Руководство системного программиста

Микропроцессор Leonhard x64

А.Ю.Попов

LEON.001.031917.03.2019

http://e-learning.bmstu.ru/Leonhard

Leonhard Euler microprocessor

Bauman Moscow State Technical University

Леонард Эйлер

LEON.001.0319
Руководство системного программиста. Микропроцессор Leonhard x64/А.Ю.Попов/ версия: 0.1. дата: 17.03.2019

Published by BMSTU

bmstu.ru

Опубликовано: Март 2019

Содержание

1. Микропроцессор Leonhard x64

1.1 Архитектура вычислительной системы на основе Leonhard x64

Структуры данных с формальной точки зрения представляет собой совокупность двух сущностей: информационную составляющую о значениях полей данных (i) и структурную составляющую, учитывающую отношения данных (ii). Такая двойственность позволяет разделить процесс обработки структур данных на два потока вычислений: во-первых, поток обработки структурной составляющей и, во-вторых, поток обработки информационной части. В связи с этим в вычислительной системе может быть два параллельно работающих микропроцессора: это специальный микропроцессор, который обрабатывает реляционную (структурную) часть структур данных, в то время как универсальный CPU выполняет вычисления над информационной составляющей структур.

У микропроцессора Leonhard x64 имеется собственная независимая шина для доступа к локальной памяти, в которой происходит хранение структур данных (Рисунок 1.1). Результаты выполнения инструкций в Leonhard x64 посылаются в CPU для дальнейшего их использования в вычислительном алгоритме.

В представленной системе существует два варианта запуска команды в SPU: команда запускается CPU, который отправляет ее через очередь команд C2S (i); SPU запускает команду, загруженную им из локальной памяти команд (ii). Если применяется первый вариант, то такая система является гетерогенной вычислительной системой (т.е. содержит неоднородные процессорные устройства). Если используется второй вариант, то такую систему можно классифицировать как параллельный компьютер с многими потоками команд и одним потоком данных.

Leonhard x64 использует два типа синхронизации выполняемых инструкций с потоком команд CPU. Первый способ использует семафоры для перевода блока выборки команд в состояние ожидания операндов. Для этой цели формат инструкций, которые поступают из локальной памяти команд (MISD Command Queue) включает в себя специальный бит тега для каждого из операндов (причем в одной команде может быть до трех операндов). Во втором режиме CPU отправляет команды в SPU через очередь команд C2S только тогда, когда они действительно должны быть запущены и со всеми

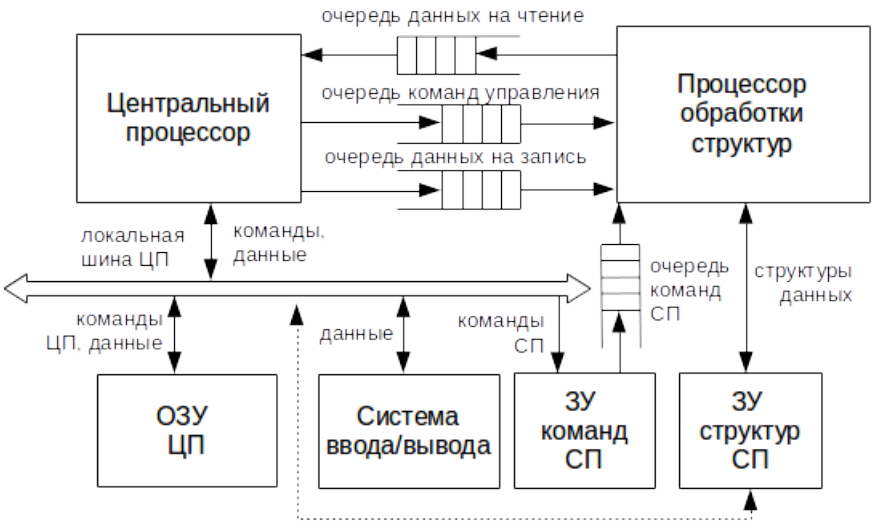

Рисунок 1.1: Архитектура вычислительной системы с несколькими потоками команд и одним потоком данных

необходимыми операндами (аналогично взаимодействию математического сопроцессора с процессорам Intel 386). Таким образом, Leonhard SPU может работать в режиме MISD (первый вариант), в режиме ускорителя (второй вариант) и в комбинированном режиме (сочетаются оба способа синхронизации).

1.2 Микроархитектура Leonhard x64

Микропроцессор Leonhard x64 представляет собой устройство для обработки структур данных (Процессор обработки структур, Structure processing unit, SPU), управляемое специальным набором команд, и предназначено для хранения и обработки больших множеств дискретной информации. Под управлением поступающих команд Leonhard x64 выполняет хранение ключей и значений в многоуровневой подсистеме памяти, выполняет поиск, изменение и выдачу информации другим устройствам системы [1].

Для ускорения поиска и обработки всего набора команд микропроцессор Leonhard x64 использует внутренне представление множеств в виде B+дерева, для которого возможна параллельная обработка нескольких вершин дерева как на промежуточных уровнях, используемых для поиска, так и на нижнем уровне, хранящем непосредственно ключи и значения. В связи с этим любая операция над структурой начинается с поиска информации в B+дереве, а заканчивается обработкой так называемых листьев дерева (вершин нижнего уровня). Микроархитектура микропроцессора Leonhard x64 показана на Рисунке 1.2.

R Последовательность вершин B+дерева, составляющая путь для аппаратного поиска ключа от корня до листа, называется **трассой**. Трасса сохраняется во внутренней памяти процессора обработки структур для возможности оперативного доступа.

После формирования полной трассы в *Каталоге* становится известным информация вершины нижнего уровня (листа), где хранятся ключи и значения. Далее следует загрузить вершину нижнего уровня из внутренней *Памяти структур* в *Операционный буфер*. Для этого требуется транслировать номер вершин в двоичный код выборки,

