскалярной архитектурой в строгом смысле, поскольку то, какие команды исполняются в одном и том же такте, указывается в командной строке программистом или компилятором, т.е. программно, а не аппаратно самим кристаллом. Отсутствует также изменение порядка исполнения команд. Зависимость по регистрам проверяется, однако, аппаратно и, где это необходимо, генерируются такты простоя. Код располагается в памяти вполне компактно и какие-либо ограничения по выравниванию командных строк отсутствуют.

Архитектура фон Неймана

Эта архитектура применяется в большинстве микропроцессоров, отличных от DSP. В этой архитектуре используется единственная шина адреса и единственная шина данных для доступа к памяти.

База (кольцевого буфера)

База (начальный адрес) кольцевого буфера устанавливается в специально зарезервированном для этой цели регистре. Таковыми могут служить регистры JB3-JB0 в J-IALU и KB3-KB0 в K-IALU.

Бит-реверсивная адресация

Целочисленное ALU (IALU) обеспечивает бит-реверсивную адресацию при пересылке данных без того, чтобы использовать реверсию адреса при промежуточном его хранении.

Блок

Блок памяти – это 4-М битное (128-К словное) устройство памяти, включающее в себя автономный блок встроенной SRAM вместе с сопутствующими буферами и схемами управления.

Блоки памяти и банки памяти

Внутренняя память процессора подразделяется на блоки внутренней встроенной SRAM. Пространства внешней памяти подразделяются на банки SRAM и SDRAM, а управление выбором между ними осуществляется соответствующими линиями.

Вычислительные блоки

Ядро процессора содержит два вычислительных устройства, называемых вычислительными блоками. Каждый вычислительный блок содержит регистровый файл и четыре независимых вычислительных подустройства – ALU, CLU, умножитель и сдвиговое устройство.

Гарвардская архитектура

Процессоры, выполняющие операции цифровой обработки сигналов (DSP) требуют большей пропускной способности, чем обеспечивает архитектура фон Неймана, поэтому многие процессоры используют архитектуру памяти, которая имеет отдельные шины адреса и данных для хранения программ и отдельные шины адреса и данных для хранения данных. Два набора шин позволяют процессору осуществлять выборку слова данных и команды одновременно. Такой тип архитектуры памяти называется гарвардской архитектурой.

Двойной регистр

Синтаксические элементы *Rsd, Rmd* или *Rnd* обозначают двойной регистр, содержащий 64-битное слово (или его подмножество битов).

Длина кольцевого буфера

Длина кольцевого буфера (количество ячеек памяти, занятых под буфер) задается специальный регистр. Им может быть JL3-JL0 в J-IALU и KL3-KL0 в K-IALU.

Загрузка (регистра из памяти)

Дополнительно к целочисленным операциям IALU выполняет операции загрузки, сохранения и пересылки данных. Операция загрузки перемещает значение из ячейки памяти в регистр.

Квадрорегистр

Синтаксические элементы *Rsq, Rmq* или *Rnq* обозначают квадрорегистр, содержащий 128- битное слово (или его подмножество битов)

Кольцевые буфера

IALU поддерживает адресацию кольцевого буфера – диапазона адресов памяти – для которого IALU обеспечивает многократный пошаговый проход с возвратом к началу по достижению конца, что имитирует движение по кольцу. В командах чтения/записи памяти режим адресации кольцевого буфера задается оператором CB.

Конвейер вычислительного блока

Последние ступени конвейера (EX1 и EX2) называются конвейером вычислительного блока.

Конвейер устройства выборки

Первые три ступени конвейера команд (IA, IM, IW) называются конвейером устройства выборки.

Конфликт по ресурсам

Конфликт по ресурсам вызывается состязанием внутренних шин при обращении к блоку памяти во время обращения к внутренней памяти.

Косвенная адресация

Косвенная адресация использует индексный регистр с ненулевым значением и в качестве модификатора адреса - либо регистр, либо непосредственное значение.

Многопроцессорная система

Эта система состоит из нескольких процессоров с хост-процесссором или без такового. Процессоры связаны внешней шиной и/или портами линков.

