

Архитектура ADSP-21060

1. Семейство процессоров SHARC ADSP
2. Общая архитектура процессора ADSP-21060
3. Топология выводов ADSP-21060

Семейство процессоров SHARC ADSP

1. ADSP-2106x

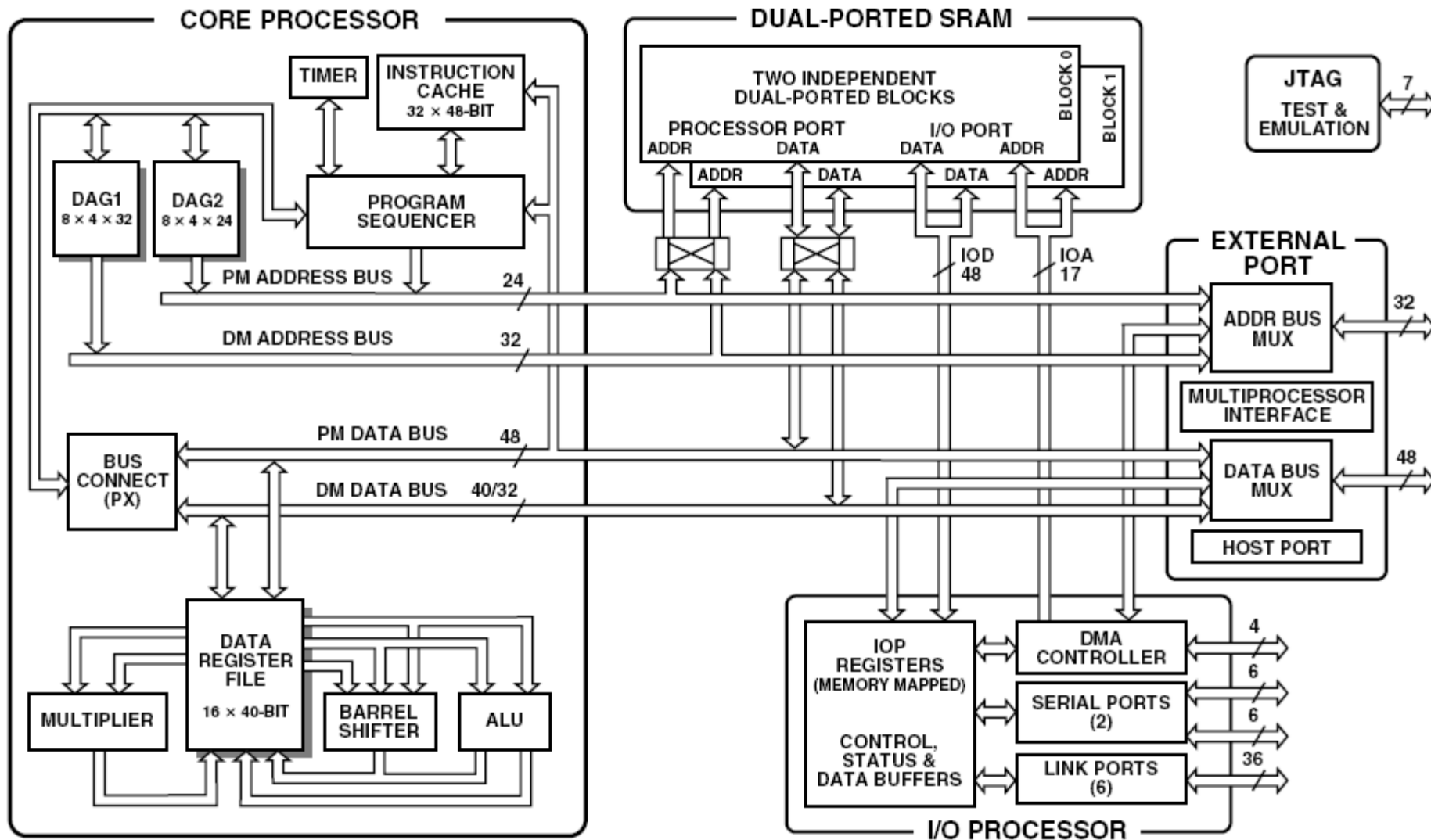
1. ADSP-21060 – базовая модель. 4 Мбит внутреннего ОЗУ. 2 последовательных и 6 линк портов (40 MIPS, 120 MFLOPS Peak)
2. ADSP-21061 – отсутствуют линк-порты. 1 Мбит внутренней памяти (50 MIPS, 120 MFLOPS Peak)
3. ADSP-21062 – 2 Мбит внутреннего ОЗУ (40 MIPS, 120 MFLOPS Peak)
4. ADSP-21065 – отсутствуют линк-порты. Поддержка интерфейсов I²S (до 8 каналов) и SDRAM. 544 Кбит внутреннего ОЗУ (66 MIPS, 198 MFLOPS Peak)

2. ADSP-2116x. ADSP-21160M: SIMD. 4 Мбит внутреннего ОЗУ. 2 последовательных и 6 линк портов (80 MIPS, 480 MFLOPS Peak)

3. ADSP-2126x. ADSP-21262: SIMD. 2 Мбит внутреннего ОЗУ. Расширенные коммуникационные возможности (200 MIPS, 1200 MFLOPS Peak)

4. ADSP-2136x. ADSP-21362: SIMD. 3 Мбит внутреннего ОЗУ. Большие коммуникационные возможности (333 MIPS, 2GFLOPS Peak)

Общая архитектура ADSP-21060

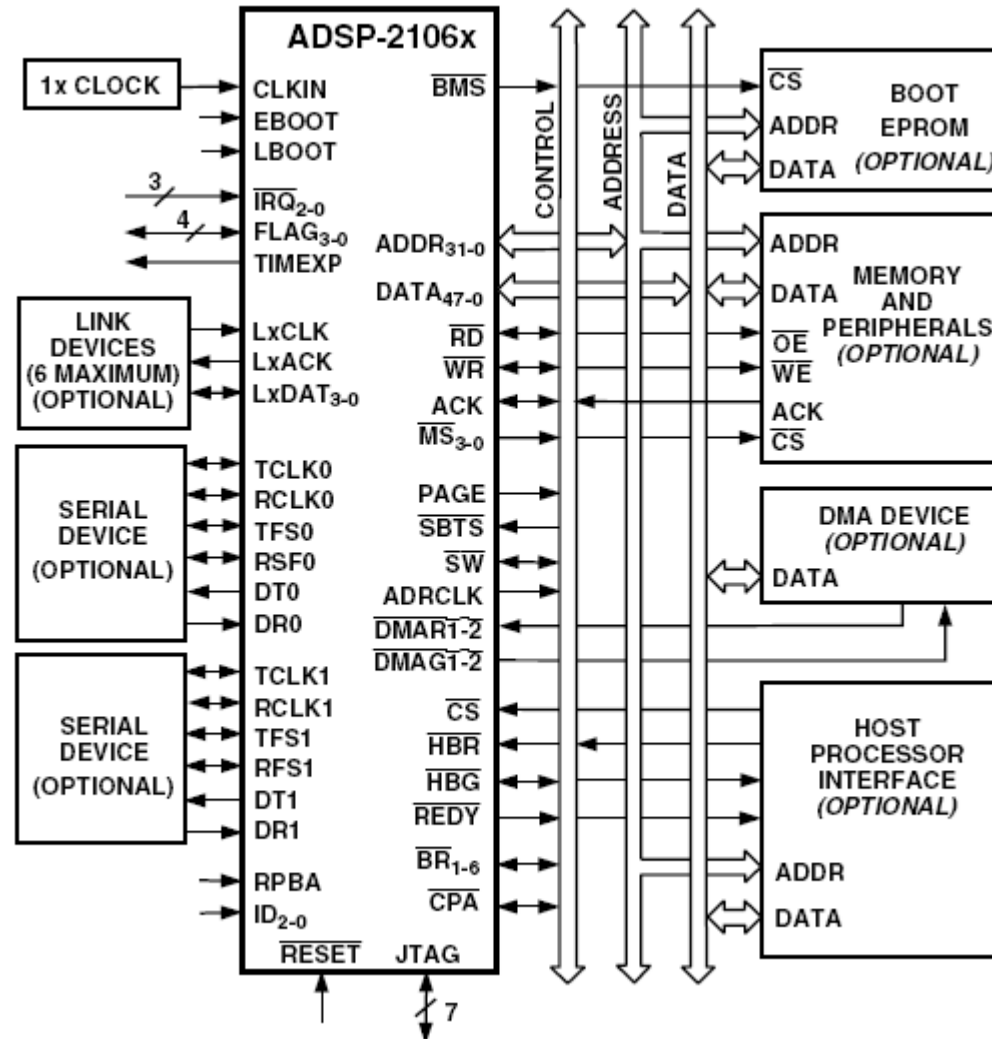


Общая архитектура ADSP-21060

Основные особенности:

- SHARC-архитектура. Поддержка PM- и DM-памяти;
- обработка 32-разрядных ФЗ и ПЗ данных;
- три независимых вычислительных блока: АЛУ (ALU), Умножитель (MAC), Сдвигатель (SHIFTER);
- регистровый файл из 16 основных и 16 альтернативных регистров;
- арифметика с насыщением и переполнением;
- независимые генераторы адресов данных для DM и PM-памяти. Поддержка до 16 буферов одновременно. Пред- и пост-модификация, кольцевые буферы, бит-реверсная адресация;
- аппаратная поддержка циклов до 6 уровней вложенности без потерь тактов;
- возможность выполнения задержанных переходов;
- кэш инструкций (на 32 инструкции);
- таймер;
- контроллер прерываний (в т.ч. вложенных);
- двухпортовая внутренняя память с независимым доступом процессорного ядра и подпроцессора ввода/вывода;
- четыре независимые 32/48-разрядные шины для доступа к внутренней памяти и выборки данных, инструкций и ввода/вывода;
- 4 Мбит внутреннего ОЗУ, конфигурируемого как 16/32 или 40/48 разрядные слова;
- полное адресуемое пространство памяти – 4 Гслов;
- встроенный подпроцессор ввода/вывода (IOP);
- 10-канальный DMA-контроллер;
- 2 синхронных последовательных (Serial) порта для подключения устройств ввода/вывода сигналов с аппаратной поддержкой А- и m-командирования;
- 6 линк-портов для межпроцессорного взаимодействия;
- встроенный интерфейс для арбитража внешней шины при организации многопроцессорной системы;
- ориентирован на высокопроизводительные приложения ЦОС в области телекоммуникаций, графики и обработки изображений

Топология выводов ADSP-21060



Организация памяти ADSP-21060

1. Организация памяти
 1. Классификация памяти: размер, разрядность, доступ процессорного ядра и IOP-процессора
 2. Структура адреса памяти
 3. Организация внутренней памяти
 4. Пространство памяти многопроцессорной системы
 5. Конфигурация и параметры доступа к внешней памяти. Интерфейс с внешней памятью
2. Адресация данных при выполнении программы
 1. Способы адресации памяти в программе на ассемблере
 2. Состав, разрядность и назначение DAG1 и DAG2
 3. Возможности регистров DAG
 4. Особенности и ограничения использования DAG-регистров
 5. PX-регистры. Обмен между шинами DMD и PMD
3. Описание карты памяти проекта в среде VisualDSP++

"Работа выполнена в рамках реализации "Программы развития ФГОУ ВПО "Южный федеральный университет" на 2007-2010гг. ("Разработка образовательных контентов и ресурсов нового поколения").

Хусаинов Н.Ш.

Технологический институт Южного федерального университета в г.Таганроге, 2007г.

Классификация памяти

1. **Internal memory.** Внутреннее статическое ОЗУ на кристалле (SRAM). Имеет размер 4 Мбит. Организовано в виде двух блоков Block0 и Block1. Имеет независимые доступ от процессорного ядра (Core) и IOP-процессора
2. **Multiprocessor memory.** Пространство памяти многопроцессорной системы. Отображает внутреннюю память других процессоров. Используется в многопроцессорной системе
3. **External memory.** Внешняя память. Адресное пространство для подключения микросхем ОЗУ (DRAM) и ПЗУ (или Flash).

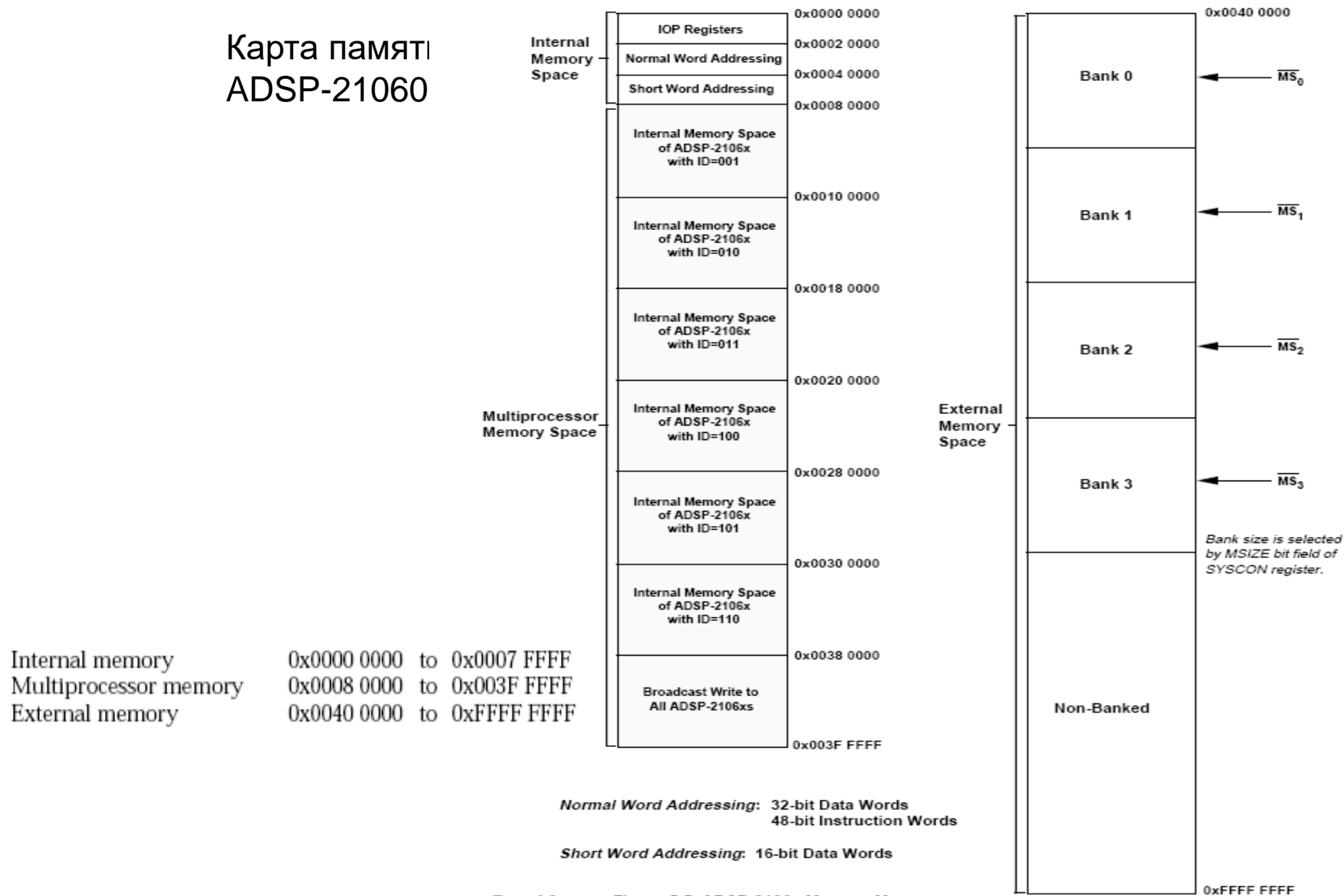
Общий размер адресуемой памяти – 4 Гслов.

