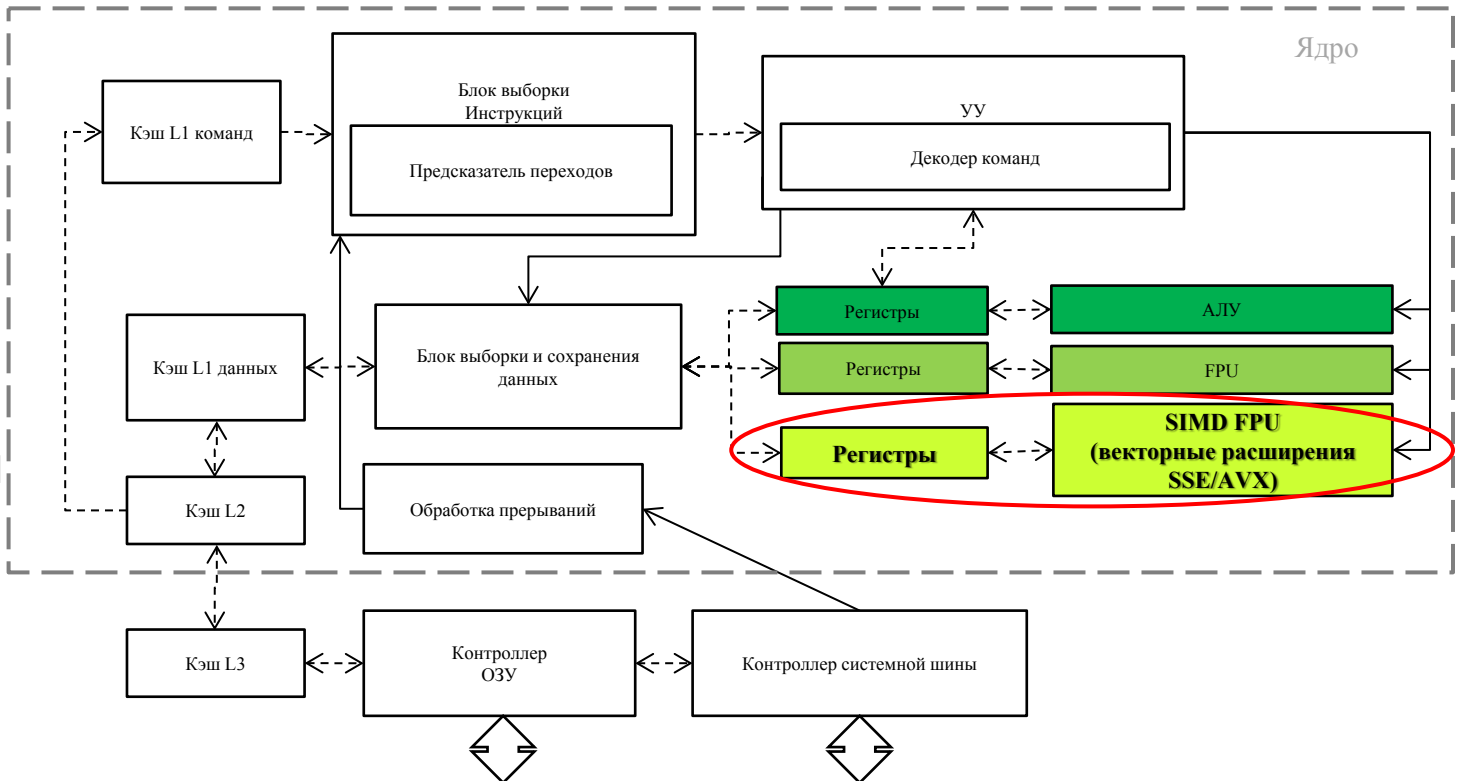
SIMD- ОУ процессоров x86

SIMD инструкции

Векторный сопроцессор SIMD FPU

SIMD FPU -

<u>Отдельное</u> операционное устройство для ***SIMD параллельных (векторных) операций*** над числами с плавающей точкой со своими РОН и системой команд



Основное применение

- Обработка медиа-данных (изображений, видео, звука, и т.п.)
  - высокая степень параллелизма
  - отсутствие ветвлений при обработке отдельных элементов
- Вычислительная геометрия
  - умножение вещественных векторов и матриц размером от 2x2 до 4x4 элементов
- Линейная алгебра

SIMD инструкции x86

- Включены в общую схему кодирования команд
- Имеют собственные регистры
- Требуют выравнивания данных в памяти
- Наборы инструкций:
  MMX, 3DNow!, SSE, SSE2, SSE3, SSSE3, SSE4, AVX, AVX-512

MMX

- Впервые: 1997, Pentium (Pentium with MMX)
- от MultiMedia eXtension
  - регистры: **mm0-mm7** (64 бит) - занимают **младшие 64 бит регистров FPU**, устарели
  - тип операндов: целые

разрядность, бит	обрабатываются параллельно, штук
8	8
16	4
32	2

3DNow!

- Впервые: 1998, AMD K6-2
  - операнды: вещественные
  - не получил распространения
  - не поддерживается Intel
  - не будет поддерживаться в будущих процессорах AMD

SSE, SSE2, SSE3, SSSE3, SSE4

- **Streaming SIMD Extensions** – потоковые SIMD расширения
- 8 новых регистров: **xmm0-xmm7** (128 бит)
- в x86-64 - еще 8 регистров **xmm8-xmm15**
- Регистр управления/статуса **MXCSR** (32 бит)

разрядность, бит	обрабатываются параллельно, штук
8	16
16	8
32	4
64	2

Версии SSE

- SSE (1999, Pentium III, AMD: Athlon XP)
  - **вещественные** числа 32 бит
  - несколько целочисленных инструкций для регистров **MMX**
- SSE2 (2001, Pentium 4, AMD: Athlon 64)
  - **вещественные** числа 64 бит
  - целые числа 8, 16, 32 бит (**инструкции MMX теперь могут работать с регистрами SSE**)
  - управление кэшем