Модификатор кольцевого буфера

Модификатор кольцевого буфера (приращение адреса в какой-либо операции с ячейками памяти) может быть указан как регистр общего назначения IALU или как непосредственное значение. Значение модификатора не может превышать длину кольцевого буфера.

Насыщение

Когда насыщение включено и происходит переполнение результата, возвращаемый результат представляет собой крайнее значение, представляемое форматом, в котором выполняется операция, в направлении правильного результата.

Одиночный регистр

Синтаксические элементы *Rs, Rm* или *Rn* обозначают одиночный регистр, содержащий 32- битное слово (или его подмножество битов).

Одна Команда - Одни Данные (SISD)

и Одна Команда - Множественные Данные (SIMD)

Префиксы вычислительного блока позволяет вам выбрать между исполнением команды на одном или обоих вычислительных блоках. Этот префикс обеспечивает выбор между исполнением в режиме Одна Команда - Одни Данные (SISD) и исполнением в режиме Одна Команда - Множественные Данные (SIMD).

Операнды

Исходные и результирующие операнды команд размещаются в одиночных, двойных и квадрорегистрах регистровых файлов *(Rs, Rsd, Rsq).*

Переполнения

Переполнение означает, что значение результата превышает максимум, представимый в результирующем формате.

Пересылка (значения из регистра в регистр)

Дополнительно к целочисленным операциям IALU выполняет операции загрузки, сохранения и пересылки данных. Операция пересылки перемещает значение из одного регистра в другой регистр.

Переходы

Переходы безвозвратно переключают поток исполнения на другой участок программной памяти.

Периферийные устройства

Периферийными устройствами называют все то, что находится вне ядра процессора. Периферийные устройства процессора включают в себя внутреннюю память, внешний порт, процессор ввода/вывода, порт JTAG и все внешние устройства, связанные с процессором.

Подпрограммы

Команды, объединенные в подпрограмму, выполняют специальную задачу в отдельной области памяти. Они исполняются, когда процессор временно прерывает последовательный поток исполнения основной программы. По завершению задачи поток возвращается в соответствующее место основной программы.

Поле команды и командная строка

Есть несколько важных моментов, которые следует отметить относительно структуры командного поля командной строки, а также того, как эта структура связана с исполнением команд. Каждая командная строка объединяет до четырех 32-битных командных полей. Командные поля ограничены одной точкой с запятой. Командные строки завершаются двумя точками с запятой. Четыре команды из командной строки могут исполняться параллельно.

Порт JTAG

Этот порт поддерживает стандарт IEEE 1149.1 Объединенной рабочей группы по автоматизации тестирования (Joint Test Action Group - JTAG). Стандарт определяет метод последовательного сканирования статуса В/В каждого компонента в системе.

Прерывания

Подпрограммы, выполнение которых провоцируется событиями (не командами), происходящими во время исполнения.

Прерывания, замаскированные и размаскированные

В регистре imask значение бита, равное “1”, означает, что прерывание размаскировано (процессор распознает и обслуживает прерывание). Значение “0” соответствует замаскированному прерыванию (процессор не распознает прерывание).

Приостановки (задержки исполнения команд)

Приостановка - это любая задержка, вызванная ситуацией зависимости или конфликтом по регистрам.

Прогнозируемые и непрогнозируемые переходы

Предположением по умолчанию, применимым ко всем условным командам перехода, является то, что значение предиката есть `true` и что переход должен быть предпринят - т.н. прогнозируемый переход. Когда по каким-то своим соображениям профаммист может предполагать, что в большинстве случаев переход не предпринимается, он должен снабдить такой переход опцией «непрогнозируемый» (np). Такой переход называется непрогнозируемым.

Прямая адресация

Прямая адресация использует указатель адреса (индексный регистр) с нулевым значением и непосредственное значение в качестве модификатора. При этом указатель адреса предмодифицируется, и полученное значение используется в качестве адреса при обращении к памяти.

Регистр-модификатор

Регистр-модификатор целочисленного ALU (IALU) задает величину приращения или шага, на который индексный регистр пред- или постмодифицируется при пересылке регистра[1].

Регистры данных (Dreg)

Ureg-регистры вычислительного блока могут быть использованы для дополнительных операций, которые недоступны для других *Ureg*-регистров. Чтобы отличать регистры XR31-0 и YR31-0 регистрового файла вычислительного блока от других *Ureg*-регистров, их именуют еще регистрами данных (Dreg).