Доступ к внутренней памяти осуществляется по отдельным шинам (PMA/PMD, DMA/DMD,) и IOA/IOD и может выполняться независимо.

При подключении к внешней шине адресные шины PMA, DMA, IOA и шины данных PMD, DMD, IOD мультиплексируются в единые шины адреса (ADDR) и данных (DATA).

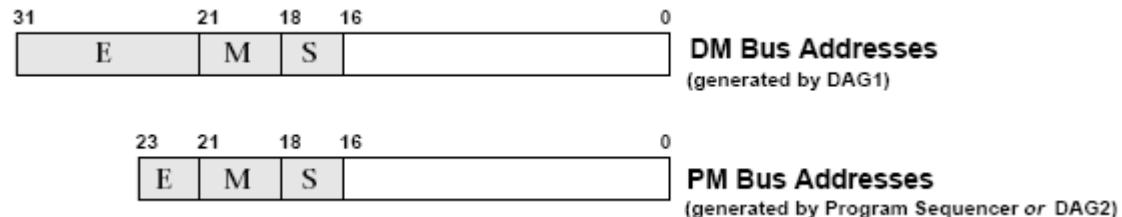
Классификация памяти

Карта памяти ADSP-21060



Классификация памяти

Структура адреса памяти:



Правила дешифрации адреса памяти:

Поле	Числовое значение	Интерпретация
E	<>0	Адрес во внешней памяти (External memory)
	=0	Адрес во внутренней памяти процессора (Internal memory) или во внутренней памяти другого процессора
M	000	Адрес во внутренней памяти процессора (Internal memory)
	<>0	Адрес во внутренней памяти процессора, чей ID = M
	111	Адрес во внутренней памяти всех процессоров (операция широковещательной записи)
S	00	Адрес IOP-регистра
	01	Адрес в пространстве нормальных слов
	1x	Адрес в пространстве коротких слов

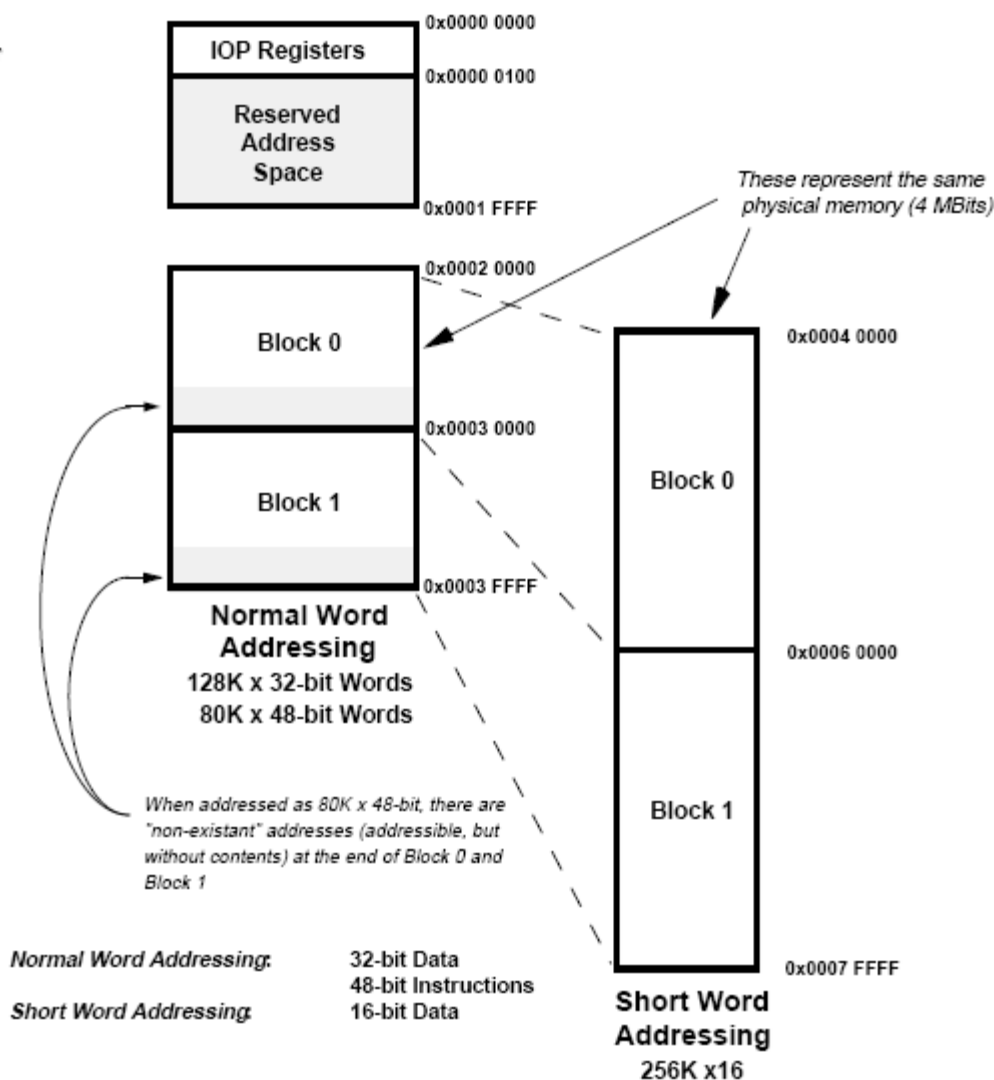
Организация внутренней памяти

Адресное пространство внутренней памяти:

- I/O Processor (IOP) Registers 0x0000 0000 to 0x0000 00FF
- Normal Word Addresses 0x0002 0000 to 0x0003 FFFF
 Interrupt Vector Table 0x0002 0000 to 0x0002 007F
- Short Word Addresses 0x0004 0000 to 0x0007 FFFF

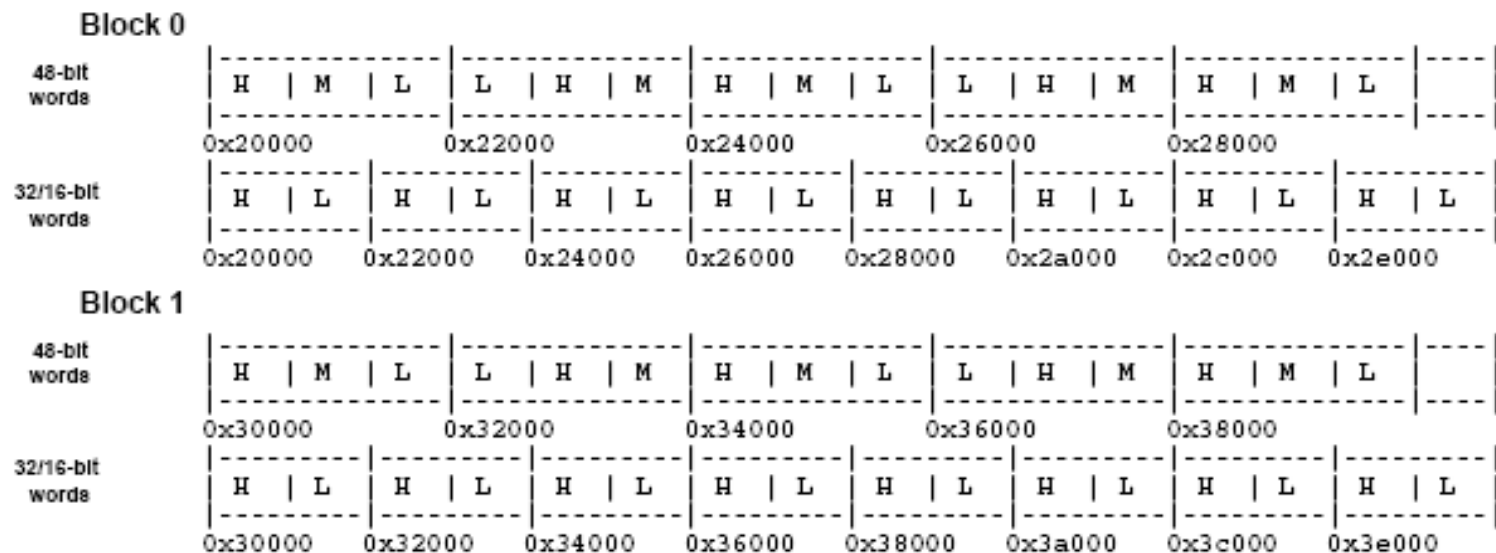
Соответствие
адресов блокам
внутренней памяти:

0x0000 0000 – 0x0000 00FF	IOP Registers (control/status registers)
0x0000 0100 – 0x0001 FFFF	Reserved addresses
0x0002 0000 – 0x0002 FFFF	Block 0 – Normal Word Addressing (32-bit, 48-bit)
0x0003 0000 – 0x0003 FFFF	Block 1 – Normal Word Addressing (32-bit, 48-bit)
0x0004 0000 – 0x0005 FFFF	Block 0 – Short Word Addressing (16-bit words)
0x0006 0000 – 0x0007 FFFF	Block 1 – Short Word Addressing (16-bit words)



Организация внутренней памяти

Физическая организация внутренней памяти:



48-битные слова:

40K слов

32-битные слова:

64K слов

16-битные слова:

128K слов

3 столбца на группу -> 5 групп -> 8K x 5 =

2 столбца на группу -> 8 групп -> 8K x 8 =

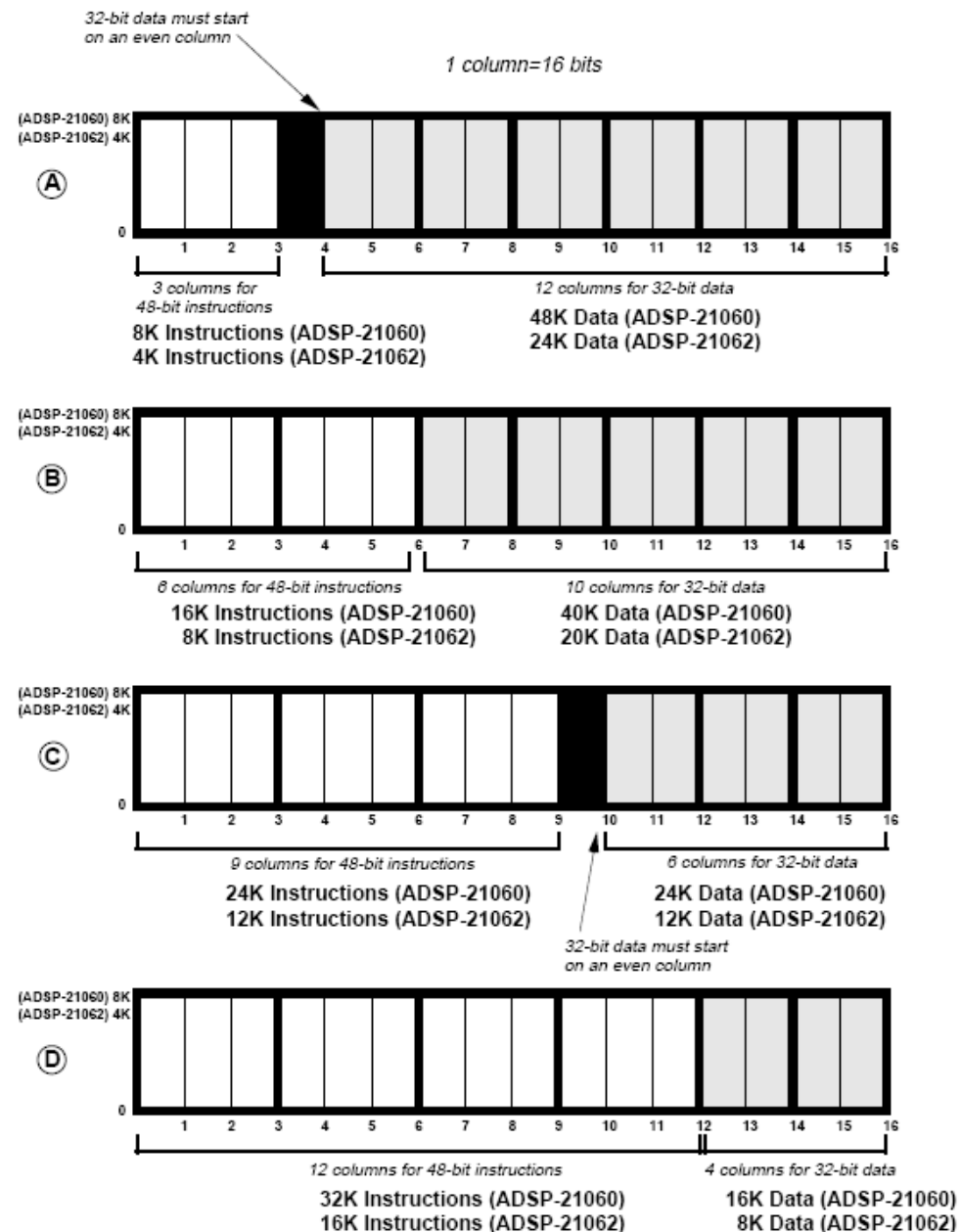
1 столбец на группу -> 16 групп -> 8K x 16 =

Организация внутренней памяти

Ограничения при размещении во внутренней памяти информации (инструкций и данных) различной разрядности:

Общие правила смешивания:

- инструкции (48-битные) должны храниться с наименьших адресов;
- данные (32-битные) должны начинаться с четного столбца.