Версии SSE

- SSE3 (2004, Pentium 4 (Prescott), Athlon 64 E)
  - горизонтальные инструкции
  - новые инструкции округления
  - загрузка вектора целых чисел без выравнивания в памяти
- SSSE3 – Supplemental SSE3 (2006, Intel Core, AMD Bobcat)
  - новые целочисленные инструкции
  - новые инструкции перестановки

Версии SSE

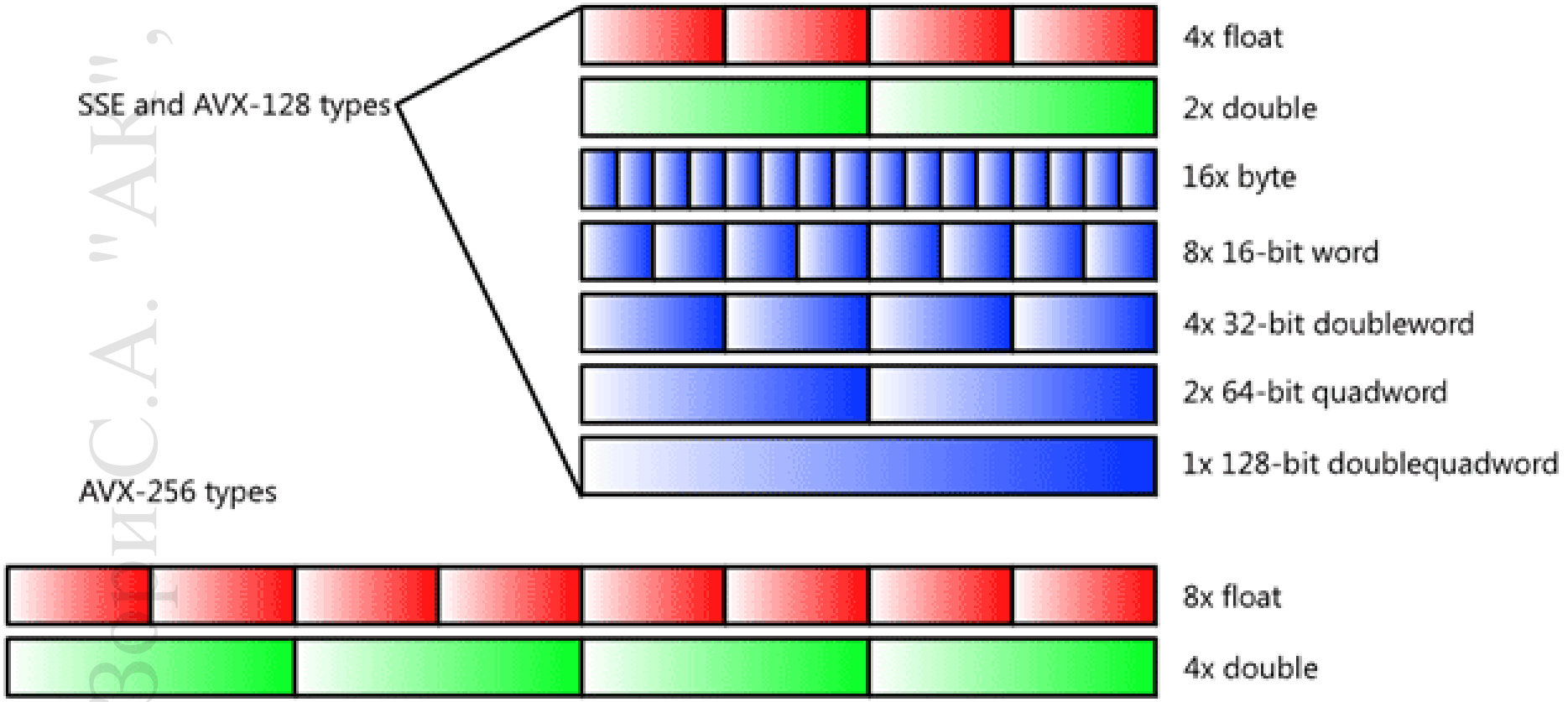
- SSE4 (2006, Intel Core, AMD K10)
  - инструкции, использующих xmm0 как неявный 3-й операнд
  - поиск и сравнение символа одновременно в двух регистрах (в сумме 256 бит)
  - подсчет числа единиц и левых нулей

AVX, AVX-512

- **AVX** (2011, Intel Sandy Bridge, AMD Bulldozer)
  - расширяет регистры **XMM** (128 бит) до регистров **YMM** (256 бит)
  - новая схема кодирования с тремя операндами
- **AVX2** (2013, Intel Haswell)
  - дополнительные инструкции
  - целочисленные инструкции расширены до 256 бит
- **AVX-512** (2015)
  - расширяет регистры **YMM** до регистров **ZMM** (512 бит)

AVX

- Типы данных AVX:



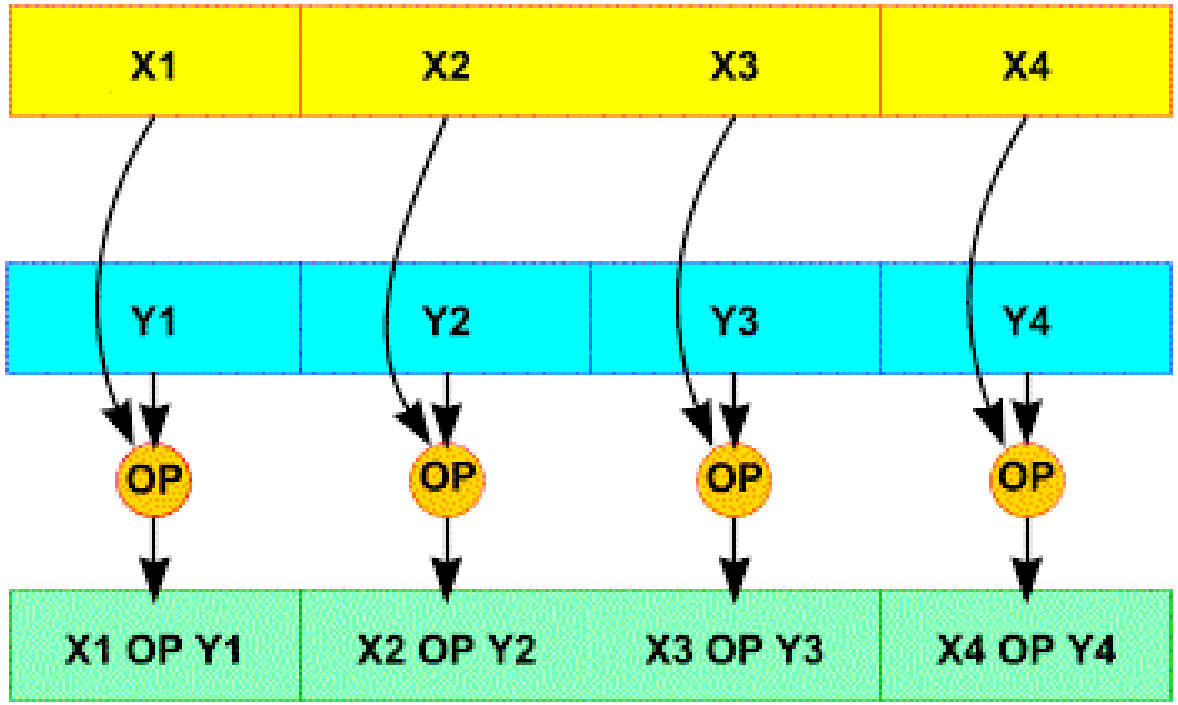
Инструкции SSE

- Последний символ кодирует разрядность:
  - **S** – Single (32 бит)
  - **D** – Double (64 бит, SSE2)

- Далее описываются инструкции для Single (для Double – аналогичны)

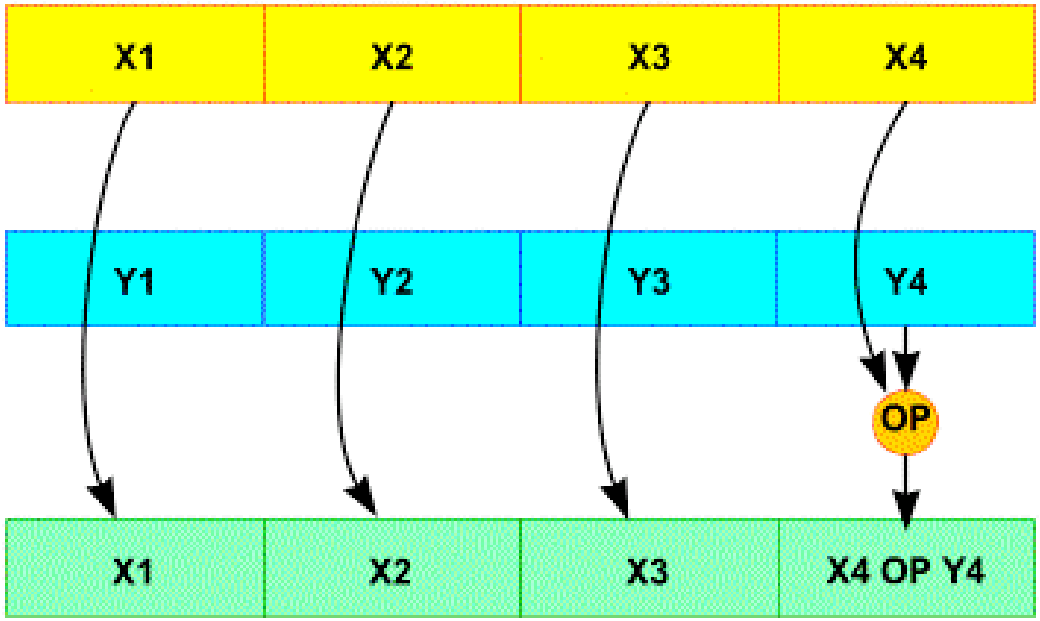
Инструкции SSE

- Векторные инструкции, имеют суффикс **PS** или **PD** (P – Packed или Parallel)



Инструкции SSE

- Скалярные инструкции, имеют суффикс **SS** или **SD** (S – Scalar)
  - обрабатывают только младшее число в регистрах XMM



Инструкции SSE

- Синтаксис инструкций (если не указано другого):

КОД**xx** dst, src

- Где **xx –** суффикс (SS, PS, SD, PD)
- Семантика:

dst := dst **операция** src

- dst – регистр **xmm**
- src – регистр **xmm** или **вектор** в памяти с **выравниванием** 128 бит

Выравнивание данных

- Выравнивание X байт означает: адрес данных в ОЗУ кратен X
- Для выравнивания данных в MASM – перед ними директива **align**

Пример:

```
align 16
my_vec dd 1.4, 2.5, …
```

Организация циклов

- При обработке массивов вещественных 32-битных чисел за 1 итерацию **обрабатывается 4 элемента**
- **Число элементов должно быть кратно 4**
- Число итераций в 4 раза меньше длины массива