Ресурсы

Ресурсы - это части процессорной архитектуры, которые активны во время выполнения команды

Сдвиговое устройство

Эта часть вычислительного блока выполняет над 32-битными операндами операции логических сдвигов, арифметических сдвигов, манипуляций с битами, заполнения полей и извлечения полей. Кроме того, сдвиговое устройство может извлекать порядок [из числа в формате плавающей точки].

Ситуация зависимости

Ситуация зависимости вызывается командой, которая использует в качестве входа результат предыдущей команды, если данные предыдущей команды не готовы к тому моменту, когда текущей команде требуется операнд.

Сохранение (регистра в памяти)

Дополнительно к целочисленным операциям IALU выполняет операции загрузки, сохранения и пересылки данных. Операция сохранения перемещает значение из регистра в ячейку памяти.

Супергарвардская архитектура

В процессорах сделан следующий шаг от гарвардской архитектуры - в них использована т.н. супергарвардская архитектура. В этой архитектуре имеются шины программ и данных, но поддерживается единое адресное пространство для хранения программ и данных. По шине памяти данных передаются только данные, а шина программной памяти обслуживает как команды, так и данные.

Типы команд

Типы команд - это описательные названия классов команд

Точность

Точность числа с плавающей точкой зависит от числа битов после двоичной точки в формате хранения числа. Процессор поддерживает два формата данных с плавающей точкой высокой точности: 32-битный формат данных с плавающей точкой одинарной точности стандарта IEEE (который использует 8 бит для порядка и 24 бита для мантиссы), а также 40-битную версию формата повышенной точности стандарта IEEE.

Транзакция системы памяти

Транзакция системы памяти – это обращение по чтению или записи к блоку памяти.

Указатель (для кольцевого буфера)

Указатель кольцевого буфера (текущий адрес) устанавливается регистром общего назначения. Это регистры J3-J0 в J-IALU и K3-K0 в K-IALU.

Умножитель

Эта часть вычислительного блока выполняет умножение с плавающей и фиксированной точки, а также операции умножения/сложения или умножения/вычитания с фиксированной точкой.

[1] Или при формировании адреса.

Универсальные регистры (Ureg)

Отображённые в память регистры являются универсальными (*Ureg).* Они называются универсальными, потому что f/reg-регистры могут быть использованы для многих видов операций и в том числе для операций вне той части ядра процессора, где *Ureg*-регистры располагаются.

Упакованные данные

Данные в 8- и 16-битных форматах всегда упаковываются в 32-битных регистрах следующим образом - одиночный регистр вмещает четыре 8-битных или два 16-битных слова, двойной регистр вмещает восемь 8-битных или четыре 16-битных слова, а квадрорегистр шестнадцать 8-битных или восемь 16-битных слов.

Условные переходы

Это команды Jump или Call (с возвратом), которые исполняются или не исполняются в зависимости от результата проверки некоторого If-условия (значения предиката).

Файл регистров данных

Это набор регистров, через которые производится обмен данными между шинами данных и вычислительными устройствами. Эти регистры также обеспечивают локальное хранение операндов и результатов.

Цикл

Цикл есть некоторая последовательность команд, которая исполняется несколько раз с нулевыми накладными расходами.

Лист регистрации изменений

№ п/п	Дата	Версия	Краткое содержание изменения	№№ изменяемых листов
1	20.11.2018	1.0.0	Введен вновь	
2	26.04.2019	2.0.0	Форматирование и добавление обновленного синтаксиса некоторых команд	12, 67, 68, 115, 174, 175, 176, 285, 290, 387, 391, 402, 403, 425
3	17.05.2019	2.1.0	Форматирование, исправление опечаток	По тексту
4	21.06.2019	2.2.0	Форматирование, исправление опечаток	По тексту
5	14.08.2019	2.3.0	Форматирование, исправление опечаток	По тексту
6	31.07.2020	2.4.0	Исправление ошибок и опечаток	44, 45, 60, 61, 63, 70, 126-129, 136, 143-148, 158, 162, 258, 269, 316, 317, 435, 436