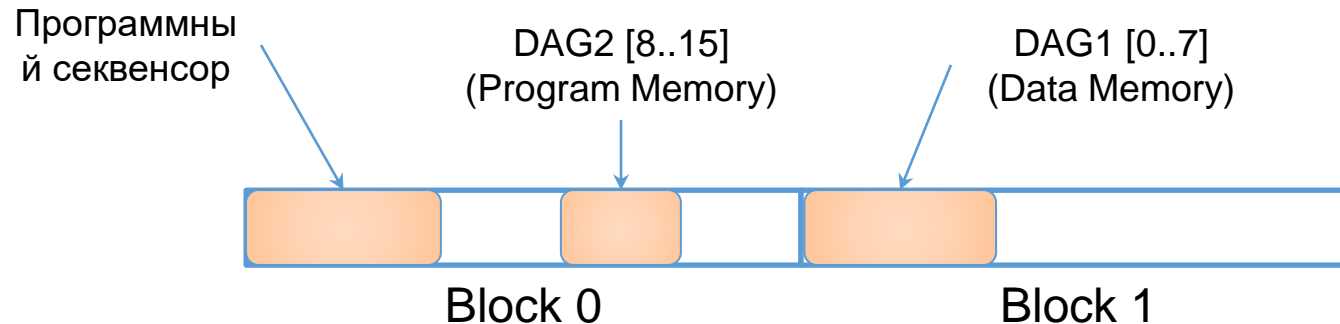


Организация внутренней памяти

Правила размещения сегментов и организации доступа к внутренней памяти в программе:

1. Два адреса, по которым идет обращение в один момент времени инструкции, должны быть расположены в разных блоках памяти
2. Один адрес данных должен генерироваться DAG1, а другой – DAG2
3. Адрес, генерируемый DAG1 не должен указывать на тот же блок памяти, из которого читаются инструкции
4. Инструкция по возможности должна иметь вид:

`compute, Rx = DM(I0-7, M0-7), Ry = PM(I8-15, M8-15);`



Организация внутренней памяти

Конфликты при доступе к внутренней памяти (приводят к дополнительным циклам ожидания):

- конфликт использования одной шины;
- конфликт использования одного блока;

Примеры нежелательного размещения сегментов и организации доступа к памяти в программе:

Размещение	Пример инструкции	Комментарии
Буфер № 1 (I ₁ , M ₁) в Block0, Инструкции в Block0	<code>compute, R_x = DM(I₁, M₁);</code>	Конфликт доступа по разным шинам к одному блоку
Буфер № 2 (I ₈ , M ₈) в Block1, Инструкции в Block0	<code>compute, R_x = PM(I₈, M₈);</code>	Конфликт доступа по одной шине к разным блокам

Пространство памяти многопроцессорной системы

- используется для отображения на единое адресное пространство (в многопроцессорной системе) внутренней памяти каждого процессора;
- для доступа к области памяти другого процессора через Multiprocessor Memory Space используется общая внешняя шина;
- процессор определяет, что выставленный на внешнюю шину адрес попадает в его внутреннюю память если $ID_{2-0} = M$;
- при обращении к собственной памяти через MMS внешняя шина не используется;
- если поле M адреса = 111, то выполняется операция “broadcast write”;
- MMS доступен только с использованием процессорного ядра.

Конфигурация и параметры доступа к внешней памяти. Интерфейс с внешней памятью

Пространство внешних адресов включает пространство памяти многопроцессорной системы (Multiprocessor Memory Space) и пространство внешней памяти (External Memory Space).

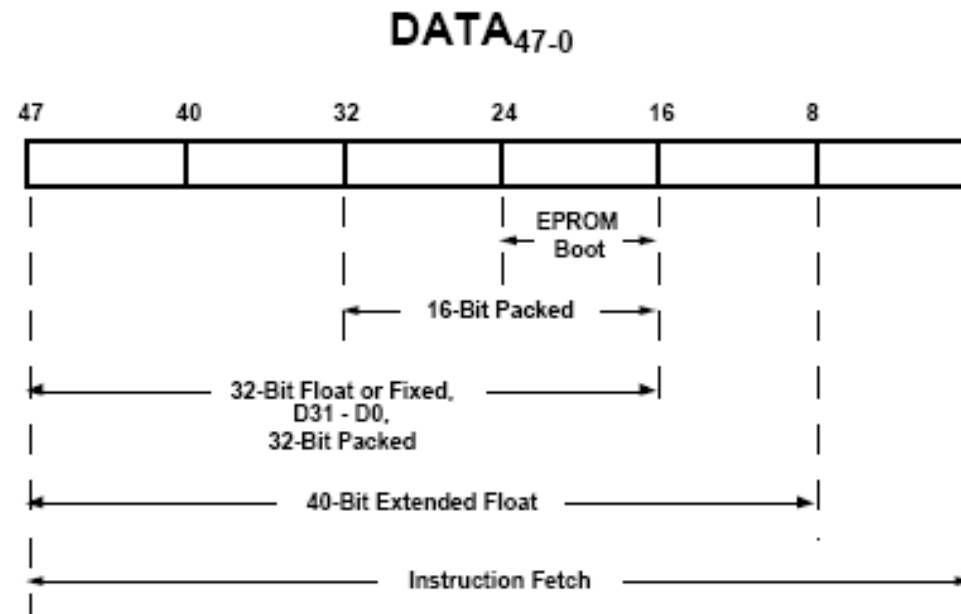
Пространство внешней памяти используется для подключения внешних микросхем памяти (ОЗУ) и ПЗУ/Flash.

Имеется возможность взаимодействия с микросхемами типа SRAM и DRAM.

Пространство внешней памяти доступно для процессорного ядра и IOP-процессора.

Конфигурация и параметры доступа к внешней памяти. Интерфейс с внешней памятью

Передача данных по внешней шине



Конфигурация и параметры доступа к внешней памяти. Интерфейс с внешней памятью

Организация внешней памяти

Внешняя память организована в виде 4-х банков одинакового размера и «небанкируемой» (unbanked) памяти.

Банк № 0 начинается с адреса 0x0040 0000.

Все банки имеют одинаковый размер. Размер кодируется 4-битовым полем MSIZE:

$$\text{Размер_банка} = 2^{(\text{MSIZE} + 13)}$$

Выбор необходимого банка памяти (микросхемы) осуществляется сигналом на линиях MS₃₋₀.

Область небанкируемой памяти расположена после последнего банка (доступ к ней – при MS₃₋₀ = 0000).

Конфигурация и параметры доступа к внешней памяти. Интерфейс с внешней памятью

Зависимость размера банка памяти от значения поля

MSIZE	
0	8
1	16
2	32
3	64
4	128
5	256
6	512
7	1024
8	2048
9	4096
10	8192
11	16384
12	32768
13	65536
14	131072
15	262144

$R0 = 0xuuuuXuuu;$

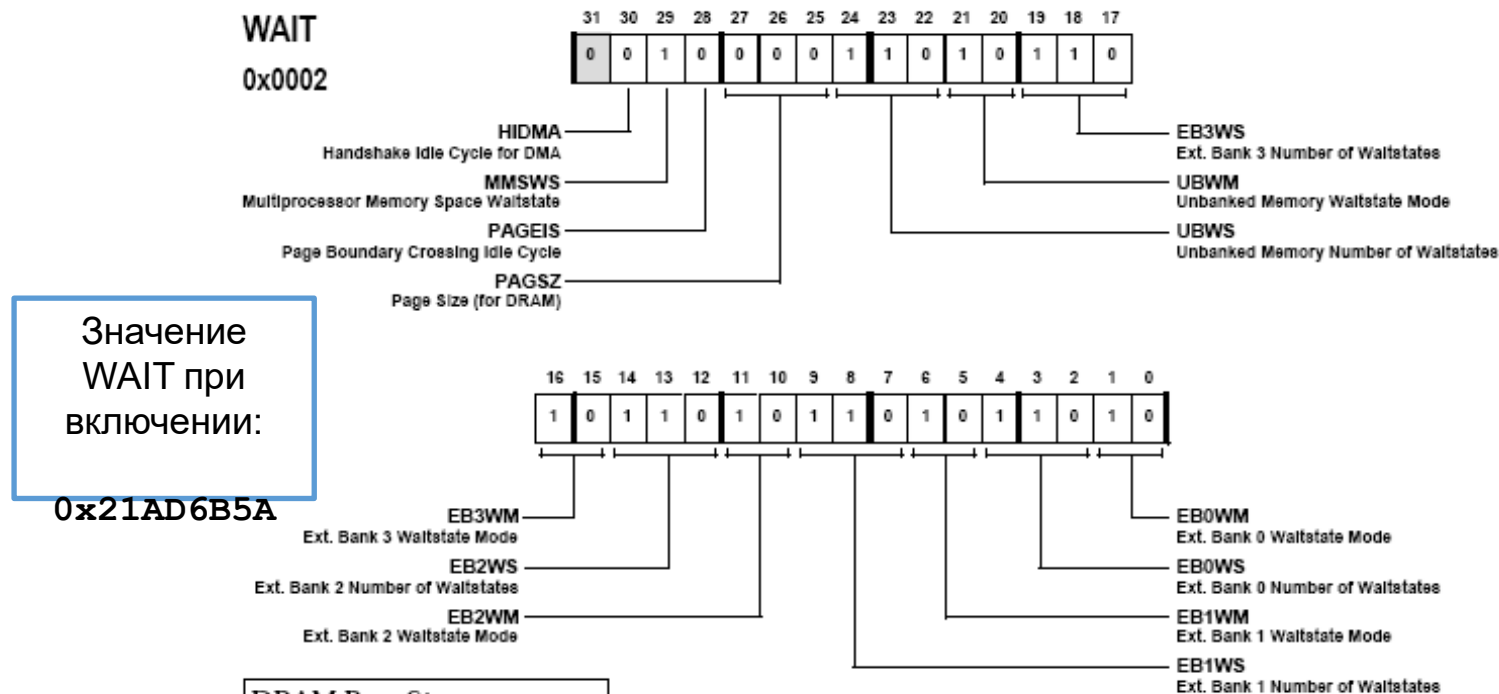
$dm(SYSCON) = R0;$

Значение SYSCON при
включении:

0x00000010

Конфигурация и параметры доступа к внешней памяти. Интерфейс с внешней памятью

Конфигурирование внешней памяти. IOP-регистр WAIT



DRAM Page Size	
PAGSZ	DRAM Page Size
000	256 words
001	512 words
010	1024 words (1K)
011	2048 words (2K)
100	4096 words (4K)
101	8192 words (8K)
110	16384 words (16K)
111	32768 words (32K)

All control and status bits are active high unless otherwise noted. Default bit values after reset are shown; if no value is shown, the bit is undefined at reset or depends upon processor inputs. Reserved bits are shown with a gray background. Reserved bits should always be written with zeros.

Конфигурация и параметры доступа к внешней памяти. Интерфейс с внешней памятью

Режимы доступа к банкам внешней памяти

При взаимодействии с «медленными» микросхемами внешней памяти для описания времени задержки и режима доступа могут использоваться поля EBxWM и EBxWS:

- EBxWS – количество дополнительных тактов ожидания (wait states) при доступе к банку внешней памяти (000b...110b);

- EBxWM – режим доступа к банку внешней памяти (000b...11b):

Режим (000b...11b)	EBxWM	Описание
External	00	Ожидание сигнала ACK
Internal	01	Доступ всегда выполняется (EBxWS+1) тактов
Both	10	Ожидание совместного выполнения первых двух условий
Either	11	Ожидание выполнения хотя бы одного из первых двух условий

Адресация данных при выполнении программы. Режимы адресации

Адресация данных

Режим адресации	Примеры
Прямая	DM(TempA) = R5; R0 = PM(0x20300);
Косвенная	PM(I8,2) = R9; R15 = DM(I0, M3);
Непосредственная	DM(I0,M6) = 0x12345678; R5 = 12;

Адресация при переходах/вызовах

Режим адресации	Примеры
Прямая	jump LabelMaxA; if EQ call 0x20040;
Косвенная	jump (M8, I9);
Относительная	call (PC, -400);

Состав, разрядность и назначение DAG1 и DAG2

DAG1 registers (32-bit)

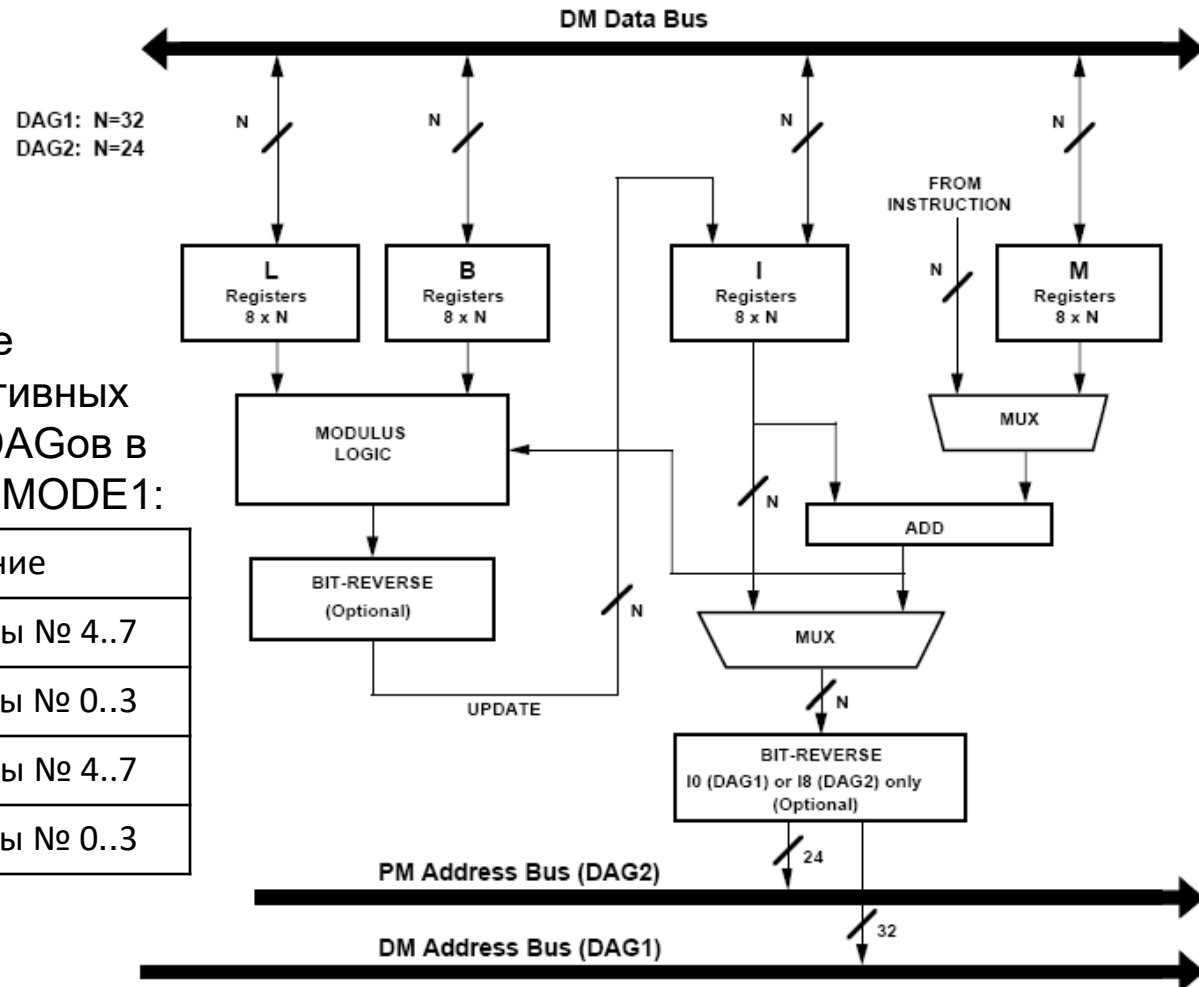
B0-B7
I0-I7
M0-M7
L0-L7

DAG2 registers (24-bit)