Инструкции перемещения

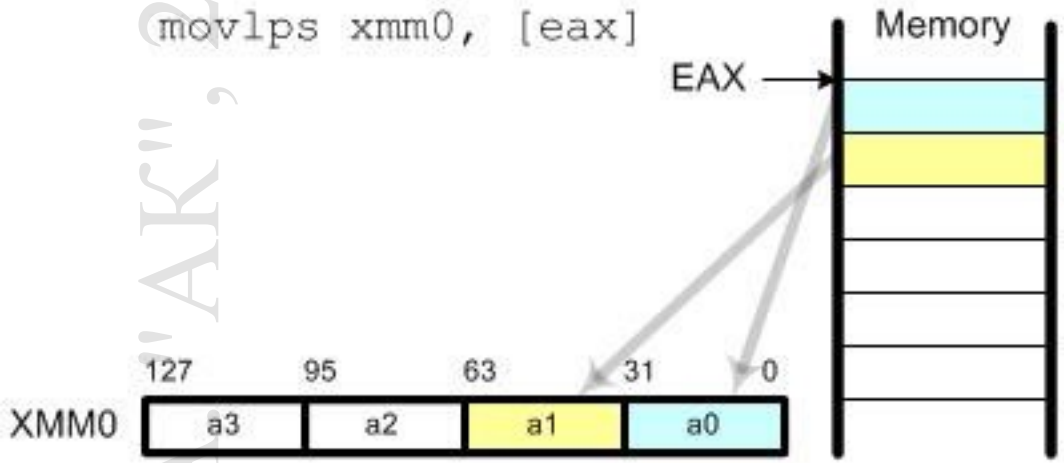
команда	направление	перемещает, бит
MOVAPS **(A - Aligned)**	пам. => xmm xmm => пам. xmm => xmm	128 *выравнивание: 128*
MOVUPS **(U - Unaligned)**	то же	128 *выравнивание: нет* медленнее, чем MOVAPS
MOVLPS **(L - Low)**	пам. => xmm xmm => пам.	64 бит из/в младшей части регистра *выравнивание: 64*
MOVHPS dst, src (H - High)	то же	то же, но для старшей части

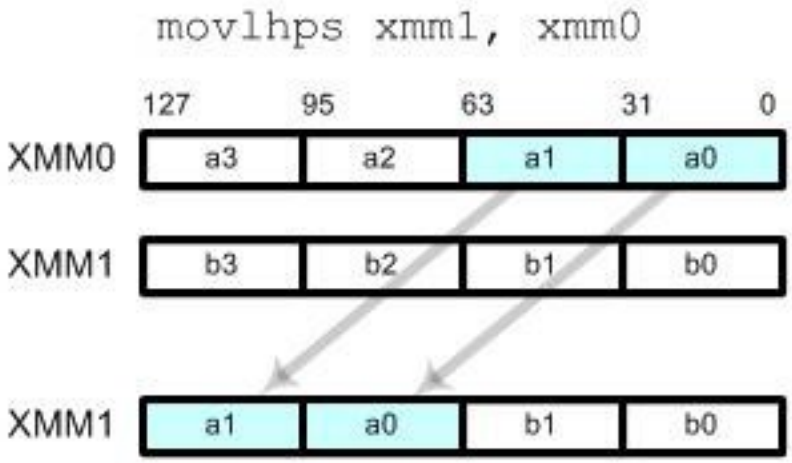
Инструкции перемещения

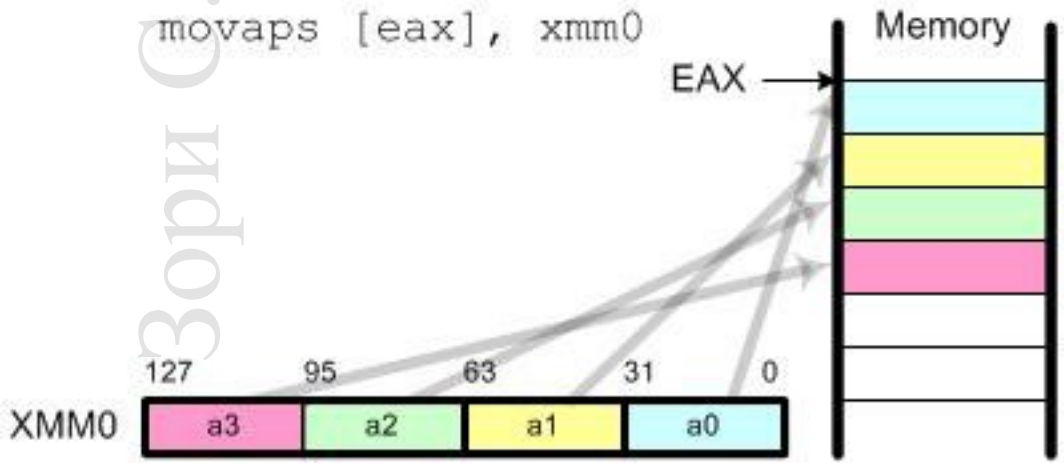
команда	направление	перемещает, бит
MOVSS	пам. => xmm xmm => пам. xmm => xmm	младшие 32 бит *выравнивание: 32*
MOVHLPS	xmm => xmm	старшие 64 бита src в младшие 64 бита dst
MOVLHPS	xmm => xmm	младшие 64 бита src в старшие 64 бита dst
MOVMSKPS **(Most Significant)**	xmm => регистр x86	старший бит из каждой каждых 32-х (всего 4 бита) *обычно используется при ветвлениях*

Инструкции перемещения

- Примеры:







Арифметические инструкции

- Основные операции: **ADDxx**, **SUBxx**, **MULxx**, **DIVxx**
- Корень: **SQRTxx**
- Минимум/максисмум **MINxx**, **MAXxx**

Обратные значения

- **RCPxx** dst, src dst := 1 / src
- **RSQRTxx** dst, src $dst := \frac{1}{\sqrt{src}}$

- Обладают меньшей точностью (12 бит мантиссы вместо 24)
- Вычисляются быстрее обычного деления

Пример

- Вычислить $r_i = \sqrt{a_i + b_i}$ для i=1…N

```
.xmm      ; включение поддержки SSE
…
.data
align 16                    ; 16 байт
a   dd 1.0, 2.0, …   ; 20 элементов
b   dd 1.1, 2.2, …
r   dd 20 dup (?)
n   dd 20             ; число, кратное 4
```

Пример

```
.code
main proc
    mov ecx, n
    shr ecx, 2      ; число итераций

    xor esi, esi    ; смещение в массиве
fori:
    movaps    xmm0, a[esi] ; xmm0 = a
    addps     xmm0, b[esi] ; xmm0 = a + b
    sqrtps    xmm0, xmm0   ; xmm0 = √(a + b)
    movaps    r[esi], xmm0 ; сохранить результат
    ;  переход к следующим элементам
    add esi, 16
    loop fori
```

