

Что следует помнить C++ разработчику об архитектуре процессора?

Александр Коботов



Quiz (о вас)

L R

11. Знаю число исполнительных портов Haswell

01. Делал анроллинг

10. Знаю что такое SIMD

00. Остальные

Представляюсь



Александр Коботов

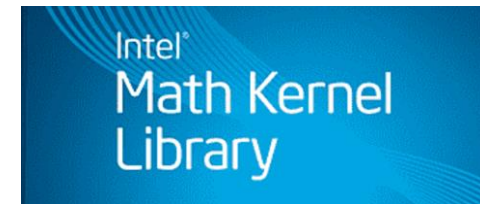
1. 30+ лет с компаниями:

- Опыт от МК54, Spectrum, ЕС ЭВМ, Yamaha MSX, 8086, 80486DX, ..., SUN Sparc, Intel IA64, Intel Xeon Phi, Huawei Taishan

2. Оптимизирую математические библиотеки.

Поработал в Sun, Intel, Яндекс, Align Tech. Теперь Huawei.

3. Преподаю в Новосибирском Гос. Университете, пишу музыку.



Yandex
CatBoost

Поговорим о:

1. Для чего помнить об архитектуре?
- 2. Что там в процессоре?**
3. Как же воспользоваться процессором на 100%?

Цена медленного кода



Killer
Serial
Stories

KILLER APP

Программист Джо не оптимизировал загрузчик операционной системы.

- Каждый пользователь **ждет 15 лишних секунд** в среднем.
- **Пользователей - 100 миллионов** каждый день

Цена медленного кода



KILLER APP

Потери:

- 417 тысяч часов (≈ 1.5 млрд секунд) ежедневно.

Цена медленного кода



KILLER APP

Косвенные потери:

- 420 тыс часов – это среднее время жизни человека (72 года минус сон)
- Значит... **Джо лишает 1ого человека жизни каждый день!!!**

Цена медленного кода



Как тебе спится, Джо — серийный программист?

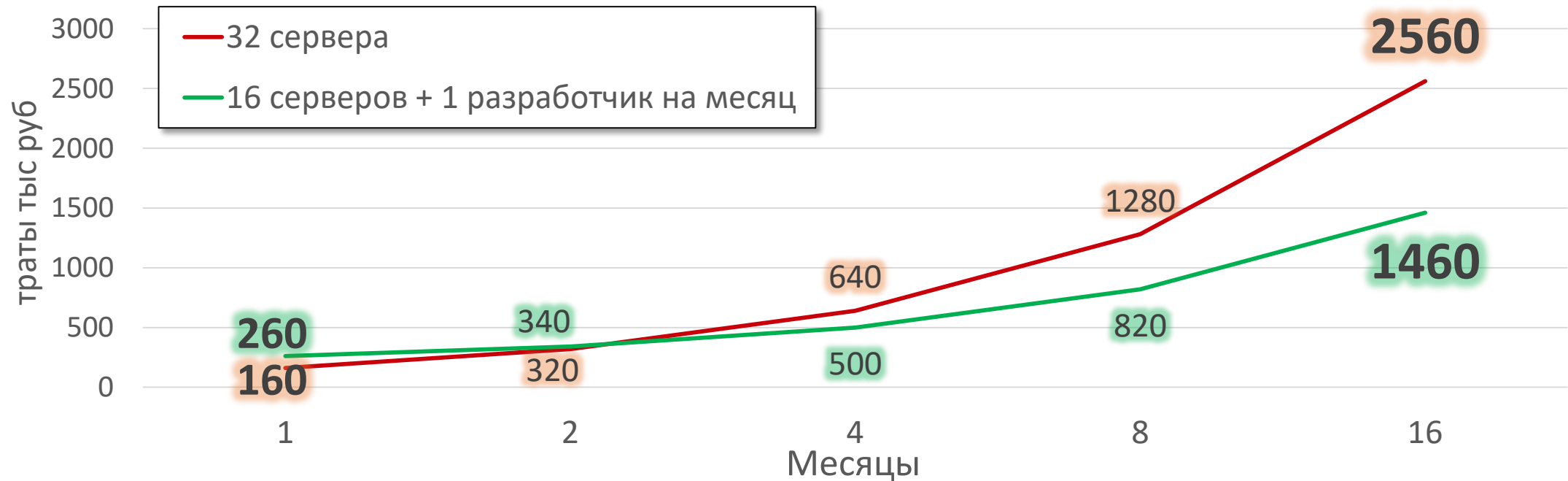
Железо



Разработчик

- Amazon AWS EC2 (4xCPU, 16Gb – t4g.xlarge)
 - ~5 тыс руб каждый месяц

- Разработчик (разово)
 - ~100 тыс руб месяц для компании
 - (Нск, средняя зп 65000руб + доп расходы - по вакансиям trud.ru)



Разработчик оптимизирует раз, а плохой код транжирит ресурсы каждый день.