B8-B15
I8-I15
M8-M15
L8-L15

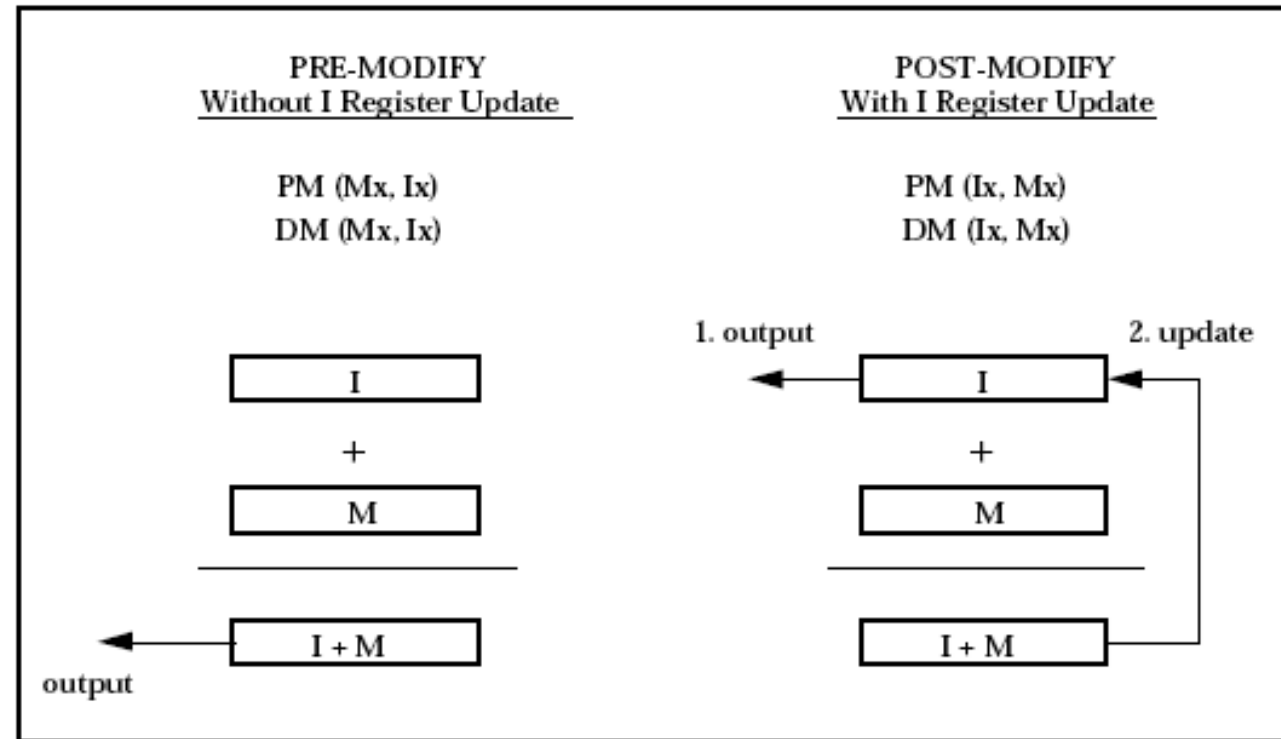
Переключение
основных/альтернативных
наборов регистров DAGов в
регистре управления MODE1:

Бит	Назначение
SRD1H	DAG1, регистры № 4..7
SRD1L	DAG1, регистры № 0..3
SRD2H	DAG2, регистры № 4..7
SRD2L	DAG2, регистры № 0..3



Возможности регистров DAG

Пред- и пост- модификация регистра указателя



Непосредственная модификация регистра указателя (без доступа к памяти:

```
MODIFY (I6, 32);
```


Возможности регистров DAG

Бит-реверсная адресация

Реверсирование битов адреса может осуществляться двумя способами:

1) с помощью инструкции BITREV над любым I-регистром DAG:

```
I0 = 0x80400000;  
BITREV (I4, 7);    // I4 <= 0xE0000201
```

2) в режиме автоматического реверсирования адреса (только для регистров I0 и I8):

```
I0 = 0x80400000;  
R1 = DM(I0, 3);    // DMA-шина <= 0x201, I0 <= 0x80400003
```

Включение режима
реверсирования адреса в
регистре управления MODE1:

Бит	Назначение
BR8	DAG2, регистр № 8
BR0	DAG1, регистр № 0

Возможности регистров DAG

Циклические буферы

- циклический доступ к массиву в режиме косвенной адресации с пост-модификацией указателя:

$$R_x = DM(I_a, M_b) ;$$

- автоматический контроль выхода за границы массива (при $L_x > 0$):

If M is positive,

$$I_{new} = I_{old} + M \qquad \text{if } I_{old} + M < \text{Buffer base} + \text{length (end of buffer)}$$

$$I_{new} = I_{old} + M - L \qquad \text{if } I_{old} + M \geq \text{Buffer base} + \text{length (end of buffer)}$$

If M is negative,

$$I_{new} = I_{old} + M \qquad \text{if } I_{old} + M \geq \text{Buffer base (start of buffer)}$$

$$I_{new} = I_{old} + M + L \qquad \text{if } I_{old} + M < \text{Buffer base (start of buffer)}$$

- генерация сигнала $\begin{matrix} (for\ M<0) & I + M < B \\ (for\ M\geq 0) & I + M \geq B + L \end{matrix}$ три «зацикливания» указателя:

Прерывания, связанные с регистрами DAG:

Описание прерывания	DAG-регистры	Вектор прерывания
Переполнение кругового буфера # 7 (DAG1)	B7, L7, I7	CB7I
Переполнение кругового буфера # 15 (DAG2)	B15, L15, I15	CBI15

Ограничения при работе с DAG-регистрами

1) Дополнительный цикл ожидания:

```
L2 = 8 ;
```

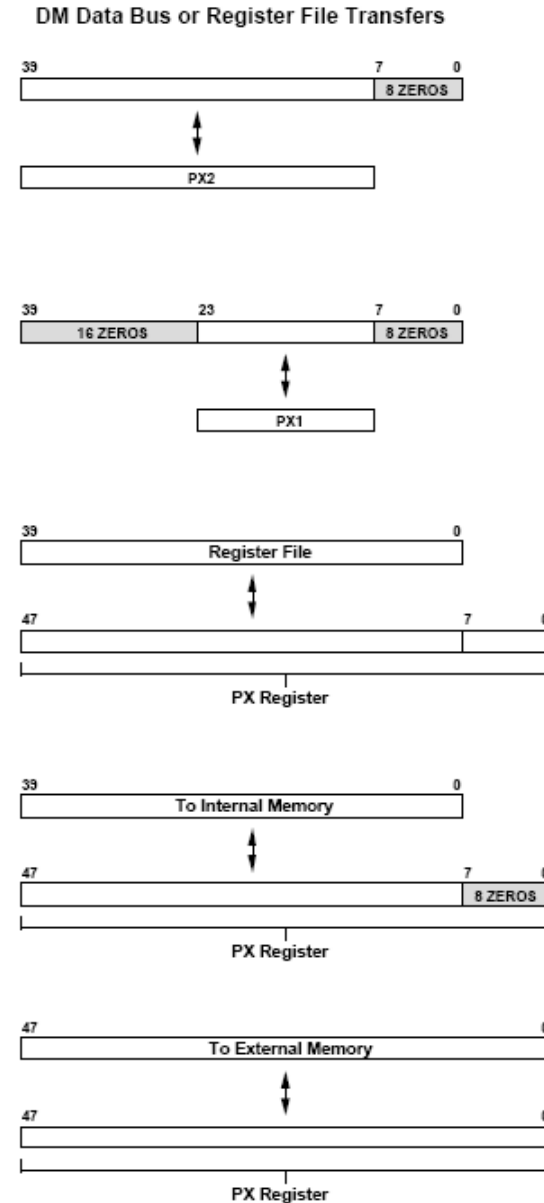
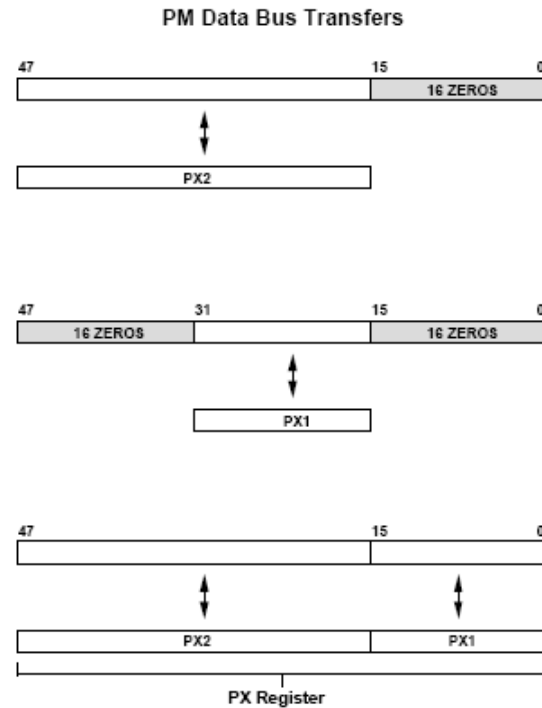
```
DM(I0, M1) = R1 ;
```

2) Некорректное выполнение (запрещены Ассемблером):

```
DM(M2, I1) = I0 ;
```

```
L2 = DM(I0, M1) ;
```

Обмен между шинами DMD и PMD. PX-регистры



ВЫЧИСЛИТЕЛЬНЫЕ БЛОКИ ADSP-21060

1. Структура, возможности и режимы работы
 1. Состав вычислительных блоков
 2. Особенности обработки ФЗ- и ПЗ-чисел. Установки
2. Арифметико-логическое устройство (ALU)
 1. Типы операций
 2. Флаги состояния и режимы работы
3. Умножитель (MULTIPLIER)
 1. Типы операций
 2. Регистр аккумулятора умножителя
 3. Суффиксы команды для операции умножителя
 4. Флаги состояния и режимы работы
4. Сдвигатель (SHIFTER)
 1. Типы операций
 2. Операции выделения и депонирования битовых полей
 3. Флаги состояния и режимы работы
5. Многофункциональные вычисления
6. Регистровый файл
7. Системные регистры

"Работа выполнена в рамках реализации "Программы развития ФГОУ ВПО "Южный федеральный университет" на 2007-2010гг. ("Разработка образовательных контентов и ресурсов нового поколения").

Хусаинов Н.Ш.

Технологический институт Южного федерального университета в г.Таганроге, 2007г.

Состав вычислительных блоков

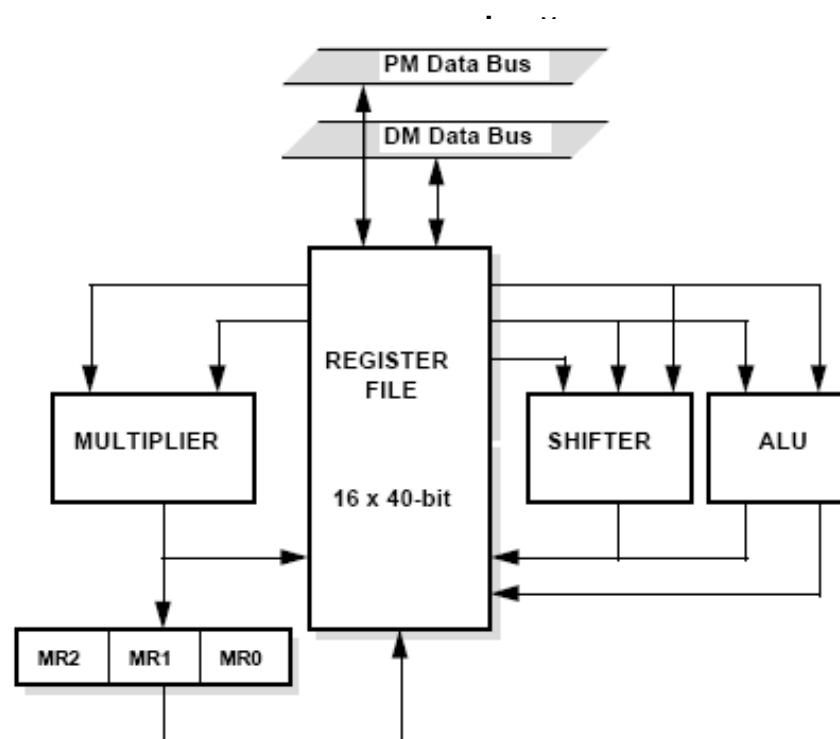
3 параллельных вычислительных блока (ALU, MULTIPLIER, SHIFTER).

Выполнение всех вычислительных операций за 1 такт.

Работа с ФЗ-данными и ПЗ-данными обычной и повышенной точности.

Читают опер
в регистр

ают результат



Особенности обработки ФЗ- и ПЗ-чисел. Установки

ФЗ- и ПЗ-данные хранятся в одних и тех же регистрах.

Тип операнда (ПЗ или ФЗ) и используемая логика вычислительных блоков задается префиксом регистра:

```
F0 = F1 + F2;    // ПЗ-сложение
```

```
R0 = R1 + R2;    // Ф3-сложение
```

Префикс не влияет на прямые пересылки данных между регистрами:

$$F0 = F1; \qquad \Leftrightarrow \qquad R0 = R1;$$

и между регистрами и памятью:

$$F0 = dm(i0, m0); \quad \Leftrightarrow \quad R0 = dm(i0, m0);$$

ФЗ-данные и ПЗ-данные одинарной точности хранятся в старших 32-х битах 40-битного регистра (ФЗ-поле).

ПЗ-данные повышенной точности занимают все 40 битов регистра.