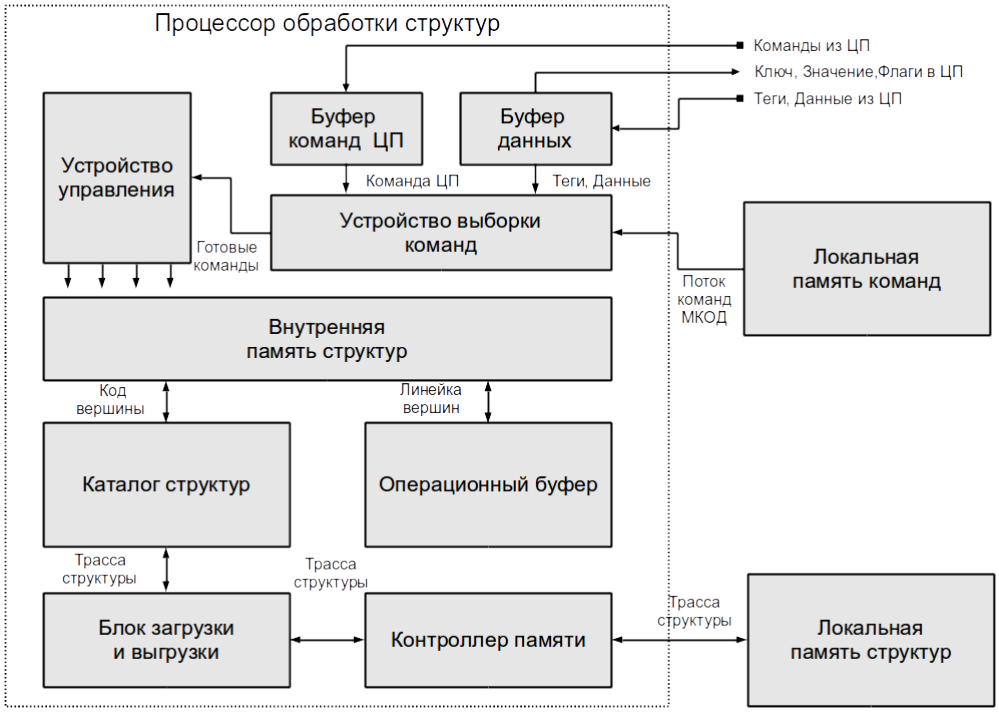

Рисунок 1.2: Подсистемы процессора обработки структур

который служит для обращения во внутреннюю *Память структур*. В качестве кода может использоваться как простой линейный адрес ОЗУ, так и код выборки строки из специального массива. Если вершина отсутствует в *Памяти структур*, это приводит к вычислению ее адреса, загрузке вершины из внешней памяти (*Локальной памяти структур*).

R Вершина нижнего уровня B+дерева, состоящая из некоторого количества ключей и значений, хранится в специальной памяти внутри *Процессора обработки структур* (*Памяти структур*) и называется **линейкой**. Информация одной линейки может обрабатываться в СП параллельно.

Операционный буфер принимает блок данных, составляющий вершину нижнего уровня и сохраняет ее во внутренних процессорных элементах, обеспечивающих выполнение однотипных команд над каждым ключом. Требуется выполнять вставку новых ключей в соответствии с их порядком, т.е. поддерживая упорядоченной при хранении. В следствии такой вставки может произойти переполнение линейки, что требует добавления вытесненного значения в следующую линейку или изменение структуры дерева таким образом, чтобы за переполненной линейкой была добавлена новая, после чего в нее будет перенесена часть ключей. Другой важной функцией *Операционного буфера* является выполнение операций над несколькими линейками для реализации И-ИЛИ-НЕ операций и команд срезов. Таким образом, *Операционный буфер* выполняет изменение линеек и передает их обратно в *Память структур*, а также передает информацию о результатах обработки *Устройству управления*. Модифицированная трасса, в случае создания новых линеек, при необходимости полной или частичной загрузки новой трассы, должна быть сохранена во внешней памяти структур.

СП состоит из следующих взаимосвязанных блоков:

- **Каталог** хранит информацию о расположении элементов структуры в *Памяти структур*. При выполнении операций, связанных с модификацией структуры, содержимое *Каталога* также модифицируется. *Каталог* позволяет хранить информацию одновременно для нескольких структур и выполнять такие сложные операции, как пересечение, объединение или дополнение структур. *Каталог* является сложным устройством, представляющим из себя вычислительную структуру типа «один поток команд и много потоков данных».
- **Операционный буфер** служит для обработки линеек, выбираемых из *Памяти структур* в соответствии с текущей командой. *Операционный буфер* также реализует принцип обработки «один поток команд и много потоков данных».
- **Память структур** является основным внутренним хранилищем структурной информации и ассоциированных с ней данных. *Память структур* разделена на линейки, состоящие из некоторого количества пар «ключ-значение». Могут быть получены варианты процессора с различными размерами линейки, что будет влиять как на аппаратную сложность, так и на производительность процессора. Больший объем *Памяти структур* приведет к более быстрому доступу к линейке, однако это требует и хранения большего количества трасс в *Каталоге*.
- **Блок загрузки и выгрузки**. Данный блок используется в том случае, если необходимо хранить структуры, превосходящие по размерам *Память структур*. В таком случае выполняется загрузка линеек из внешней памяти в процессор, а также их выгрузка. Блок загрузки и выгрузки генерирует адреса чтения и записи внешней памяти и содержит FIFO буферы данных, позволяющие накапливать большое количество транзакций.
- **Контроллер памяти**: конвейерный контроллер внешнего запоминающего устройства большой емкости. Целесообразно использовать такие типы памяти, как: *SDR SDRAM, DDR SDRAM, DDR2 SDRAM, DDR3 SDRAM*, соответствующие стандарту организации *JESD*. Контроллер должен поддерживать различные варианты разделения адреса, ускоренный доступ к многобанковой памяти, контроль ошибок памяти, а также буферизированный и не буферизированный ECC доступ.
- **Буфер данных** получает информацию от процессора и передает результаты. От ЦП передаются ключи, значения, номера структур, а также информацию о синхронизации ветвей алгоритма. Из СП через *Буфер данных* передаются результаты выполненных команд: ключи и значения искомых структур, а также флаги результатов.
- **Буфер команд ЦП** используется в режиме сопроцессора. Через этот буфер СП получает от *Центрального процессора* команды управления, которые совместно с данными в *Буфере данных* составляют команду.
- **Устройство управления** обеспечивает параллельное управление всеми перечисленными выше устройствами в соответствии с принципами конвейерной обработки. Параллельно могут выполняться: обработка линеек структур, изменение содержимого Каталога, передача данных из *Памяти структур* во *Внешнюю память структур* и обратно.
- **Блок выборки команд** обеспечивает загрузку команд потока обработки структур из локальной памяти, принимает сигналы синхронизации потоков от ЦП, передаваемых в виде специальных сообщений (подробнее данный вопрос будет рассмотрен ниже). Блок выборки команд также принимает команды управления от ЦП.

Машинные команды процессора обработки структур являются командами высокого

уровня. В рассматриваемой версии реализованы 20 команд: команды поиска, добавления, удаления, минимума/максимума, мощность, И-ИЛИ-НЕ команды, срезы, обход структуры, сжатие структур и команда перехода. Обработка команд начинается со стадии загрузки из ОЗУ в *Блок выборки команд* кодов операции, операндов и тегов.