Логические операции

- Выполняют операцию над всеми битами регистра, игнорируя тип содержимого
- Только параллельные команды
- Команды: **ANDPS**, **ANDNPS, ORPS**, **XORPS**

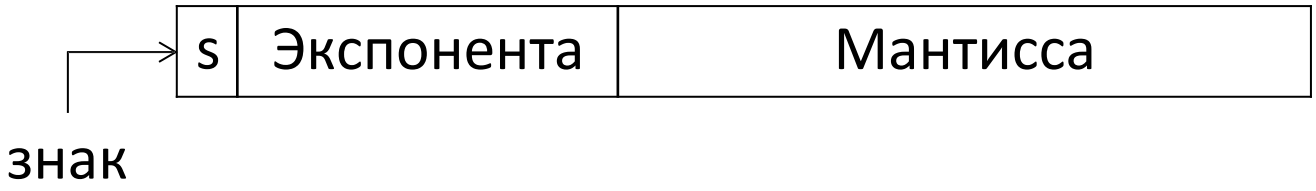
- Семантика **ANDNPS**:

$$\mathrm{dst} := \overline{\mathrm{dst}}\ \&\ \mathrm{src}$$

- NOT отсутствует. Вместо NOT используется **XORPS** с одним из операндов 111…1

Логические операции

- Нестандартное использование с вещественными числами (для 32-х бит):
- $|A|$ - **ANDPS** с 7FFFFFFFh
- $-|A|$ - **ORPS** с 80000000h
- $-A$ – **XORPS** с 80000000h



Операции сравнения

- Синтаксис: CMP**cc**__xx__ dst, src
- **cc** – код сравнения

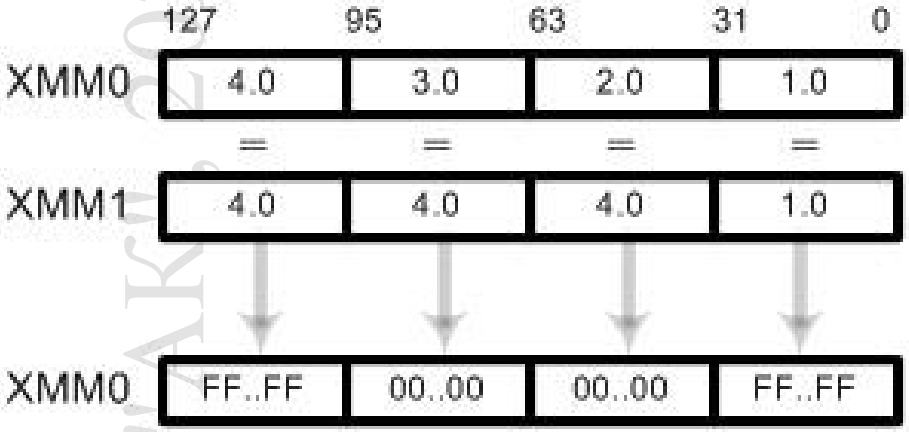
код	условие	код	условие	расшифровка
EQ	=	**NEQ**	≠	EQual
LT	<	**NLT**	≥	Less Than
LE	≤	**NLE**	>	Less Equal

- Семантика:

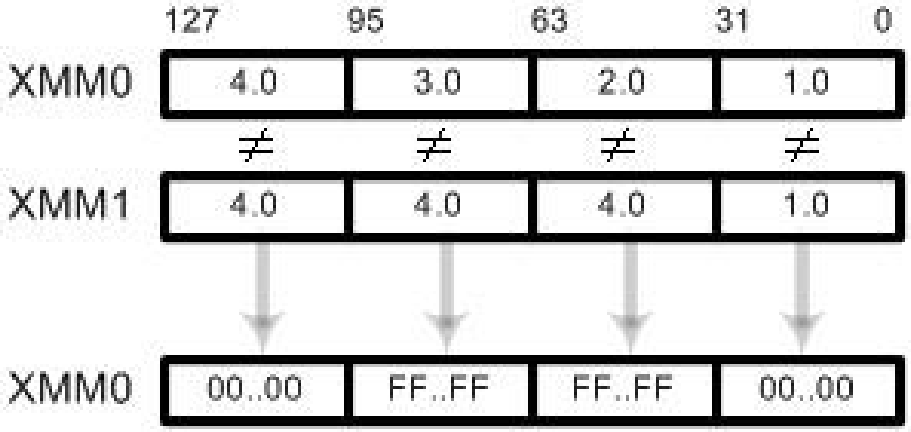
$$\text{dst} := \begin{cases} 11\ldots1, & \text{если выполняется (dst } \mathbf{cc} \text{ src)} \\ 00\ldots0, & \text{иначе} \end{cases}$$