39	7
8	0
MSB	LSB
ФЗ-данные, ПЗ-данные один.точ.	0
ПЗ-данные повышенной точности	

Особенности обработки ФЗ- и ПЗ-чисел. Установки

Выбор режима округления

Режим округления для ПЗ-операций АЛУ и умножителя выбирается путем установки бита TRUNC в системном регистре MODE1:

- TRUNC=1 – «округление к нулю»;
- TRUNC=0 – «округление к ближайшему»

Выбор формата ПЗ-чисел

Формат ПЗ-чисел выбирается путем установки бита RND32 в системном регистре MODE1:

- RND32=1 – «одинарная точность»;
- RND32=0 – «повышенная точность»

Арифметико-логическое устройство

Типы операций

- ФЗ/ПЗ-операции: сложение, вычитание, дуальное сложение/вычитание, среднее;
- ФЗ-операции: сложение с переносом, вычитание с заемом, декремент;
- ПЗ-операции: масштабирование, извлечение мантиссы;
- логические побитовые операции: AND, OR, XOR, NOT;
- функции: модуль числа, мин, макс, клиппирование, сравнение;
- форматные: преобразование ПЗ <-> ФЗ;
- приближенное вычисление обратного числа и квадратного корня.

Принцип функционирования АЛУ

АЛУ читает 1 или 2 входных операнда (регистры РФ) в начале такта.

АЛУ записывает 1 или 2 результата в регистр(ы) РФ в конце такта, или просто модифицирует флаги. Один и тот же регистр РФ может выступать в качестве операнда и приемника результата для любой операции АЛУ.

Режимы работы АЛУ

Переключение режима работы АЛУ выполняется путем установки бита ALUSAT в регистре управления MODE1:

- ALUSAT=1 – «режим арифметики с насыщением»;
- ALUSAT=0 – «режим арифметики с циклическим переносом»

Арифметико-логическое устройство

Флаги состояния, на которые влияет выполнение операций АЛУ

Флаг	Определение	Когда устанавливается
В регистре ASTAT		
AZ	Флаг нуля	Результат АЛУ=0, потеря значимости при ПЗ-операции, потеря значимости при преобразовании ПЗ<->ФЗ
AV	Флаг переполнения	Для ФЗ-операций – по обычным правилам. Для ПЗ-операций – если $e > 127$
AN	Флаг отрицательного результата	Для ФЗ и ПЗ операций
AC	Флаг переноса	Для ФЗ-операций
AS	Флаг знака входного операнда	Если входной операнд < 0 (для операций ABS и MANT)
AI	Флаг некорректной ПЗ-операции	Входной операнд = NAN Попытка вычесть бесконечности одного знака...
AF	Флаг ПЗ-операции	Если последняя операция – ПЗ-операция
CACC	Флаг аккумулирующих сравнений	Только для операции COMP с ФЗ/ПЗ-операндами

Арифметико-логическое устройство

Липкие флаги состояния, на которые влияет выполнение операций АЛУ

Флаг	Определение	Когда устанавливается
В регистре STKY		
AUS	Липкий флаг потери значимости	Только для ПЗ-операций и преобразований ПЗ<->ФЗ
AVS	Липкий флаг ПЗ-переполнения	Только для ПЗ-операций – если $e > 127$
AOS	Липкий флаг ФЗ-переполнения	Только для ФЗ-операций
AIS	Липкий флаг некорректной ПЗ-операции	Только для ПЗ-операций

Умножитель

Типы операций

- ФЗ/ПЗ-операции: умножение, умножение с накоплением (сложением или вычитанием) и округлением результата (опц.);
- округление регистра результата;
- насыщение регистра результата;
- очистка регистра результата.

Принцип функционирования Умножителя

Умножитель читает 2 входных операнда (регистры РФ) в начале такта.

Умножитель записывает 1 результат в регистр аккумулятора или в регистр РФ в конце такта и модифицирует флаги.

Типы ФЗ-операндов и необходимость округления результата указываются в суффиксе команды.

Тип результата соответствует типу входных операндов.

Один и тот же регистр РФ может выступать в качестве операнда и приемника результата для любой операции Умножителя.

Особенности перемножения знаковых дробных ФЗ-чисел

Если оба ФЗ-операнда – знаковые дробные числа, то Умножитель автоматически сдвигает результата влево на 1 бит чтобы удалить избыточный знаковый бит.

Умножитель

Расширенная разрядность регистра аккумулятора Умножителя для операций с ФЗ-числами

- наличие расширения «вправо» приводит к сокращению числа округления после операций умножения – следовательно уменьшается ошибка вычислений:

С округлением после каждого шага	С одним округлением в конце
$(0,4 * 0,7) + (0,4 * 0,9)$ $0,3 + 0,4 = 0,7$	$(0,4 * 0,7) + (0,4 * 0,9)$ $0,28 + 0,36 = 0,64 (0,6)$

- наличие расширения «влево» приводит к сохранению промежуточного результата, когда он превышает верхнюю границу 32-битного числа

С отбрасыванием возможного переноса после каждого сложения	С отбрасывание возможного переноса в конце
$(0,9 * 0,8) + (0,8 * 0,7) - (0,6 * 0,7)$ $0,72 + 0,56 - 0,42$ $0,99 - 0,42 = 0,57$ или $0,28 - 0,42 = -0,24$	$(0,9 * 0,8) + (0,8 * 0,7) - (0,6 * 0,7)$ $0,72 + 0,56 - 0,42$ $1,28 - 0,42 = 0,86$

Умножитель

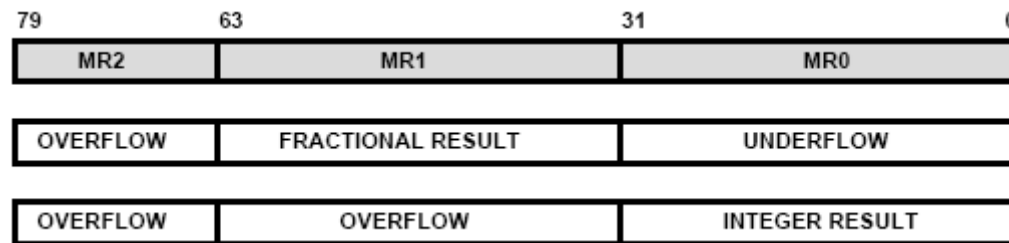
Регистр аккумулятора Умножителя

ФЗ-результат операции умножителя может быть записан как в регистр РФ, так и в регистр аккумулятора Умножителя.

Для ПЗ-операций регистр аккумулятора не используется.

Регистр аккумулятора - 80 битный регистр, позволяющий избежать ошибок вследствие операций округления во время промежуточного суммирования перемноженных значений.

Размещение результата в регистра аккумулятора определяется его типом:



Поля регистра аккумулятора могут быть записаны/переданы в регистр РФ. При записи значения в MR1x выполняется знаковое расширение на MR2. При записи значения в MR0x и MR2x такое расширение не выполняется.

Умножитель содержит 2 идентичных регистра аккумулятора: MRF и MRB, активных одновременно. Выбор регистра осуществляется в инструкции.

Умножитель

Операции с регистром аккумулятора Умножителя

- очистка:

$$\text{MRF} = 0;$$

- округление (к ближайшему по 32-битной границе):

$$\text{MRF} = \text{RND } \text{MRF} \text{ (SF / UF) ;}$$

$$\text{R1} = \text{RND } \text{MRB} \text{ (SF / UF) ;}$$

- насыщение:

$$\text{MRB} = \text{SAT } \text{MRB} \text{ (SI / UI / SF / UF) ;}$$

$$\text{R4} = \text{SAT } \text{MRF} \text{ (SI / UI / SF / UF) ;}$$

При насыщении в зависимости от типа значения (указывается в суффиксе) используются следующие границы диапазонов:

<u>MR2</u>	<u>MR1</u>	<u>MR0</u>	
<i>Maximum twos-complement fractional number</i>			
0000	7FFF FFFF	FFFF FFFF	positive
FFFF	8000 0000	0000 0000	negative
<i>Maximum twos-complement integer number</i>			
0000	0000 0000	7FFF FFFF	positive
FFFF	FFFF FFFF	8000 0000	negative
<i>Maximum unsigned fractional number</i>			
0000	FFFF FFFF	FFFF FFFF	
<i>Maximum unsigned integer number</i>			
0000	0000 0000	FFFF FFFF	

Умножитель

Флаги состояния, на которые влияет выполнение операций Умножителя

Флаг	Определение	Когда устанавливается
В регистре ASTAT		
MN	Флаг потери значимости	Для ФЗ-операций – если $e < -126$. Для ФЗ-операций <u>с дробными числами</u> : <ul style="list-style-type: none">• знаковые: $MR2=MR1=0$ (или 1), $MR0 \neq 0$;• беззнаковые: $MR2=MR1=0$, $MR0 \neq 0$.
MV	Флаг переполнения	Для ПЗ-операций – если $e > 127$. Для ФЗ-операций – см. выше.
MU	Флаг отрицательного результата	Для ФЗ и ПЗ операций
MI	Флаг некорректной ПЗ-операции	Для ПЗ-операций если один из операндов=NAN или выполняется $\text{Infinity} * \text{Zero}$

Умножитель

Липкие флаги состояния, на которые влияет выполнение операций Умножителя

Флаг	Определение	Когда устанавливается
В регистре STKY		
MOS	Липкий флаг ФЗ-переполнения	Только для ФЗ-операций
MVS	Липкий флаг ПЗ-переполнения	Только для ПЗ-операций
MUS	Липкий флаг потери значимости	Для ФЗ- и ПЗ-операций
MIS	Липкий флаг некорректной ПЗ-операции	Только для ПЗ-операций

Сдвигатель

Типы операций (только с ФЗ-данными)

- сдвиги;
- операции с битом (установка, сброс, инверсия, тестирование);
- операции над битовым полем (выделение и депонирование);
- поддержка преобразования ФЗ <-> ПЗ.

Принцип функционирования Сдвигателя

Умножитель читает от 1 до 3 входных операнда (регистры РФ или константы в инструкции) в начале такта.

Сдвигатель записывает 1 результат в регистр РФ в конце такта и модифицирует флаги.

Один и тот же регистр РФ может выступать в качестве операнда и приемника результата для любой операции Умножителя.

Некоторые операции Сдвигателя генерируют 8- или 6-битный результат, которые помещаются в младшие биты ФЗ-поля, а знаковые бит результата распространяются на «свободные» старшие разряды.

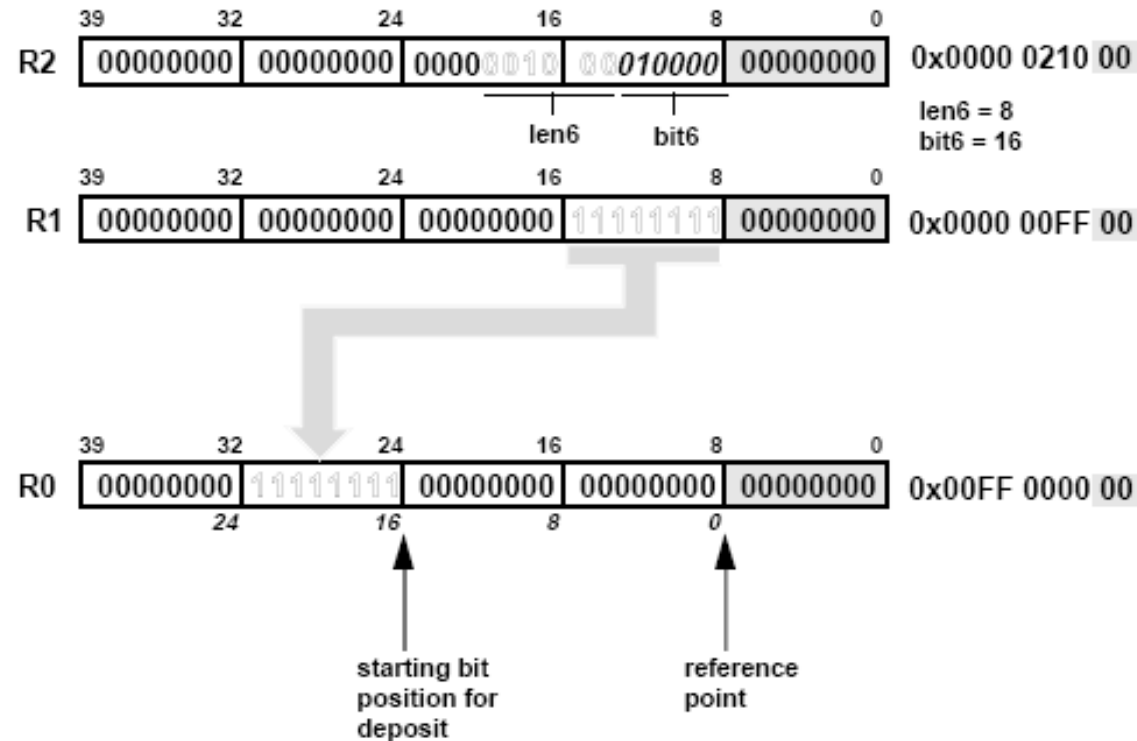
Сдвигатель

Операция депонирования битового поля

R0=FDEP R1 BY R2;

R0=FDEP R1 BY R2;

R1=0x000000FF00
R2=0x0000021000



Второй операнд содержит параметры битового поля (длину и место назначения (сдвиг) от начала ФЗ-поля).

Сдвигатель

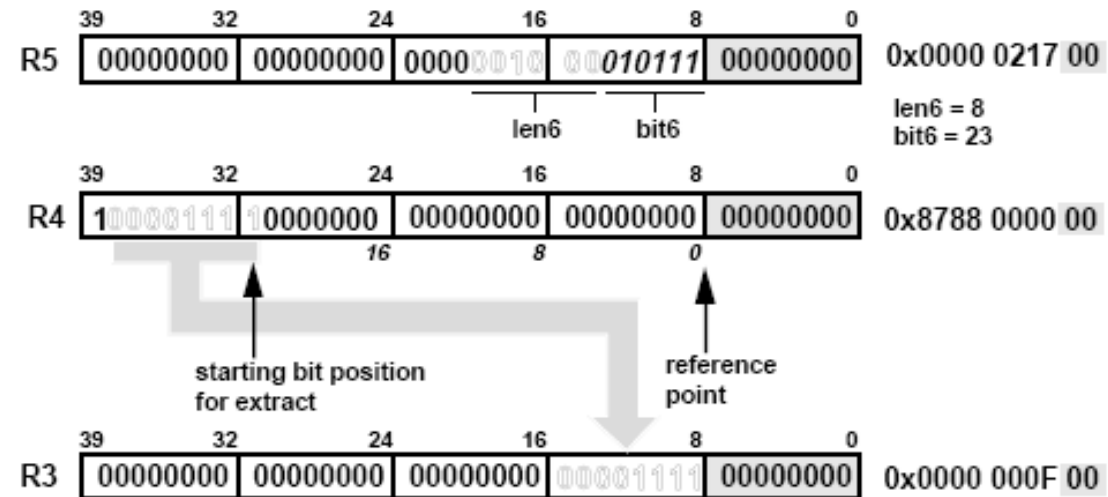
Операция выделения битового поля

R3=FEXT R4 BY R5;

R3=FEXT R4 BY R5;

R4=0x8788000000

R5=0x0000021700



Второй операнд содержит параметры битового поля (длину и место назначения (сдвиг) от начала ФЗ-поля).