1.3 Набор команд дискретном математики

Микропроцессор Leonhard x64 хранит информацию о множествах в виде неперекрывающихся B+ деревьев. Последняя версия набора команд Leonhard x64 была расширена двумя новыми инструкциями (NSM и NGR) для обеспечения требований некоторых алгоритмов. Каждая инструкция набора включает до трех операндов (таблица 1.1):

Таблица 1.1: Формат данных Leonhard x64

Структура	Ключ	Значение
3 бита	64 бита	64 бита

Набор команд состоит из 20 высокоуровневых кодов операций, перечисленных ниже.

- **Search (SRCH)** выполняет поиск значения, связанного с ключом.
- **Insert (INS)** вставляет пару ключ-значение в структуру. SPU обновляет значение, если указанный ключ уже находится в структуре.
- **Операция Delete (DEL)** выполняет поиск указанного ключа и удаляет его из структуры данных.
- **Neighbors (NSM, NGR)** выполняют поиск соседнего ключа, который меньше (или больше) заданного и возвращает его значение. Операции могут быть использованы для эвристических вычислений, где интерполяция данных используется вместо точных вычислений (например, кластеризация или агрегация).
- **Maximum /minimum (MAX, MIN)** ищут первый или последний ключи в структуре данных.
- **Операция Cardinality (CNT)** определяет количество ключей, хранящихся в структуре.
- **Команды AND, OR, NOT** выполняют объединения, пересечения и дополнения в двух структурах данных.
- **Срезы (LS, GR, LSEQ, GREQ)** извлекают подмножество одной структуры данных в другую.
- **Переход к следующему или предыдущему (NEXT, PREV)** находят соседний (следующий или предыдущий) ключ в структуре данных относительно переданного ключа. В связи с тем, что исходный ключ должен обязательно присутствовать в структуре данных, операции NEXT/PREV отличаются от NSM/NGR.
- **Удаление структуры (DELS)** очищает все ресурсы, используемые заданной структурой.
- **Команда Squeeze (SQ)** дефрагментирует блоки памяти DSM, используемые структурой.
- **Команда Jump (JT)** указывает SPU код ветвления, который должен быть
- синхронизирован с CPU (команда доступна только в режиме MISD).

В таблице Основные параметры реализации опытного образца приведены в Таблице 1.2 приведены основные параметры Leonhard x64.

Микропроцессор реализован на базе микросхемы ПЛИС XC6VLX240T-1FFG1156, входящей в состав отладочной платы ML605 (рисунок 1.3).

Таблица 1.2: Параметры Leonhard x64

Параметр	**Пояснение**	**Значение**
CMD_MISD_VAL	Максимальный размер команды в локальной памяти команд СП (LCM):	144 бит
CMD_CPU_VAL	Максимальный размер команды из ЦП:	160 бит
REZ_VAL	Максимальный размер результата из СП в ЦП:	64 бит
KEY_VAL	Количество разрядов поля ключа	64 бита
DATA_VAL	Количество разрядов поля значения	64 бита
MSB	Расположение байт в памяти	Младший байт по младшему адресу
EXT_MEM	Размер внешней памяти структур	4 ГБайта
STR_VAL	Максимальное количество ключей в структуре	100.663.296
VIRT_NUM	Кратность вершины B+ дерева	8
LEAF_NUM	Количество ключей на нижнем уровне дерева	6
STR_NUM	Максимальное количество хранимых структур	7

Рисунок 1.3: Реализация микропроцессора Loenhard x64

2. Программная модель микропроцессора

R Команды могут поступать в Leonhard x64 как от ЦП (режим сопроцессора), так и из *Локальной памяти команд* (режим MISD). В связи с этим СП исполняет два набора команд: команды ЦП и команды MISD. Отличие наборов команд следующее: команды ЦП содержат поле кода операции, а также поля операндов; команды MISD содержат код операции, поля операндов, поле тега для каждого из операндов.

2.1 Форматы команд в режиме MISD

Формирование команд ЦП выполняется поэтапно передачей информационных слов по *Системной шине*. Операнды (ключи, значения, номера структур) загружаются в *Буфер данных* последовательно, после чего в *Буфер команд ЦП* загружается код команды и она принимается в *Блок упорядочивания потоков команд* (**??**). Таким образом, формат команд ЦП определяется разрядностью полей и адресами регистров на системной шине, что реализуется специальными API функциями. Для унификации мнемокодов команд обоих форматов примем, что формат вызова команд из кода ЦП совпадает с форматом команд МКОД.

Следует также отметить, что даже в случае возникновения ошибок при выполнении команд (обычно, это связано с нехваткой места для размещения новых ключей и значений), структура находится в непротиворечивом состоянии, т.е. никакие данные не уничтожаются безвозвратно. Подробнее вопрос реакции СП на возникновение ошибок будет рассмотрен применительно к каждой команде.

В данном разделе представим мнемокоды команд ассемблера СП и их коды, а также опишем основные правила языка ассемблера. Команды состоят из следующих полей: кода операции (КОП); от одного до 3-х полей операндов и соответствующих им тегов (см. Таблицу 2.21). Напомним, что тег указывает на достоверность операндов в памяти команд СП. Если тег равен 0, то операнд не является достоверным и СП ожидает его поступления. Эта ситуация обозначается в мнемокоде команды знаком «?». Например:

min ?; - указан код операции min, но номер структуры является неизвестным и ожидается его получение из ЦП.

add ?,1,?; указан код операции add, номер структуры не указан, ключ равен 1, данные не указаны.

Результат выполнения команды может сохраняться в очереди результатов. В другом случае запись в эту очередь может быть запрещена. За буферизацию результатов отвечает бит Q, который явно указывается в названии команды:

addq 1,2,3; добавление в структуру 1 ключа 1 и значения 2 с буферизацией;

add 1,2,3; добавление без буферизации.

Для команд ЦП операнд всегда должен быть указан явно в команде. Расположение операндов в мнемокоде команде слева направо: **коп оп0,оп1,оп3**. Расположение операндов в памяти: младший по младшему адресу.

Таким образом, если бит Q = 1 то буферизация результатов выполняется, если Q = 0 то буферизация результата не выполняется.

Комментарии начинаются с символа «;» и продолжается до конца строки. В командах перехода используются адреса команд, которые могут указываться непосредственно в виде адреса или с помощью символьных меток.

Search(Str,Key)

Команда выполняет поиск ключа в заданной структуре. Номер структуры задается в диапазоне (1..STR_NUM), что для опытного образца составляет 3 двоичных разряда. Поле ключа составляет KEY_VAL разрядов (см. Таблицу 2.1).

Примеры:

search ?,? - оба тега (структура и ключ) не валидны, буферизация результата не выполняется.

search 1,? - ключ поиска не валиден, буферизация результата не выполняется.

searchq ?,? - оба тега (структура и ключ) не валидны, буферизация результата выполняется.

searchq 1,? - ключ поиска не валиден, буферизация результата не выполняется.