Примеры

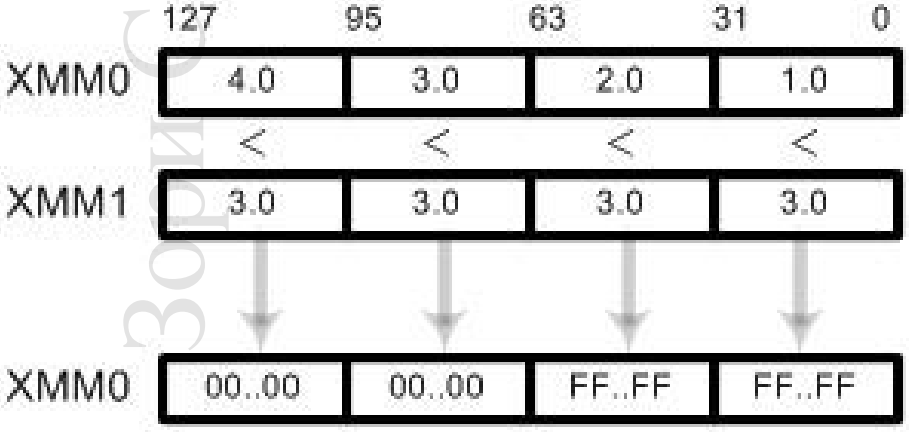
`cmpeqps xmm0, xmm1`



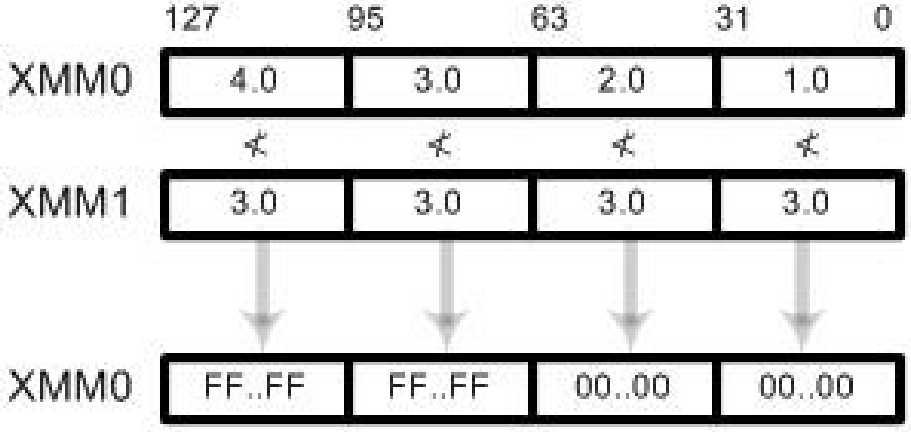
`cmpneqps xmm0, xmm1`



`cmpltps xmm0, xmm1`



`cmpnltps xmm0, xmm1`



Вычисления с условиями

- Пример: вычислить

$$r_i := \begin{cases} \dfrac{a_i}{b_i}, & \text{если } b_i \neq 0 \\ a_i + b_i, & \text{иначе} \end{cases}$$

- Проблема: ко всем элементам применяются одни и те же операции

Вычисления с условиями

- Решение:
  - разбить формулу на сумму частей
  - каждая часть соответствует одной ветви
  - вычисляются условия
  - числа, которые не должны входить в сумму, обнуляются с помощью **ANDPS**, **ANDNPS**

$$r_i := \frac{a_i}{b} \,\&\, c_i + (a_i + b_i) \,\&\, c_i$$

$$c_i := \begin{matrix} {}_i111\ldots1, & \text{если } b_i \neq 0 \\ 000\ldots0, & \text{иначе} \end{matrix}$$

Пример

- Фрагмент цикла:

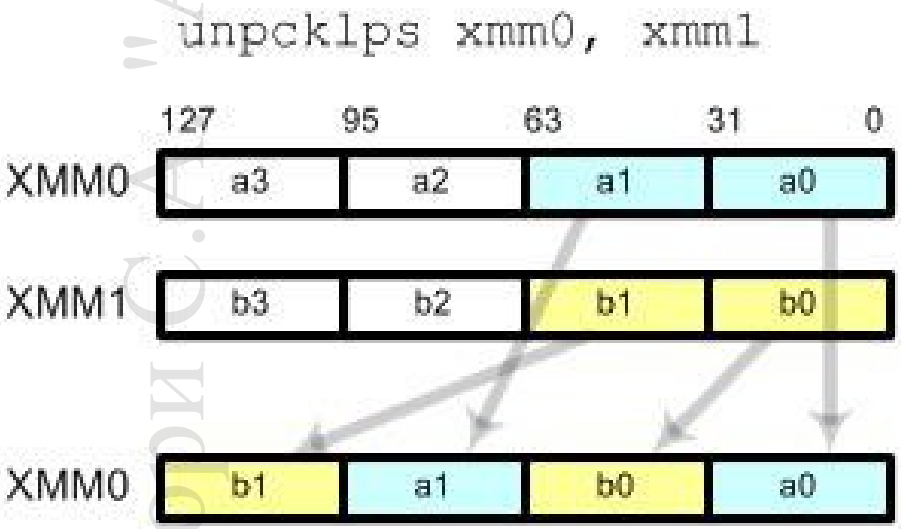
```
movaps    xmm0, a[esi] ; xmm0 = a
movaps    xmm1, b[esi] ; xmm1 = b

; условие
xorps     xmm2, xmm2   ; xmm2 = 0
cmpneqps  xmm2, xmm1   ; xmm2 = c
```

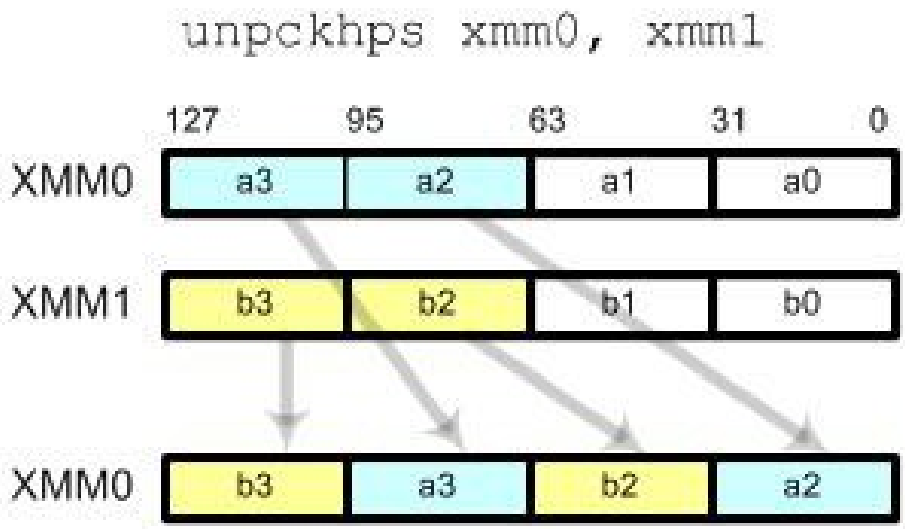
Пример


```
; части формулы
movaps      xmm3, xmm1      ; xmm3 = b
addps       xmm3, xmm0      ; xmm3 = a + b
divps       xmm0, xmm1      ; xmm0 = a / b

; обнуление
andps       xmm0, xmm2      ; xmm0 = c & (a / b)
andnps      xmm2, xmm3      ; xmm2 = ~c & (a + b)

; объединение
orps        xmm0, xmm2      ; xmm0 = r
```