Сдвигатель

Флаги состояния, на которые влияет выполнение операций Сдвигателя (только регистр ASTAT)

Флаг	Определение	Когда устанавливается
В регистре ASTAT		
SV	Флаг переполнения	<ul style="list-style-type: none">• когда значащие биты выдвигаются влево за границы ФЗ-поля;• в операции с битом указан номер бита > 31;• битовое поле частично или полностью выходит влево за границы ФЗ-поля
SZ	Флаг нулевого результата	<ul style="list-style-type: none">• результат равен нулю;• проверяемый бит равен 0;• в операции с битом указан номер бита > 31
SS	Флаг отрицательного входного операнда	Для операций выделения экспоненты устанавливается в соответствии со знаком операнда. Для остальных операций сбрасывается в 0.

Многофункциональные вычисления

Принцип выполнения многофункциональных вычислений

Многофункциональное вычисление (инструкция) – выполнение в течение одного такта:

- одной операции АЛУ и одной операции Умножителя;
- двух операций АЛУ (только дуальное сложение/вычитание).

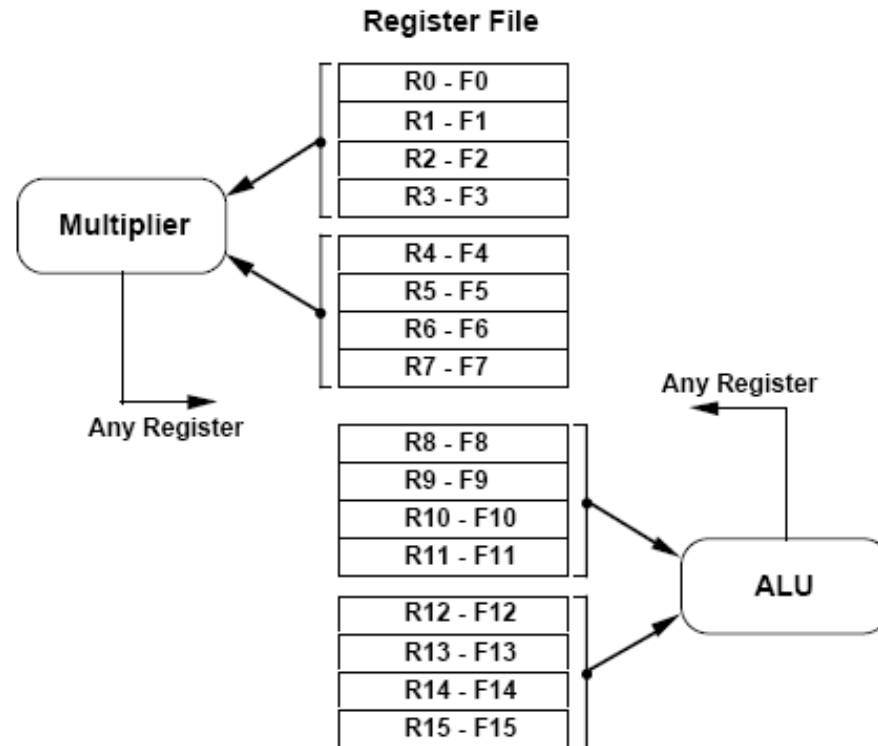
Инструкции АЛУ и Умножителя, разрешенные для одновременного выполнения, выбираются из ограниченного набора.

Наборы допустимых для распараллеливания операций различаются для ФЗ и ПЗ-операндов.

В режиме многофункциональной обработки ФЗ-данных Умножитель поддерживает только операции с дробными знаковыми числами(!).

Многофункциональные вычисления

Ограничения при размещении операндов для многофункциональных вычислений



Первый операнд АЛУ – Reg0..Reg3, второй операнд АЛУ – Reg4..Reg7.
Первый операнд Умножителя – Reg8..Reg11, второй операнд Умножителя – Reg12..Reg15.

Регистровый файл

Порядок записи значений в регистры РФ в случае одновременного доступа (в одном такте)

Приоритетность записи значений в регистры РФ (если несколько операций записывают значение в один и тот же регистр):

- значение с шины DMD или из универсального регистра;
- значение с шины PMD ;
- результат вычисления АЛУ;
- результат вычисления Умножителя;
- результат вычисления Сдвигателя.

Переключение основных/альтернативных наборов регистров РФ в регистре управления MODE1:

Бит	Назначение
SRRFH	регистры № 4..7
SRRFL	регистры № 8..15

Системные регистры

Имя	Функции	Значение
Системные регистры процессора (Core Processor)		
MODE1	Регистр управления № 1	0x0000 0000
MODE2	Регистр управления № 2 (и ID ₂₋₀)	0xn000 0000
ASTAT	Флаги арифметического статуса (и ножки FLAG ₃₋₀)	0x00nn 0000
STKY	Липкие флаги статуса	0x0540 0000
IRPTL	Регистр защелки прерывания	0x0000 0000
IMASK	Маска прерывания	0x0003
IMASKP	Указатель временной маски прерывания	0x0000 0000
USTAT1	Пользовательский регистр статуса № 1	0x0000 0000
USTAT2	Пользовательский регистр статуса № 2	0x0000 0000
Системные регистры IOP-процессора		
SYSCON	Регистр конфигурации	0x0000 0010
SYSTAT	Регистр состояния (статуса)	0x0000 0nn0
WAIT	Регистр параметров внешней памяти	0x21AD 6B5A
VIRPT	Регистр вектора прерывания в многопроцессорной системе	0x0002 0x0014
MSGRx	Регистры обмена сообщениями в многопроцессорной системе	не определен
BMAX, BCNT	Регистры параметров доступа к общей внешней шине	0x0000 0000

Системные регистры

Команды работы с битами системных регистров (Core Processor)

BIT <operation> SysRegName Const

Операция (operation)	Назначение	Пример (нач. знач.: USTAT1 = 0xFFFF 0000)
SET	Установка битов	BIT SET USTAT1 0x0000 0200 (результат: USTAT1 = 0xFFFF 0200)
CLR	Сброс битов	BIT CLR USTAT1 0x8001 0100 (результат: USTAT1 = 0x7FFE 0000)
TGL	Инверсия битов	BIT TGL USTAT1 0x0020 0700 (результат: USTAT1 = 0xFFDF 0700)
XOR	Сравнение регистра	BIT TGL USTAT1 0xFFFF 0000 (результат: BTF = 1) BIT TGL USTAT1 0xFFFF 0001 (результат: BTF = 0)
TST	Тестирование битов	BIT SET USTAT1 0x0001 8000 (результат: BTF = 0) BIT SET USTAT1 0x0001 0000 (результат: BTF = 1)

Системные регистры

Работа с регистрами IOP-процессора

Работа с регистрами IOP-процессора выполняется как с ячейками памяти.

Невозможны прямые операции с битами регистров IOP-процессора.

Способ модификации регистра IOP-процессора:

```
Rx = dm (SYSCON) ;  
Rx = ...;          // обработка, модификация  
dm (SYSCON) = Rx;
```

Программный секвенсор ADSP-21060

1. Функции секвенсора
2. Архитектура секвенсора. Этапы выполнения инструкции
3. Переходы, вызовы подпрограмм, возвраты
4. Организация циклов
 1. Стеки, используемые при организации циклов
 2. Циклы по счетчику
 3. Циклы по арифметическому условию
 4. Ограничения при организации циклов
 5. Флаги состояния секвенсора
5. Кэш инструкций
 1. Архитектура кэша
 2. Пример влияния размещения сегментов в памяти на эффективность выполнения программы

"Работа выполнена в рамках реализации "Программы развития ФГОУ ВПО "Южный федеральный университет" на 2007-2010гг. ("Разработка образовательных контентов и ресурсов нового поколения").

Хусаинов Н.Ш.

Технологический институт Южного федерального университета в г.Таганроге, 2007г.

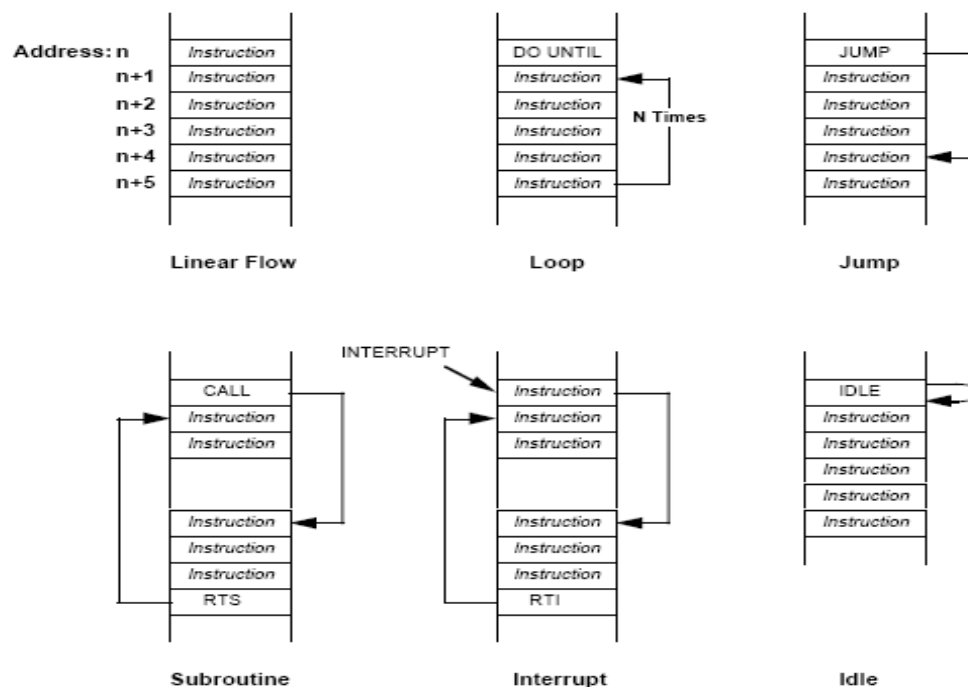
Функции секвенсора

Выполнение инструкций в линейном порядке с инкрементом адреса выборки следующей команды.

Варианты нарушения линейного порядка следования инструкций:


- переходы (условные и безусловные);
- вызовы подпрограмм и возвраты (условные и безусловные);
- вызовы обработчиков прерываний и возвраты (условные и безусловные);

- циклы;
- инструкция Idle.



Архитектура секвенсора.

Этапы выполнения инструкции



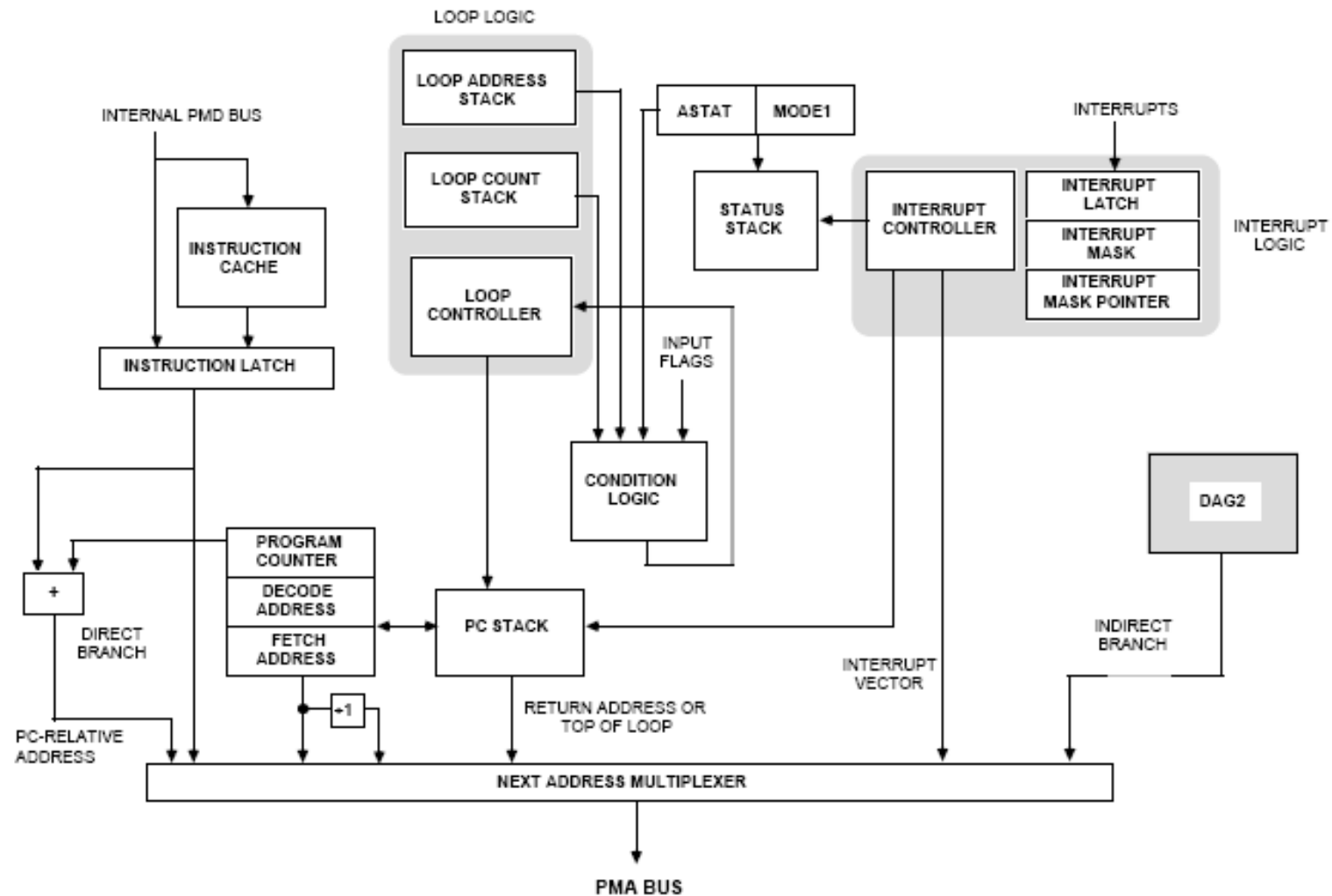
time (cycles)	Fetch	Decode	Execute
1	0x08		
2	0x09	0x08	
3	0x0A	0x09	0x08
4	0x0B	0x0A	0x09
5	0x0C	0x0B	0x0A

Адрес выполняемой (Execute) инструкции – регистр PC.

Адрес декодируемой (Decode) инструкции – регистр DADDR.

Адрес выбираемой (Fetch) инструкции – регистр FADDR.

Архитектура секвенсора. Этапы выполнения инструкции



Архитектура секвенсора.