Таблица 2.1: Формат команды поиска (search)

КОП	ТЕГ#2	ТЕГ#1	ТЕГ#0	Q	СТРУКТУРА		КЛЮЧ
5 бит #5	N 1 бит	1 бит	1 бит	1 бит	N 4 бит*	3 бит	3 бит

* здесь и далее в разделе **??**: N указывает количество не используемых разрядов.

Результат выполнения команды составляет код результата (8 бит), в котором указан код завершения команды, найденный ключ (совпадает с ключом поиска), а также найденное значение (Таблица 2.2).

Таблица 2.2: Результат команды поиска (search)

КОД РЕЗУЛЬТАТА				КЛЮЧ	ЗНАЧЕНИЕ
КОД РЕЗУЛЬТАТА	V	N	R	64 бит	64 бит

В случае выполнения команды с установленным флагом Q (Q = 1), результат выполнения команды сохраняется в выходной очереди данных СП и может быть прочитан кодом ЦП для последующего анализа. Разряд R указывает на готовность кода результата. В то время когда результат команды не готов (команда не закончила свою

работу), разряд R равен 0. Если команда закончила работу, разряд R = 1 (аналогично для всех команд СП).

Разряд V указывает на положительный результат поиска: если V = 0 то ключ не обнаружен и поле КОД РЕЗУЛЬТАТА выдается специальный код, уточняющий возникшую ошибку (коды представлены в документации); при V = 1 ключ найден и КОД РЕЗУЛЬТАТА не содержит значимой информации (аналогичная логика для всех команд).

Insert(Str,Key,Data)

Команда добавления **Insert** выполняет как вставку нового значения, так и изменения поля данных существующего значения. Команда принимает на вход номер структуры, поле ключа, а также поле значения (Таблица 2.3).

Примеры команды:

insertq ?,?,? - операция добавления, номер структуры, ключ и значение не валидны и должны поступить из ЦП.

insertq 1,2,? - операция добавления, номер структуры = 1, ключ = 2, поле значения не валидно.

insert 1,2,3 - команда содержит все поля: номер структуры = 1; ключ = 2; значение = 3. Результат в выходной очереди не формируется.

Таблица 2.3: Формат команды добавления (insert)

КОП	ТЕГ#2	ТЕГ#1	ТЕГ#0	Q	СТРУКТУРА		КЛЮЧ
5 бит #2	1 бит	1 бит	1 бит	1 бит	N 4 бит	3 бит	64 бит

Если операция проведена успешно, то в поле V = 1. В противном случае указывается бит V = 0. Возможны следующие типы ошибок:

- Добавление выполнено успешно, но вытеснена последняя запись структуры. Причиной этого является отсутствие свободного места в *Локальной памяти структур*. В таком случае вытесненное значение передается в полях КЛЮЧ и ЗНАЧЕНИЕ (Таблица 2.4).
- Добавление не выполнено, так как невозможно выделить место для вставки нового значения. В этом случае поля КЛЮЧ и ЗНАЧЕНИЕ не содержат валидной информации.

Таблица 2.4: Результат команды добавления (insert)

КОД РЕЗУЛЬТАТА				КЛЮЧ	ЗНАЧЕНИЕ
КОД РЕЗУЛЬТАТА	V	N	R	64 бит	64 бит

В случае успешного добавления поле КОД РЕЗУЛЬТАТА содержит информацию, было ли выполнена вставка нового ключа и значения, или произошло изменение значения существующего ключа.

Del(Str,Key)

Операция удаления **Del** получает номер структуры и ключ (Таблица 2.5).

Пример операции:

delq(1,2) - операция удаления из структуры 1 по значению ключа = 2. Результат поместит в очередь данных.

Таблица 2.5: Формат команды удаления (del)

КОП	ТЕГ#2	ТЕГ#1	ТЕГ#0	Q	СТРУКТУРА		КЛЮЧ
5 бит #1	N 1 бит	1 бит	1 бит	1 бит	N 4 бит	3 бит	64 бит

Положительный результат операции (бит V поля результата) означает, что искомый ключ найден и удалении информации проведено успешно (Таблица 2.6).

Таблица 2.6: Результат команды удаления (del)

КОД РЕЗУЛЬТАТА				КЛЮЧ	ЗНАЧЕНИЕ
КОД РЕЗУЛЬТАТА	V	N	R	64 бит	64 бит

Ошибка при V = 0 возникает в следующих случаях:

- Указанная структура не найдена
- Структура существует, но указанного ключа в ней нет (Таблица 2.6).

DelStr(Str)

Операция удаления структуры получает номер удаляемой структуры Str. В результате из *Локальной памяти структур* удаляются все ключи и значения, хранимые в указанной структуре (см. Таблицу 2.7).

Пример команды:

del 6 - удалить структур «6».

delq ? - удалить структур, номер которой будет передан из ЦП. Результат поместить в очередь данных.

Таблица 2.7: Формат команды удаления структуры (delstr)

КОП	ТЕГ#2	ТЕГ#1	ТЕГ#0	Q	СТРУКТУРА	
5 бит #F	1 бит	N 1 бит	N 1 бит	1 бит	N 4 бит	3 бит

Результат выполнения команды возможны два результата: если такая структура существует, то при удалении устанавливается флаг V = 1, в противном случае V = 0 (см. Таблицу 2.8).

Таблица 2.8: Результат команды удаления структуры (delstr)

КОД РЕЗУЛЬТАТА			
КОД РЕЗУЛЬТАТА	V	N	R

Max(Str) и Min(Str)

Команды поиска максимума и минимума имеют один операнд: номера структуры Str (см. Таблицу 2.9). Пример вызова команды:

minq ? - команда поиска минимального значения в структуре. Номер структуры передается из ЦП. Результат помещается в очередь данных.

maxq 3 - команда поиска максимального значения в структуре 3. Результат помещается в очередь данных.

Таблица 2.9: Формат команды поиска минимума или максимума (min, max)

КОП	ТЕГ#2	ТЕГ#1	ТЕГ#0	Q	СТРУКТУРА	
5 бит #4	N 1 бит	N 1 бит	1 бит	1 бит	N 4 бит	3 бит

Код операции команды min - #3; код команды max - #4. Результат выполнения команды составляет код результата (8 бит), в котором указан код завершения команды, найденный ключ (совпадает с ключом поиска), а также найденное значение (таблица 2.10).