Инструкции перестановки

- **UNPCKLPS** – заполняет dst чередующимися числами из младших частей src и dst
- **UNPCKHPS** – то же, но из старших частей

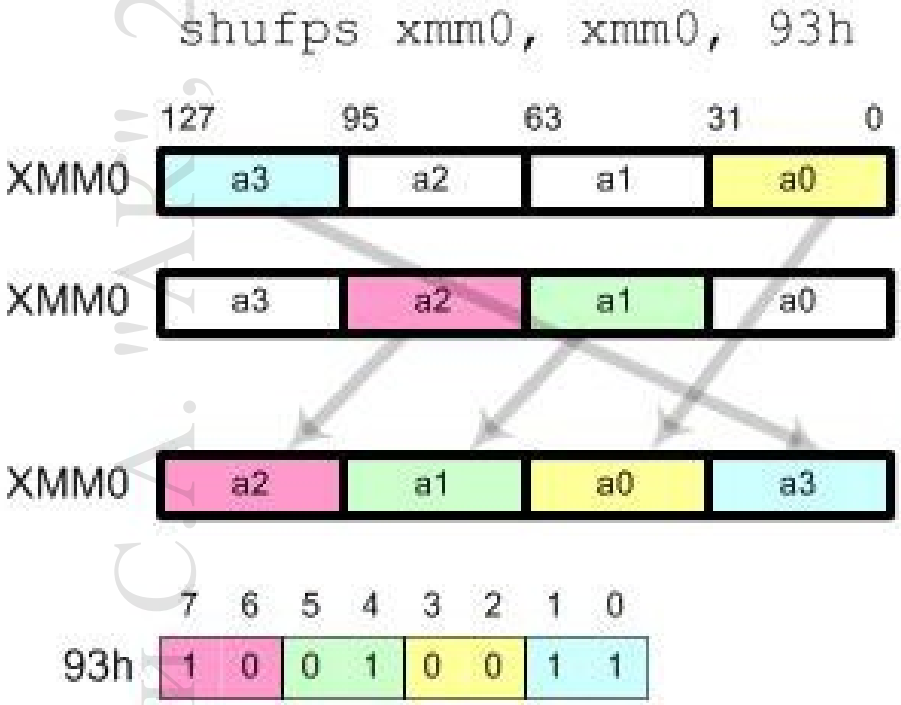


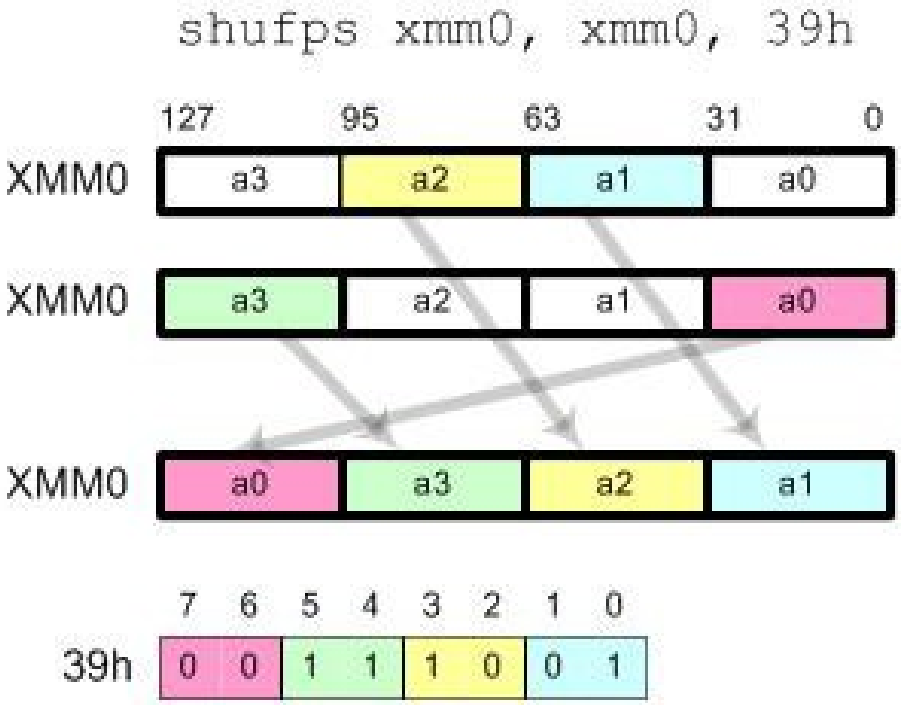


Инструкции перестановки

- **SHUFPS dst, src, code**
- Заполняет
  - младшую часть **dst** – числами из **dst**
  - старшую часть **dst** – числами из **src**
- Номера чисел в регистрах указаны в **code**
  - размер **code** – 1 байт
  - номер чисел кодируется парой бит