Этапы выполнения инструкции

Регистры программного секвенсора

Регистр	Содержимое	Разрядность	Прим.
FADDR	Адрес выбираемой из памяти инструкции	24	Только чтение
DADDR	Адрес декодируемой инструкции	24	Только чтение
PC	Адрес выполняемой инструкции	24	Только чтение
PCSTK	Содержимое верхней ячейки стека PC	24	
PCSTKP	Указатель стека PC	5	
LADDR	Содержимое верхней ячейки стека цикла	32	
CURLCNTR	Содержимое верхней ячейки стека счетчика цикла	32	
LCNTR	Число итераций следующего цикла	32	

Переходы и вызовы подпрограмм

Переходы, вызовы подпрограмм, возвраты

При выполнении переходов, вызовов и возвратов выборка очередной инструкции выполняется по адресу, который не является следующим по счету за адресом предыдущей выборки.

При вызове подпрограммы дополнительно в стек Программного секвенсора (PC Stack) помещается адрес возврата.

При возврате из подпрограммы или из обработчика прерывания адрес возврата выталкивается из стека Программного секвенсора (при возврате из обработчика прерывания выполняются и дополнительные действия).

Переходы и вызовы подпрограмм

Варианты выполнения

Переходы, вызовы и возвраты могут быть условными и безусловными:

```
IF cond jump Proc2;           jump Proc2;
```

При переходах и вызовах режим адресации может быть прямой, косвенный, относительный:

```
call Proc3;  
IF cond call (M12, I14);  
jump (PC, 0x0004);
```

Переходы, вызовы и возвраты могут быть обычные и задержанные:

```
call Proc4;                   rts (db);
```

Переход может прерывать выполнение цикла.

```
jump Proc5 (LA);
```

Переходы и вызовы подпрограмм

Обычный («незадержанный») переход (вызов, возврат)

NON-DELAYED JUMP OR CALL

CLOCK CYCLES →

Execute Instruction	n	nop	nop	j
Decode Instruction	n+1->nop	n+2->nop	j	j+1
Fetch Instruction	n+2	j	j+1	j+2

n+1 suppressed

n+2 suppressed; for call, n+1 pushed on PC stack

n = Branch instruction

j = Instruction at Jump or Call address

r = Instruction at Return address

NON-DELAYED RETURN

CLOCK CYCLES →

Execute Instruction	n	nop	nop	r
Decode Instruction	n+1->nop	n+2->nop	r	r+1
Fetch Instruction	n+2	r	r+1	r+2

n+1 suppressed

n+2 suppressed; r popped from PC stack

Недостаток – необходимость перезагрузки конвейера и потеря двух тактов.

Преимущество – понятность: порядок выполнения соответствует порядку записи.

Переходы и вызовы подпрограмм

Задержанный переход (вызов, возврат)

DELAYED JUMP OR CALL

CLOCK CYCLES →

Execute Instruction	n	n+1	n+2	j
Decode Instruction	n+1	n+2	j	j+1
Fetch Instruction	n+2	j	j+1	j+2

for call, n+3
pushed on PC
stack

n = Branch instruction

j = Instruction at Jump or Call address

r = Instruction at Return address

DELAYED RETURN

CLOCK CYCLES →

Execute Instruction	n	n+1	n+2	r
Decode Instruction	n+1	n+2	r	r+1
Fetch Instruction	n+2	r	r+1	r+2

r popped from
PC stack

Преимущество – отсутствует необходимость перезагрузки конвейера. Недостаток – усложнение программы.

Ограничение: в 2-х адресах за инструкцией задержанного перехода не могут находиться инструкции других переходов (вызовов, возвратов), команды работы со стеком Программного секвенсора (PC Stack), инструкции организации циклов (DO... UNTIL...), команда IDLE.

Переходы и вызовы подпрограмм

Мнемоники условных инструкций

<u>No.</u>	<u>Mnemonic</u>	<u>Description</u>	<u>True If</u>
0	EQ	ALU equal zero	AZ = 1
1	LT	ALU less than zero	See Note 1 below
2	LE	ALU less than or equal zero	See Note 2 below
3	AC	ALU carry	AC = 1
4	AV	ALU overflow	AV = 1
5	MV	Multiplier overflow	MV = 1
6	MS	Multiplier sign	MN = 1
7	SV	Shifter overflow	SV = 1
8	SZ	Shifter zero	SZ = 1
9	FLAG0_IN	Flag 0 input	FI0 = 1
10	FLAG1_IN	Flag 1 input	FI1 = 1
11	FLAG2_IN	Flag 2 input	FI2 = 1
12	FLAG3_IN	Flag 3 input	FI3 = 1
13	TF	Bit test flag	BTF = 1
14	BM	Bus Master	
15	LCE	Loop counter expired (DO UNTIL term)	CURLCNTR = 1
15	NOT LCE	Loop counter not expired (IF cond)	CURLCNTR ≠ 1
<i>Bits 16-30 are the complements of bits 0-14</i>			
16	NE	ALU not equal to zero	AZ = 0
17	GE	ALU greater than or equal zero	See Note 3 below
18	GT	ALU greater than zero	See Note 4 below
19	NOT AC	Not ALU carry	AC = 0
20	NOT AV	Not ALU overflow	AV = 0
21	NOT MV	Not multiplier overflow	MV = 0
22	NOT MS	Not multiplier sign	MN = 0
23	NOT SV	Not shifter overflow	SV = 0
24	NOT SZ	Not shifter zero	SZ = 0
25	NOT FLAG0_IN	Not Flag 0 input	FI0 = 0
26	NOT FLAG1_IN	Not Flag 1 input	FI1 = 0
27	NOT FLAG2_IN	Not Flag 2 input	FI2 = 0
28	NOT FLAG3_IN	Not Flag 3 input	FI3 = 0
29	NOT TF	Not bit test flag	BTF = 0
30	NBM	Not Bus Master	
31	FOREVER	Always False (DO UNTIL)	always
31	TRUE	Always True (IF)	always

Переходы и вызовы подпрограмм

Примеры использования условных инструкций

Условный переход (вызов, возврат):

```
IF EQ jump Proc2;  
IF NE call Proc2 (db);  
IF NOT SZ rts;
```

Условное выполнение :

```
IF LE R1=R1+R4;  
IF GT F4=F8*F9, R2=dm(I7,m0);  
IF TF call (M8,I15), ELSE R2=R2+R8;
```

Всегда истинное условие:

```
IF TRUE call Proc4;
```

Бесконечный цикл (всегда ложное условие):

```
DO Label UNTIL FOREVER;
```

Организация циклов

Общие сведения

Процессор SHARC ADSP имеет механизм аппаратной поддержки циклов.

Проверка условия выхода из цикла проверяется за 3 такта до конца тела цикла, что позволяет избежать неверного выбора Fetch-адреса и дополнительных циклов для перезагрузки конвейера (для циклов длиной из 3-х и более инструкций).

Использование специализированных стеков Программного секвенсора позволяет использовать аппаратную поддержку циклов до 6 уровней вложенности.

Есть ограничения на использование команд переходов, вызовов и возвратов в конце тела цикла.

Особого внимания требует организация коротких циклов с выходом по арифметическому условию.

Стеки, используемые при организации циклов

Занесение значений в вершины стеков выполняется инструкцией

```
DO LabelEndLoop UNTIL cond;
```

```
...
```

```
...
```

```
...
```

```
LabelEndLoop: ...
```

Выталкивание значений из вершин стеков выполняется автоматически при проверке условия выхода (если оно истинно) или при выполнении инструкции перехода с прерыванием цикла

```
jump LabelOutOfLoop (LA);
```

Стек	Наименование	Назначение (при организации циклов)
PC Stack	Стек Программного секвенсора	Адрес первой инструкции тела цикла
Loop Address Stack	Стек адреса цикла	Адрес последней инструкции цикла, код завершения, тип цикла
Loop Counter Stack	Стек счетчика цикла	Количество итераций до конца цикла

Стеки, используемые при организации циклов

Стек Программного секвенсора (PC Stack)

Глубина – 30 слов.

Разрядность – 24 бита.

Регистр PCSTK – содержит верхушку стека.

Указатель PCSTKP – содержит количество занятых ячеек в PC Stack.

PCSTKP=0, если стек пуст.

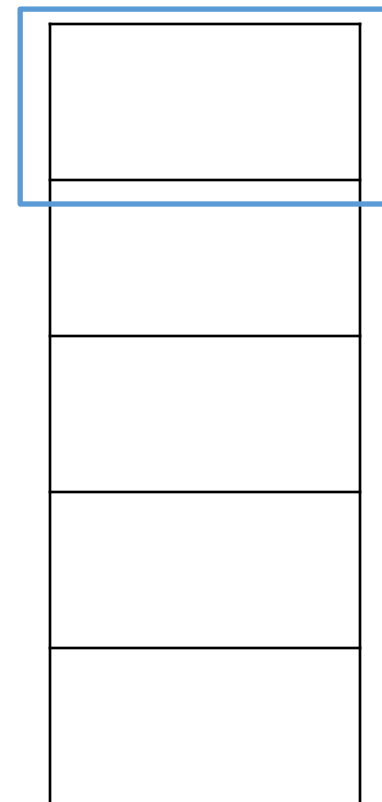
Принудительные запись/выталкивание

`PUSH PCSTK;`

`POP PCSTK;`

`PCSTK = 0xUUUU UUUU;`

PCSTK



Стеки, используемые при организации циклов

Стек адреса цикла (Loop Address Stack)

Глубина – 6 слов.

Разрядность – 32 бита.

Регистр LADDR – содержит верхушку стека.

Если LADDR=0xFFFF FFFF, то стек адреса цикла пуст.

LADDR

31 30	28 24	23 0
Код типа цикла:	Код завершения	Адрес последней инструкции тела цикла
«00»-арифм. «01» – по счетчику, 1 инструкция «02» – по счетчику, 2 инструкции «03» - по счетчику, 3 и более инструкций	Соответствует условию после UNTIL	

Стеки, используемые при организации циклов

Стек счетчика цикла (Loop Counter Stack)

Глубина – 6 слов.

Разрядность – 32 бита.

Регистр CURLCNTR – содержит верхушку стека.

Если CURLCNTR=0xFFFF FFFF, то стек счетчика цикла пуст.

Регистр LCNTR находится «над вершиной» стека счетчика цикла.

При выполнении команды

Do...UNTIL его значение

проталкивается в CURLCNTR

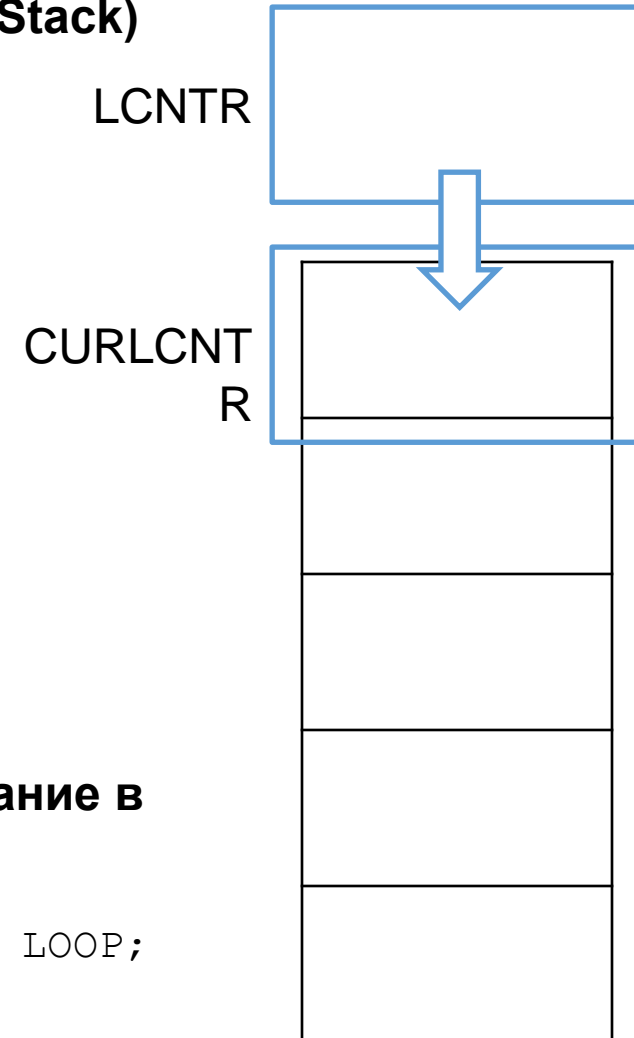
Принудительные запись/выталкивание в стеки адреса и счетчика циклов

```
LCNTR = value;
```

```
LADDR = struct;
```

```
PUSH LOOP;
```

```
POP LOOP;
```



Циклы по счетчику

Цикл по счетчику длиной 3 и более инструкций

```

0x100:      ...
0x101:      LCNTR = 2, DO EndLoop UNTIL LCE;
0x102:      R1 = R1 + 1;
0x103:      R2 = R2 - 1;
0x104:      DM(I3,M8) = R1;
0x105: EndLoop:  DM(I4,M1) = R2;
0x106:      ...
    
```

Проверка условия
выхода из цикла

PC	0x101	0x102	0x103	0x104	0x105	0x102	0x103	0x104	0x105	0x106
DADDR	0x102	0x103	0x104	0x105	0x102	0x103	0x104	0x105	0x106	0x107
FADDR	0x103	0x104	0x105	0x102	0x103	0x104	0x105	0x106	0x107	0x108
	->PCSTK ->LAS ->SCS		CURLCNTR = 1 ? CURLCNTR--	Нет			CURLCNT = 1 ? CURLCNTR— PCSTK-> LAS-> SCS->	Да		

Циклы по счетчику

Цикл по счетчику длиной в 1 инструкцию из 3-х и более итераций

```

0x100:          ...
0x101:          LCNTR = 5, DO EndLoop UNTIL LCE;
0x102: EndLoop:      DM(I4,M1) = R2;
0x103:          ...
    
```

Проверка условия
выхода из цикла

PC	0x101	0x102	0x102	0x102	0x102	0x102	0x103
DADDR	0x102	0x102	0x102	0x102	0x102	0x103	0x104
FADDR	0x103	0x102	0x102	0x102	0x103	0x104	0x105
	->PCSTK ->LAS ->SCS	CURLCNTR = 3 ? CU RLCNTR-- (=4)	CURLCNTR = 3 ? CURLCNTR-- (=3)	CURLCNT = 3 ? CURLCNTR=0xffffffff PCSTK->; LAS->; SCS->			

Циклы по счетчику

Цикл по счетчику длиной в 1 инструкцию из 1-й итерации

0x101: LCNTR = 1, DO EndLoop UNTIL LCE;