Таблица 2.10: Результат команды поиска минимума и максимума (min, max)

КОД РЕЗУЛЬТАТА				КЛЮЧ	ЗНАЧЕНИЕ
КОД РЕЗУЛЬТАТА	V	N	R	64 бит	64 бит

В случае отсутствия указанной структуры в разряде V указывается ноль (V = 0), а поле КЛЮЧ и ЗНАЧЕНИЕ не являются валидными. В противном случае, если структура такая имеется, бит V=1 и в полях КЛЮЧ,ЗНАЧЕНИЕ выдается найденный минимальный(максимальный) ключ и его значение.

Next(Str,Key), Prev(Str,Key), NSM(Str,Key), NGR(Str,Key)

Команды перехода к следующему/предыдущему и ближайшему меньшему/большему значению по порядку ключей используют два операнда: номер структуры и предшествующий ключ. Сначала выполняется поиск места для хранения указанного ключа, после чего выполняется переход к следующему/предыдущему и найденный ключ и значение передаются в качестве результата (см. Таблицу 2.11). Код операции команды Next - #10; код команды Prev - #11, команды NSM - #12; код команды NGR - #13.

Таблица 2.11: Формат команды поиска следующего (next)

КОП	ТЕГ#2	ТЕГ#1	ТЕГ#0	Q	СТРУКТУРА		КЛЮЧ
5 бит #10	N 1 бит	1 бит	1 бит	1 бит	N 4 бит	3 бит	64 бит

Возможны несколько вариантов неудачного поиска. Во первых, указанная структура может отсутствовать, что делает невозможным поиск в ней ключа. Во вторых, указанный базовый ключ может отсутствовать в структуре (является ошибкой только для команд Next и Prev). Также может оказаться, что базовый ключ является последним и следующий за ним ключ просто отсутствует. Результат выполнения команды формируется следующим образом (Таблица 2.12):

Таблица 2.12: Результат команды поиска следующего

КОД РЕЗУЛЬТАТА				КЛЮЧ	ЗНАЧЕНИЕ
КОД ОШИБКИ	V	N	R	64 бит	64 бит

Power(Str)

Команда power позволяет определить количество элементов в заданной структуре. Формат команды указан в Таблице 2.13.

Таблица 2.13: Формат команды поиска следующего (power)

КОП	ТЕГ#2	ТЕГ#1	ТЕГ#0	Q	СТРУКТУРА	
5 бит #7	N 1 бит	N 1 бит	1 бит	1 бит	N 4 бит	3 бит

Результат выполнения команды передается в поле ЗНАЧЕНИЕ (Таблица 2.14). При этом, если структура существует, то значение разряда V = 1. В противном случае поле ЗНАЧЕНИЕ и разряд V равны 0.

Таблица 2.14: Результат команды определения мощности структуры

КОД РЕЗУЛЬТАТА				ЗНАЧЕНИЕ
КОД ОШИБКИ	V	N	R	64 бит

AND(A,B,R), OR(A,B,R), NOT(A,B,R)

Операции объединения, пересечения и разности выполняется над двумя исходными структурами (A и B), результат помещается в новую структур R (Таблица 2.15).

Таблица 2.15: Формат команд И-ИЛИ-НЕ (AND,OR,NOT)

КОП	ТЕГ#2	ТЕГ#1	ТЕГ#0	Q	СТРУКТУРА R		СТРУКТУРА A		СТРУКТУРА B	
5 бит #7	1 бит	1 бит	1 бит	1 бит	N 4 бит	3 бит	N 5 бит	3 бит	N 5 бит	3 бит

Код операции объединения структур AND - #9, код операции пересечения OR - #8, код операции разности - #A.

Ошибка при выполнении команд может означать следующее:

- Структура результата R уже существует, что делает невозможным запись новой структуры результата.
- Исходные структуры A или B отсутствуют в *Локальной памяти структур*.
- Для формирования структуры R недостаточно места в *Локальной памяти структур*.

В случае ошибки ее код передается в поле КОД ОШИБКИ, а наличие ошибки определяется по разряду V (V=0) (Таблица 2.16).

Таблица 2.16: Результат работы команд И-ИЛИ-НЕ

КОД РЕЗУЛЬТАТА			
КОД ОШИБКИ	V	N	R

LS(Key,A,R), LSEQ(Key,A,R), GR(Key,A,R), GREQ(Key,A,R)

Команды срезов получают в качестве операндов ключ среза Key, номер исходной структуры среза A, а также номер структуры результата R (см. Таблицу 2.17).

Пример вызова команд:

lseqq(1,2,3) - выполнить срез структуры 2, выбрав все элементы меньше или равно 1 и поместить в структуру 3.

Таблица 2.17: Формат команд срезов (LS, LSEQ, GR, GREQ)

КОП	ТЕГ#2	ТЕГ#1	ТЕГ#0	Q	СТРУКТУРА R		СТРУКТУРА A		КЛЮЧ
5 бит #7	1 бит	1 бит	1 бит	1 бит	N 4 бит	3 бит	N 5 бит	3 бит	64 бит

Код операции среза «меньше» ls - #C, среза «меньше или равно» lseq - #B, среза «больше» gr - #E, среза «больше или равно» greq - #d. Причины возникновения ошибок могут быть следующие:

- Структура результата R уже существует, что делает невозможным запись новой структуры результата.
- Исходная структура A отсутствует в *Локальной памяти структур*.
- Для формирования структуры R недостаточно места в *Локальной памяти структур*.

В случае ошибки ее код передается в поле КОД ОШИБКИ, а наличие ошибки определяется по разряду V (см. Таблицу 2.18).

Таблица 2.18: Результат работы команд срезов

КОД РЕЗУЛЬТАТА			
КОД ОШИБКИ	V	N	R

SQUEEZE Sq(Str)

Команда сжатия используется для получения компактного представления структуры в *Локальной памяти структур*. Команда получает в качестве операнда номер структуры Str (см. Таблицу 2.19).

Пример вызова команд:

squeeze(3) - выполнить срез структуры 2, выбрав все элементы меньше или равно 1 и поместить в структуру 3.

Таблица 2.19: Формат команд сжатия (SQUEEZE)

КОП	ТЕГ#2	ТЕГ#1	ТЕГ#0	Q	СТРУКТУРА	
5 бит #6	N 1 бит	N 1 бит	1 бит	1 бит	N 4 бит	3 бит

Код операции команды min - #3; код команды max - #4. Результат выполнения команды составляет код результата (8 бит), в котором указан код завершения команды, найденный ключ (совпадает с ключом поиска), а также найденное значение (Таблица 2.20).

Таблица 2.20: Результат команды поиска минимума и максимума (min, max)

КОД РЕЗУЛЬТАТА			
КОД РЕЗУЛЬТАТА	V	N	R

В случае отсутствия указанной структуры в разряде V указывается ноль (V = 0), а поле КЛЮЧ и ЗНАЧЕНИЕ не являются валидными. В противном случае, если структура такая имеется, бит V=1.