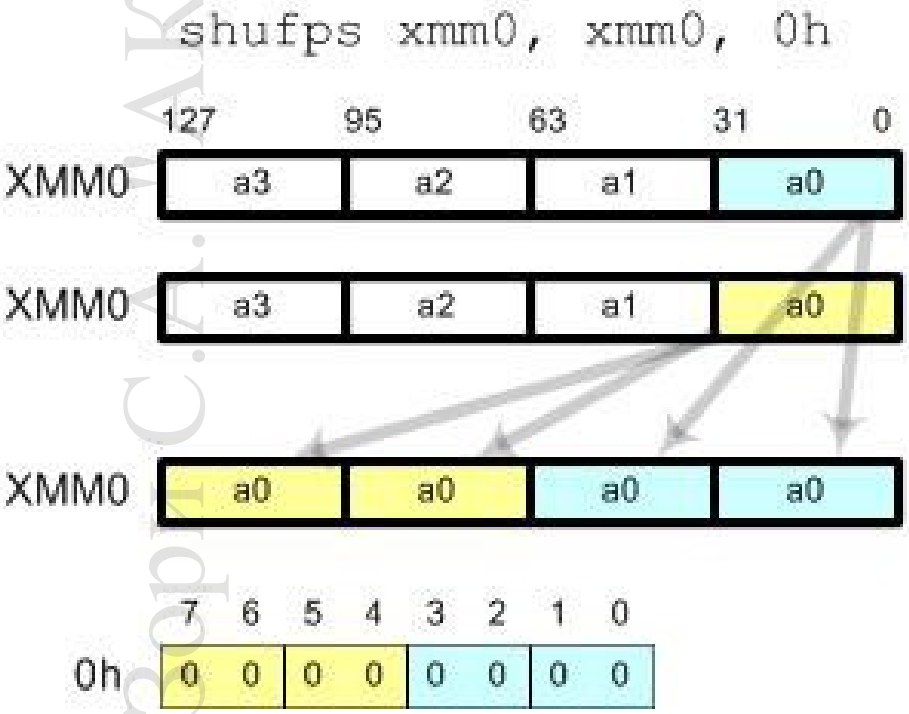
SHUFPS

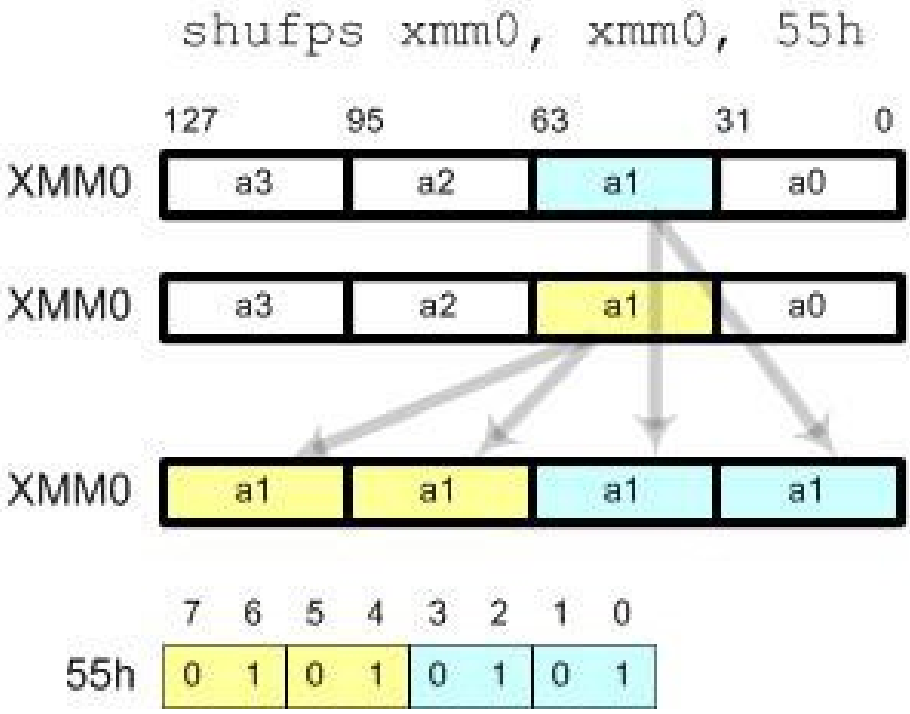




SHUFPS

- **SHUFPS** можно использовать для заполнения всего регистра одним числом





Пример

- Вычислить $a_i = i * a_i$

…

```
align 16
    a          dd 2.4, …
    starti     dd 0.0, 1.0, 2.0, 3.0
    const4     dd 4.0
```

…

Пример (продолжение)


```
…
    movaps   xmm5, starti
    movss    xmm4, const4
    shufps   xmm4, xmm4, 0h ; xmm4 = 4
fori:
    movaps   xmm0, a[esi] ; xmm0 = a
    mulps    xmm0, xmm5   ; xmm0 = a*i
    addps    xmm5, xmm4   ; следующее i
    movaps   a[esi], xmm0
    add esi, 16
    loop
```

Пример

- Вычислить $b_i = a_{2*i} * a_{2*i+1}$
- За одну итерацию обрабатывать 2 раза по 4 элемента a и получать 4 элемента b

```
…
    xor esi, esi
fori:
```

Пример (продолжение)

```
movaps xmm0, a[2*esi]      ; xmm0 = 3,2,1,0
movaps xmm4, a[2*esi+16]   ; xmm4 = 7,6,5,4

; часть 1
movaps    xmm1, xmm0
shufps    xmm1, xmm0, 088h ; 88 = 10 00 10 00
                           ; xmm1 = 2, 0, …
shufps    xmm0, xmm0, 0DDh ; DD = 11 01 11 01
                           ; xmm0 = 3, 1, …
mulps     xmm1, xmm0       ; xmm1 = 2*3, 0*1, …
```

Пример (продолжение)

```
; часть 2
shufps xmm5, xmm4, 088h ; xmm2 = 6, 4, …
shufps xmm4, xmm4, 0DDh ; xmm0 = 7, 5, …
mulps  xmm5, xmm4       ; xmm5 = 6*7, 5*4,…

; объединение
shufps xmm1, xmm5, 0EEh ; E4 = 11 10 11 10
          ; xmm5 = 6*7, 5*4, 2*3, 0*1
movaps   b[esi], xmm1

add esi, 16
loop fori
```

Некоторые инструкции SSE3

инструкция	результат (A=**dst**, B=**src**)			
	dst3	dst2	dst1	dst0
ADDSUBPS (Add Subtract Packed)	A3+B3	A2−B2	A1+B1	A0−B0
HADDPS (Horizontal Add Packed)	B2+B3	B0+B1	A2+A3	A0+A1
HSUBPS (Horizontal Subtract Packed)	B2-B3	B0-B1	A2-A3	A0-A1