PC	0x101	0x102	nop	nop	0x103
DADDR	0x102	0x102	0x102	0x103	0x104
FADDR	0x103	0x102	0x103	0x104	0x105
	->PCSTK ->LAS ->SCS	CURLCNTR = 1 ? CU RLCNTR=0xffffffff PCSTK->; LAS->; SCS->			

Цикл по счетчику длиной в 1 инструкцию из 2-х итераций

0x101: LCNTR = 2, DO EndLoop UNTIL LCE;

PC	0x101	0x102	0x102	nop	nop	0x103
DADDR	0x102	0x102	0x102	0x102	0x103	0x104
FADDR	0x103	0x102	0x102	0x103	0x104	0x105
	->PCSTK ->LAS ->SCS	CURLCNTR = 1 ? CU RLCNTR-- (=1)	CURLCNTR = 1 ? CU RLCNTR=0xffffffff PCSTK->; LAS->; SCS->			

Цикл по счетчику из одной инструкции должен выполняться не менее 3-х раз чтобы избежать потерь производительности

Циклы по счетчику

Цикл по счетчику длиной в 2 инструкцию из 2-х и более итераций

```
0x100:          ...
0x101:          LCNTR = 3, DO EndLoop UNTIL LCE;
0x102:          DM(I4,M1) = R1;
0x103: EndLoop:  DM(I4,M1) = R2;
0x104:          ...
```

Проверка условия
выхода из цикла

PC	0x101	0x102	0x103	0x102	0x103	0x102	0x103	0x104
DADDR	0x102	0x103	0x102	0x103	0x102	0x103	0x104	0x105
FADDR	0x103	0x102	0x103	0x102	0x103	0x104	0x105	0x106
	->PCSTK ->LAS ->SCS		CURLCNTR = 2 ? CURLCNTR- (=2)		CURLCNT = 2 ? CURLCNTR = 0xFFFFFFFF PCSTK->; LAS->; SCS->			

Циклы по счетчику

Цикл по счетчику длиной в 2 инструкции из 1-й итерации

```
0x100:      ...
0x101:      LCNTR = 1, DO EndLoop UNTIL LCE;
0x102:      DM(I4,M1) = R1;
0x103: EndLoop:      DM(I4,M1) = R2;
0x104:      ...
```

PC	0x101	0x102	0x103	nop	nop	0x104
DADDR	0x102	0x103	0x102	0x103	0x104	0x105
FADDR	0x103	0x102	0x103	0x104	0x105	0x106
	->PCSTK ->LAS ->SCS		CURLCNTR = 1 ? CU RLCNTR=0xffffffff PCSTK->; LAS->; SCS->			

Цикл по счетчику из двух инструкций должен выполняться не менее 2-х раз чтобы избежать потерь производительности

Циклы по счетчику

Примеры неоптимальной и оптимальной организации цикла по счетчику

Задание.

Даны массивы A и B одинаковой размерности (N элементов).

Вычислить:

Неоптимальная организация цикла - 1

```
I0 = A;  M0 = 1;  I8 = B;  M8 = 1;  
...  
MRF = 0;  
R1 = 0;  
R2 = 0;  
LCNTR = N, DO LabelEndLoop UNTIL LCE;  
                R1 = DM(I0,M0), R2 = PM(I8,M8);  
LabelEndLoop:   MRF = MRF + R1*R2 (SSI);  
                R0 = MRF;
```

Количество тактов: $3*1 + 1*1 + (2*2 + (N-2)*1 + N*1) + 1*1 = 2*N+7$

Выполняется «лишнее» чтение из памяти с дополнительным сдвигом указателей

Циклы по счетчику

Примеры неоптимальной и оптимальной организации цикла по счетчику

Неоптимальная организация цикла - 2

```
I0 = A; M0 = 1; I8 = B; M8 = 1;  
...  
MRF = 0;  
R1 = 0;  
R2 = 0;  
LCNTR = N+1, DO LabelEndLoop UNTIL LCE;  
LabelEndLoop: MRF = MRF + R1*R2 (SSI), R1 = DM(I0,M0), R2 = PM(I8,M8);  
R0 = MROF;
```

Количество тактов: $3*1 + 1*1 + (3*2 + (N-2)*1) + 1*1 = N+9$

Выполняется «лишнее» чтение из памяти с дополнительным сдвигом указателей

Циклы по счетчику

Примеры неоптимальной и оптимальной организации цикла по счетчику

Оптимальная организация цикла

```
I0 = A; M0 = 1; I8 = B; M8 = 1;  
...  
MRF = 0, R1 = DM(I0,M0), R2 = DM(I8,M8);  
LCNTR = N-1, DO LabelEndLoop UNTIL LCE;  
LabelEndLoop:    MRF = MRF + R1*R2 (SSI), R1 = DM(I0,M0), R2 = PM(I8,M8);  
                  R0 = MRF + R1 * R2 (SSI);
```

Количество тактов: $1*2 + 1*1 + (2*2 + (N-3)*1) + 1*1 = N+5$

«Лишнее» чтение из памяти с дополнительным сдвигом указателей не выполняется

Циклы по счетчику

Примеры неоптимальной и оптимальной организации цикла по счетчику
Задание.

Даны массивы вещественных чисел A и B одинаковой размерности (N элементов).

Вычислить:

Неоптимальная организация цикла

```
I0 = A; M0 = 1; I8 = B; M8 = 1;
```

```
...
```

```
F8 = 0;
```

```
LCNTR = N, DO LabelEndLoop UNTIL LCE;
```

```
    R1 = DM(I0,M0), R2 = PM(I8,M8);
```

```
    F3 = F1 * F2;
```

```
LabelEndLoop:    F8 = F8 + F3;
```

```
    DM(...) = R8;
```

Количество тактов: $1*1 + 1*1 + (1*2 + (N-1)*1 + 2*N*1) + 1*1 = 3*N+4$

Циклы по счетчику

Примеры неоптимальной и оптимальной организации цикла по счетчику

Оптимальная организация цикла

```
I0 = A;  M0 = 1;  I8 = B;  M8 = 1;
```

```
...
```

```
R1 = DM(I0,M0), R4 = PM(I8,M8);
```

```
F8 = F1*F4, R1 = DM(I0,M0), R4 = PM(I8,M8);
```

```
F12 = F1*F4, R1 = DM(I0,M0), R4 = PM(I8,M8);
```

```
LCNTR = N-3, DO LabelEndLoop UNTIL LCE;
```

```
LabelEndLoop:  F12 = F1*F4, F8 = F8+F12, R1 = DM(I0,M0), R4 = PM(I8,M8);
```

```
F12 = F1*F4, F8 = F8+F12;
```

```
F8 = F8 + F12;
```

```
DM(...) = R8;
```

Количество тактов: $1*2 + 1*2 + 1*2 + 1*1 + (3*2 + (N-6)*1) + 1*1 + 1*1 + 1*1 = N+10$

Циклы по арифметическому условию

Цикл по арифм. условию длиной 3 и более инструкций

```
0x100:          R1 = 2;
0x101:          DO EndLoop UNTIL EQ;
0x102:          R1 = R1 - 1;
0x103:          R2 = R2 + R6;          Проверка условия
                                         выхода из цикла
0x104:          DM(I3,M8) = R1;
0x105: EndLoop:  DM(I4,M1) = R2;
0x106:          ...
```

PC	0x101	0x102	0x103	0x104	0x105	0x102	0x103	0x104	0x105	0x106
DADDR	0x102	0x103	0x104	0x105	0x102	0x103	0x104	0x105	0x106	0x107
FADDR	0x103	0x104	0x105	0x102	0x103	0x104	0x105	0x106	0x107	0x108
	->PCSTK ->LAS ->SCS		AZ ?	Нет			AZ ? PCSTK->; LAS->; SCS->	Да		

Циклы по арифметическому условию

Сложности в организации короткого (1-2 инструкции) цикла по арифметическому условию - 1

```
0x100:          R1 = 3;  
0x101:          DO EndLoop UNTIL EQ;  
0x102: EndLoop:  R1 = R1 - 1;  
0x103:          ...
```

Проверка условия
выхода из цикла

PC	0x101	0x102	0x102	0x102	0x102	0x102	0x102	0x103
DADDR	0x102	0x102	0x102	0x102	0x102	0x102	0x103	0x104
FADDR	0x103	0x102	0x102	0x102	0x102	0x103	0x104	0x105
	->PCSTK ->LAS ->SCS	AZ ? (R1=2)	AZ ? (R1=1)	AZ ? (R1=0)	AZ ! (R1=-1) PCSTK->; LAS->; SCS->	(R1=-2)	(R1=-3)	

Условие выхода из цикла проверяется в начале процессорного такта.

Значение флага AZ устанавливается по результатам выполнения операции АЛУ в конце процессорного такта.

Выполняются дополнительные итерации цикла.

Циклы по арифметическому условию

Сложности в организации короткого (1-2 инструкции) цикла по арифметическому условию - 2

```
0x100:          R1 = 2;
0x101:          DO EndLoop UNTIL EQ;
0x102:          R5 = DM(I1,M2);
0x103: EndLoop: R1 = R1 - 1;
0x104:          ...
```

Проверка условия
выхода из цикла

PC	0x101	0x102	0x103	0x102	0x103	0x102	0x103	0x102	0x103	0x104
DADDR	0x102	0x103	0x102	0x103	0x102	0x102	0x102	0x103	0x104	0x105
FADDR	0x103	0x102	0x103	0x102	0x103	0x103	0x103	0x104	0x105	0x106
	->PCSTK ->LAS ->SCS		AZ ? (R1=1)	AZ-const	AZ ? (R1=0)	AZ-const	AZ ? (R1=-1) PCSTK->; LAS->; SCS->		(R1=-2)	

Условие выхода из цикла проверяется в начале процессорного такта.

Значение флага AZ устанавливается по результатам выполнения операции АЛУ в конце процессорного такта.

Выполняются дополнительные итерации цикла.

Циклы по арифметическому условию

Сложности в организации короткого (1-2 инструкции) цикла по арифметическому условию - 3

0x100: R1 = 2; R5 = 6;

0x101: DO EndLoop UNTIL EQ;

0x102: R5 = R5 + 1;

0x103: EndLoop: R1 = R1 - 1;

0x104: ...

Проверка условия
выхода из цикла

PC	0x101	0x102	0x103	0x102	0x103	0x102	0x103	0x102	0x103	0x102
DADDR	0x102	0x103	0x102	0x103	0x102	0x102	0x102	0x103	0x102	0x103
FADDR	0x103	0x102	0x103	0x102	0x103	0x103	0x103	0x102	0x103	0x102
	->PCSTK ->LAS ->SCS		AZ ? (R1=1) AZ = 0	AZ = 0	AZ ? (R1=0) AZ = 1	AZ = 0	AZ ? (R1=-1) AZ = 0	AZ = 0	AZ ? (R1=-2) AZ = 0	AZ = 0

Значение флага AZ, проверяемого в начале цикла выполнения инструкции 0x103, изменяется инструкцией 0x102.

Возможно получение бесконечного цикла.

Ограничения при организации циклов

Вложенные циклы не могут заканчиваться на одной и той же инструкции.

Для предотвращения потерь производительности циклы по счетчику из одной инструкции должны иметь не менее 3-х итераций, а из двух инструкций – не менее 2-х итераций.

«Короткие» циклы не должны содержать никакие команды перехода, вызова и возврата.

Последние 3 инструкции любого цикла не должны содержать никакие команды перехода, вызова и возврата. В противном случае цикл может быть отработан некорректно. Единственное исключение – «незадержанный» вызов подпрограммы CALL и возврат RTS с модификатором (LR).

Флаги состояния секвенсора

Флаги состояния стеков Программного секвенсора в регистре STKY

Флаг	Описание	Когда устанавливается	Прим.
PCFL	Стек PC Stack полон	Когда в стек записывается слово № 29 (из 30-х возможных)	не липкий
PCEM	Стек PC Stack пуст		не липкий
SSOV	Переполнение стека статуса	Когда заносится слово в полный стек	липкий
SSEM	Стек статуса пуст		не липкий
LSOV	Переполнение стека адреса и счетчика цикла	Когда заносится слово в полный стек	липкий
LSEM	Стеки адреса и счетчика цикла пусты		не липкий

Кэш инструкций

Принцип работы кэша

Кэш инструкций – ассоциативный кэш с местом для хранения 32-х инструкций, «прозрачный» для программиста.

Процессор кэширует только те инструкции, выборка которых из РМ-памяти конфликтует с обращением к данным в РМ:

```
PC =>          0x101:  R1 = R2 + R3,   R2 = PM(I8,M9) ;  
Decode =>      0x102:  R4 = 0 ;  
Fetch =>      0x103:  R5 = R2*R4 ;
```

При выборке инструкции процессор всегда сначала обращается за инструкцией в кэш, а при кэш-промахе – в РМ-память, вызывая дополнительный цикл на выборку инструкции.

Архитектура кэша

	LRU Bit	Instruction	Address	Valid Bit
Set 0	<input type="checkbox"/>			
Set 1	<input type="checkbox"/>			
Set 2	<input type="checkbox"/>			
•	•	•	•	•
•	•	•	•	•
Set 13	<input type="checkbox"/>			
Set 14	<input type="checkbox"/>			
Set 15	<input type="checkbox"/>			

Выбор набора осуществляется по младшим 4 битам адреса инструкции. В поле Address хранятся только старшие 20 битов адреса инструкции.

Бит LRU (Least Recently Used) – показывает более редко используемую инструкции в наборе (кандидат на замещение новой инструкцией).

Бит Valid Bit показывает наличие реальных данных в каждой строке набора.

Кэш может быть заморожен (бит CAFRZ в регистре MODE1) или выключен (бит CADIS в регистре MODE1).

Пример влияния размещения сегментов в памяти на эффективность выполнения программы

```
0x0100: LCNTR=1024, DO tight UNTIL LCE;
0x0101:          R0=DM(I0,M0), PM(I9,M9);
0x0102:          R1 = R0-R15;
0x0103:          IF EQ CALL sub;
0x0104:          F2 = FLOAT R1;
0x0105:          F3 = F2 * F2;
0x0106: tight:   F3 = F3 + F4;
...
0x0200: sub:     R1 = R13;
0x0201:          R14 = PM(I9,M9);
0x0202:          R6 = R6 + 1;
0x0203:          R7 = PASS R12;
...
0x0211:          PM(I9,M9) = R12;
0x0212:          R4 = R4-1;
0x0213:          R7 = R5+R6;
...
0x021F:          rts;
```

Три инструкции, выполняющие обращение к РМ-памяти и вызывающие конфликт доступа к памяти. Кэшируемые инструкции (0xYYY3), попадают в один набор (№ 3) кэша, что приводит к постоянному замещению самой «старой» инструкции на новую.

Как следствие – кэш неэффективен.

Решение – сдвинуть сегменты в памяти друг относительно друга, чтобы «проблемные» инструкции приходились на разные наборы кэша.