JT(тег,адрес)

Команда используется только в режиме МКОД, а при поступлении от ЦП в режиме сопроцессора отбрасывается. При этом в ассемблере СП представляет тег (условие перехода) и адрес перехода. Если тег равен нулю, переход по адресу не происходит, а код начинает выполняться со следующей команды за командой ветвления. Если тег равен единице, то происходит безусловный переход по адресу, указанному в команде. Тег также может быть не задан, что обозначается символом «?», что указывает на необходимость ожидания операнда. Возможны несколько вариантов реализации команды:

- **безусловный переход – jt тег,адрес**. Так как команда не ожидает операндов, она выполняется в СП незамедлительно.
- **условный переход по тегу – jt ?,адрес**. Процессор обработки структур ожидает из ЦП только тег направления перехода. По нулевому тегу совершается переход по указанному адресу, а по единичному тегу - переход к следующей команде. ЦП исполняет команду PUT(тег), по которой тег посылается в СП.
- **условный переход по адресу – jt тег,?**. Процессор обработки структур ожидает адрес перехода, и при его поступлении выполняет переход. ЦП исполняет команду PUT(адрес), по которой адрес перехода посылается в СП.

- **условный переход по адресу и тегу – jt ?,?**. Процессор обработки структур ожидает тег и адрес перехода.

Для кодирования команды используется два тега: Тег #0 для кодирования готовности команды, а Тег #1 для указания готовности адреса. Нулевое начальное значение Тега #0 задается в мнемокоде знаком «?» (например: **jt ?,адрес** и эквивалентную ей команду **jt 0,адрес**). При такой команде ЦП ожидает получения от ЦП Тега #0, после чего команда воспринимается готовой к переходу. В зависимости от полученного значения происходит либо переход по указанному в команде адресу (Тега #0 = 1), либо переход на следующую команду (Тега #0 = 0).

Команда **jt ?,?** кодируется с помощью нулевых значений тегов (Тега #0 = 0, Тега #1 = 0), команда **jt тег,адрес** кодируется с помощью единичных значений тегов (Тега #0 = 1, Тега #1 = 1).

Результат команды отсутствует.

2.2 Форматы команд и работа очереди результатов в режиме сопроцессора

Машинные команды процессора обработки структур, передаваемые Центральным процессором, также имеют аналогичные коды операций и набор операндов, однако не содержат поля тегов. В настоящий момент реализовано 19 команд обработки множеств и структур данных, доступных для Центрального процессора (команда JT в режиме сопроцессора недоступна). В таблице 2.21 представлены форматы команд микропроцессора.

Таблица 2.21: Форматы команд микропроцессора Leonhard x64

Команда	**Мнемокод**	**Операнды***			**Результаты***			**СОПР****	**МКОД*****
Поиск	SEARCH,SRCH	R	K	-	C	K	З	+	+
Добавление	INSERT,INS	R	K	З	C	K	З	+	+
Удаление	DELETE,DEL	R	K	-	C	K	З	+	+
Удаление структуры	DELS	R	-	-	C	-	-	+	+
Максимум	MAX	R	-	-	C	K	З	+	+
Минимум	MIN	R	-	-	C	K	З	+	+
Мощность	CARDINALITY,C	R	-	-	C	K	З	+	+
И	AND	R	A	B	C	-	-	+	+
ИЛИ	OR	R	A	B	C	-	-	+	+
НЕ	NOT	R	A	B	C	-	-	+	+
Срез LS	LS	R	A	K	C	-	-	+	+
Срез LSEQ	LSEQ	R	A	K	C	-	-	+	+
Срез GR	GR	R	A	K	C	-	-	+	+
Срез GREQ	GREQ	R	A	K	C	-	-	+	+
Следующий	NEXT	R	K	-	C	K	З	+	+
Ближайший больший	NGR	R	K	-	C	K	З	+	+
Ближайший меньший	NSM	R	K	-	C	K	З	+	+
Сжатие	SQUEEZE,SQ	R	-	-	C	-	-	+	+
Переход по тегу	JT	Тег	Адрес	-	-	-	-	-	+

Пояснение к таблице 2.21: *R - номер структуры результата; A,B - номера исходных структур; K - ключ; З - значение; C - статус. **СОПР - команда доступна в режиме сопроцессора. ***МКОД - команда доступна в режиме МКОД.

Все указанные команды изменяют регистры статуса. По состоянию этого регистра можно определить, было ли выполнение команды успешным. В результате команд SEARCH, DELETE, MAX, MIN, NEXT NSM, NGR в очередь данных записываются ключ и значение найденных записей (KEY, DATA), которые могут быть использованы ЦП в алгоритме. Операнды R,A и B являются номерами структур, над которыми

выполняются команды: операнд R указывает на номер структуры, в которой будет сохранен результат; структуры A и B используются в И-ИЛИ-НЕ операциях и срезах в качестве исходных.

R Для каждой команды возможно указать флаг Q в регистре команды для буферизации результата: при Q=0 результат выполнения команды не буферизируется и доступен только до получения результата следующей команды (событие "overwrite"); при Q=1 в очереди результатов сохраняется статус, ключ (при его наличии) и значение результата (при его наличии).

2.3 Программно доступные регистры

Взаимодействие ЦП и микропроцессора Leonhard x64 выполняется через программно доступные регистры. Доступ осуществляется по шине PCIe 2.0 x8 бит, режим адресации: 32 битный. Все доступные регистры отображены в пространство памяти, которое занимает 256 байт. Доступ по шине PCIe осуществляется без использования режима DMA, что связано со спецификой обработки данных Leonhard x64 (длительное исполнение команд, малый объем передаваемых данных). В связи с этим Leonhard x64 допускает единовременное обращение для записи и чтения не более 4х байт информации. В связи с этим все регистры представлены в виде 32 битных частей. В таблице 2.22 приведены программно доступные регистры.

Таблица 2.22: Программно доступные регистры Leonhard x64

Смещение	Регистр	Режим	Назначение
0x00	KEY_L	RW	Ключ[31..0]. В режиме записи: используется для указания ключа. В режиме чтения: содержит ключ результата последней выполненной операции
0x04	KEY_H	RW	Ключ[63..32]. Старшая часть ключа. Использование аналогично KEY_L
—	—	—	Не используется
0x20	VALUE_L	RW	Значение[31..0]. В режиме записи: используется для указания значения. В режиме чтения: содержит значение результата последней выполненной операции
0x24	VALUE_H	RW	Значение[63..32]. Старшая часть значения. Использование аналогично VALUE_L
—	—	—	Не используется
0x40	CMD	W	Регистр команды: используется для записи кода команды и номера структуры. Запуск команды на исполнение осуществляется при каждой записи в регистр

Таблица 2.22 (продолжение)

Смещение	Регистр	Режим	Назначение
0x40	KEYQ_L	R	Ключ[31..0]. При использовании буферизации (Q=1) ключ результата записывается в очередь результата. Регистр содержит буферизированный ключ результата.
0x44	CONTROL0	W	Регистр управления #0. Используется для настроек режимов работы Leonhard x64
0x44	KEYQ_H	R	Ключ[63..31]. Старшая часть ключа. Использование аналогично KEYQ_L.
0x48	CONTROL1	W	Регистр управления #1. Используется для сброса подсистем и управления очередями ввода/вывода
—	—	—	Не используется
0x50	LCMW_ADR	W	Адрес записи LCM: используется для указания адреса памяти команд (Local Command Memory) при инициализации LCM
0x54	LCMR_ADR	W	Адрес чтения LCM: используется для указания адреса памяти команд (Local Command Memory) при инициализации LCM
0x58	LSMW_ADR	W	Адрес записи LSM: используется для указания адреса памяти структур (Local Ctructure Memory) при инициализации LSM
0x5C	LSMR_ADR	W	Адрес чтения LSM: используется для указания адреса памяти структур (Local Ctructure Memory) при инициализации LSM
0x60	TAGx_L	W	Регистр тега [31..0]. Используется в режиме MISD для передачи данных на конвейер Leonhard x64 от ЦП. Содержит младшую часть 64 бит данных
0x60	VALUEQ_L	R	Значение[31..0]. При использовании буферизации (Q=1) значение результата записывается в очередь результата. Регистр содержит буферизированное значение результата.
0x64	VALUEQ_H	R	Ключ[63..31]. Старшая часть значения. Использование аналогично VALUEQ_L.
—	—	—	Не используется
0x80	TAG0_H	W	Регистр тега #0 [63..32]. Старшая часть. Использование совместно с младшей частью, сохраненной в TAGx_L. Передача тега в Leonhard x64 осуществляется при каждой записи в регистр.

Таблица 2.22 (продолжение)

Смещение	Регистр	Режим	Назначение
0x80	POWER	R	Мощность структуры. Содержит количество элементов структуры, указанной при выполнении команды Power(Str).
0x84	TAG1_H	W	Регистр тега #1 [63..32]. Старшая часть. Использование совместно с младшей частью, сохраненной в TAGx_L. Передача тега в Leonhard x64 осуществляется при каждой записи в регистр.
0x84	TSC_L	R	Счетчик тактов [31..0]. Используется для программной профилировки работы Leonhard x64. Запуск осуществляется в регистре CONTROL0 (бит CONTROL0.4), сброс осуществляется в регистре CONTROL1 (бит CONTROL1.3)
0x88	TAG2_H	W	Регистр тега #2 [63..32]. Старшая часть. Использование совместно с младшей частью, сохраненной в TAGx_L. Передача тега в Leonhard x64 осуществляется при каждой записи в регистр.
0x88	TSC_L	R	Счетчик тактов [64..0]. Старшая часть счетчика тактов. Используется аналогично TSC_L.
0x90	STATE0	R	Регистр статуса #0. Используется для анализа выполнения текущей операции.
0x90	STATE1	R	Регистр статуса #1. Используется для анализа работы процессора.
—	—	—	Не используется
0xA0	LCMD0	R	Регистр данных локальной памяти команд [31..0]. Используется для чтения данных из локальной память команд.
0xA4	LCMD1	R	Регистр данных локальной памяти команд [63..32]. Используется для чтения данных из локальной память команд.
0xA8	LCMD2	R	Регистр данных локальной памяти команд [95..64]. Используется для чтения данных из локальной память команд.
0xAC	LCMD3	R	Регистр данных локальной памяти команд [127..96]. Используется для чтения данных из локальной память команд.
0xB0	LCMD4	R	Регистр данных локальной памяти команд [159..128]. Используется для чтения данных из локальной память команд.
—	—	—	Не используется

Таблица 2.22 (продолжение)

Смещение	Регистр	Режим	Назначение
0xC0	LMD0	RW	Регистр данных локальной памяти [31..0] (LCM или LSM в зависимости от значения бита CONTROL0.2 и CONTROL0.3). Используется для чтения LSM и записи данных в LSM и LCM.
0xC4	LMD1	RW	Регистр данных локальной памяти [63..32] (LCM или LSM в зависимости от значения бита CONTROL0.2 и CONTROL0.3). Используется для чтения LSM и записи данных в LSM и LCM.
0xC8	LMD2	RW	Регистр данных локальной памяти [95..64] (LCM или LSM в зависимости от значения бита CONTROL0.2 и CONTROL0.3). Используется для чтения LSM и записи данных в LSM и LCM.
0xCC	LMD3	RW	Регистр данных локальной памяти [127..96] (LCM или LSM в зависимости от значения бита CONTROL0.2 и CONTROL0.3). Используется для чтения LSM и записи данных в LSM и LCM.
0xD0	LMD4	RW	Регистр данных локальной памяти [159..128] (LCM или LSM в зависимости от значения бита CONTROL0.2 и CONTROL0.3). Используется для чтения LSM и записи данных в LSM и LCM.
0xD4	LMD5	RW	Регистр данных локальной памяти [191..160] (LCM или LSM в зависимости от значения бита CONTROL0.2 и CONTROL0.3). Используется для чтения LSM и записи данных в LSM и LCM.
0xD8	LMD6	RW	Регистр данных локальной памяти [223..192] (LCM или LSM в зависимости от значения бита CONTROL0.2 и CONTROL0.3). Используется для чтения LSM и записи данных в LSM и LCM.
0xDC	LMD7	RW	Регистр данных локальной памяти [255..224] (LCM или LSM в зависимости от значения бита CONTROL0.2 и CONTROL0.3). Используется для чтения LSM и записи данных в LSM и LCM.
0xE0	LMD8	RW	Регистр данных локальной памяти [287..256] (LCM или LSM в зависимости от значения бита CONTROL0.2 и CONTROL0.3). Используется для чтения LSM и записи данных в LSM и LCM.

Таблица 2.22 (окончание)

Смещение	Регистр	Режим	Назначение
0xE4	LMD9	RW	Регистр данных локальной памяти [319..288] (LCM или LSM в зависимости от значения бита CONTROL0.2 и CONTROL0.3). Используется для чтения LSM и записи данных в LSM и LCM.
0xE8	LMD10	RW	Регистр данных локальной памяти [351..320] (LCM или LSM в зависимости от значения бита CONTROL0.2 и CONTROL0.3). Используется для чтения LSM и записи данных в LSM и LCM.
0xEC	LMD11	RW	Регистр данных локальной памяти [383..352] (LCM или LSM в зависимости от значения бита CONTROL0.2 и CONTROL0.3). Используется для чтения LSM и записи данных в LSM и LCM.
0xF0	LMD12	RW	Регистр данных локальной памяти [415..384] (LCM или LSM в зависимости от значения бита CONTROL0.2 и CONTROL0.3). Используется для чтения LSM и записи данных в LSM и LCM.
0xF4	LMD13	RW	Регистр данных локальной памяти [447..416] (LCM или LSM в зависимости от значения бита CONTROL0.2 и CONTROL0.3). Используется для чтения LSM и записи данных в LSM и LCM.
0xF8	LMD14	RW	Регистр данных локальной памяти [479..448] (LCM или LSM в зависимости от значения бита CONTROL0.2 и CONTROL0.3). Используется для чтения LSM и записи данных в LSM и LCM.
0xFC	LMD15	RW	Регистр данных локальной памяти [511..480] (LCM или LSM в зависимости от значения бита CONTROL0.2 и CONTROL0.3). Используется для чтения LSM и записи данных в LSM и LCM.

2.3.1 Регистры управления

Регистры управления CONTROL0 и CONTROL1 предназначены для настройки и сброса подсистем микропроцессора Leonhard x64. Записанное в CONTROL0 сохраняется до аппаратного сброса по сигналу RST или до отключения питания (таблица 2.23).

Таблица 2.23: Назначение разрядов регистра CONTROL0

Разряд	По умолчанию	Назначение
0	0	Разрешение MISD режима и LCM памяти. 0 - MISD режим запрещен; 1 - MISD режим разрешен.

Таблица 2.23 (окончание)

Разряд	По умолчанию	Назначение
1	0	Останов записи во входную очередь команд. 0 - запись разрешена; 1 - запись запрещена.
2	0	Разрешение DMA памяти LCM. 0 - работа LCM DMA запрещена; 1 - работа LCM DMA разрешена.
3	0	Разрешение DMA памяти LSM. 0 - работа LCM DMA запрещена; 1 - работа LCM DMA разрешена.
4	0	Разрешение работы Time stamp counter (TSC). 0 - TSC запрещен; 1 - TSC разрешен.
5	0	Разрешение прерывания по готовности данных DATA_READY_INT. Прерывание DATA_READY_INT типа Legacy передается по шине PCIe в ЦП, вызывается при завершении выполнения команды Leonhard x64 вне зависимости от ее успешности. Прерывание 0 - прерывание запрещено; 1 - прерывание разрешено.
6	0	Разрешение прерывания по переполнению очереди команд QUEUE_OVF_INT. Прерывание типа Legacy передается по шине PCIe в ЦП, вызывается при переполнении входной очереди команд и невозможности приема очередной команды. 0 - прерывание запрещено; 1 - прерывание разрешено.
—	—	Не используется
24-31	0x00	Адрес регистра системного монитора. Используется для чтения регистров System Monitor ПЛИС Virtex 6. Адрес #3 используется для получения значения от АЦП о падении напряжения на резисторе V_int FPGA. Позволяет определить потребляемый ток рассеиваемую мощьность как для FPGA, так и для платы ML605.

Регистр CONTROL1 предназначен для сброса различных устройств в составе микропроцессора и его состояние не сохраняется в микропроцессоре (таблица 2.24). Активное значение сброса устройства соответствует логической единице ("1"). Нулевое значение не приводит к сбросу. Повторная запись значения нуля ("0") в разряды регистра не требуется, т.к. производится микропроцессором автоматически.

Таблица 2.24: Назначение разрядов регистра CONTROL1

Разряд	По умолчанию	Назначение
0	0	Сброс микропроцессора и удаление всех структур.
1	0	Сброс очередей ввода/вывода (очереди между PCIe контроллером и Leonhard x64)
2	0	Сброс очереди результатов SPU2CPU
3	0	Сброс счетчика тактов TSC
4	0	Сдвиг с вытеснением одной записи очереди SPU2CPU
5	0	Сдвиг очереди данных канала DMA из памяти LCM и LSM
6	0	Сброс указателя команд IP режима MISD
7	0	Сброс флага прерывания DATA_READY_INT

Таблица 2.24 (окончание)

Разряд	По умолчанию	Назначение
8	0	Сброс флага прерывания QUEUE_OVF_INT
—	—	Не используется

2.3.2 Регистры статуса

Регистры статуса используются для контроля работы микропроцессора, чтения флагов результатов команд и т.д.

Регистр STATE0 используется в командах для контроля выполнения текущей команды.

Таблица 2.25: Назначение разрядов регистра STATE0

Разряд	Название	Назначение
0	READY	Флаг готовности к запуску команды (предыдущая команда завершена). 0 - не готов; 1 - готов.
1	ERROR	Флаг ошибки выполнения предыдущей команды. 0 - нет ошибки; 1 - ошибка.
2	ERRORQ	Буферизированный флаг ошибки из очереди результатов. Показывает флаг ошибки наиболее ранней из сохраненных в очереди результатов команд. 0 - нет ошибки; 1 - ошибка.
3	SPU2CPUQ_OWR	Произошло автоматическое вытеснение из очереди результатов. 0 - переподнения не было; 1 - было переполнение.
4	DDR_BIST	Результат теста памяти DDR3. 0 - ошибка теста BIST; 1 - BIST прошел успешно
5	CPU2SPU_FULL	Очередь команд от ЦП в Leonhard x64 переполнена. 0 - нет переполнения; 1 - переполнение очереди.
—	—	Не используется
31-16	SYSMON_DATA	Данные из регистра АЦП System Monitor

Регистр статуса STATE1 используется для контроля состоянием очередей и линий прерываний.

Таблица 2.26: Назначение разрядов регистра STATE1

Разряд	Название	Назначение
0	PCI2SYS_EMPTY	Очередь транзакций PCIe пуста
1	PCI2SYS_FULL	Очередь транзакций PCIe переполнена
2	SPU2CPU_EMPTY	Очередь результатов пуста
3	SPU2CPU_FULL	Очередь результатов переполнена
4	SYS2PCI_EMPTY	Очередь транзакций на шину PCIe пуста
5	SYS2SPU_EMPTY	Очередь команд пуста
6	DATA_READY_INT	Состояние линии прерывания DATA_READY_INT
7	QUEUE_OVF_INT	Состояние линии прерывания QUEUE_OVF_INT
—	—	Не используется

3. Приложения

3.1 Список версий

1. Версия 0.1 (17.03.2019). Начальная версия.

Литература

[1] Aleksey Popov. “An Introduction to the MISD Technology”. B: *HICSS50*. Proceedings of the 50th Hawaii International Conference on System Sciences, 2017, страницы 1003—1012 (цитируется на странице 6).