Эффективность — ВЫГОДНО

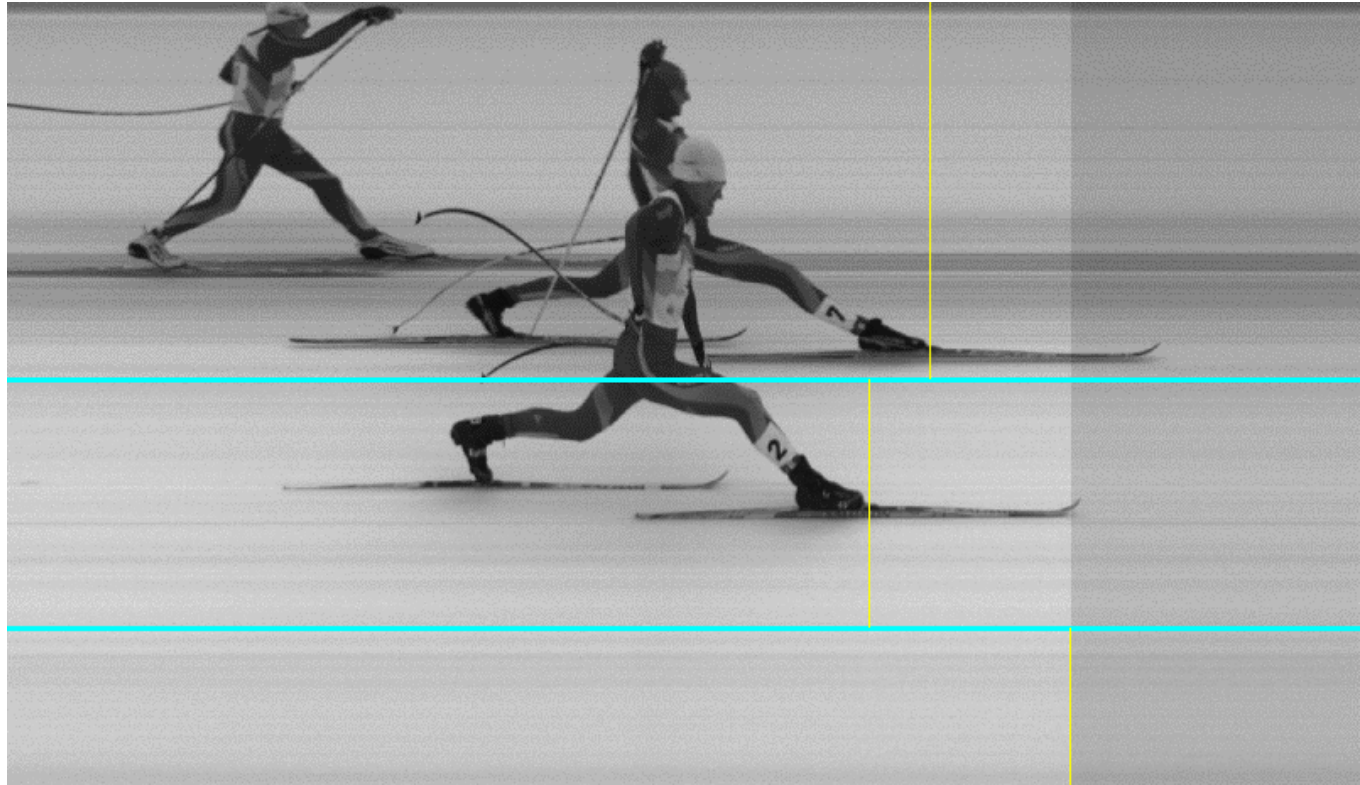
- **Самый быстрый != самый лучший**

Важна скорость на Рубль/Доллар/Юань/../Ватт



Эффективность – выгодно?

- Контрпример
 - Трейдинг/банки



Эффективность — как измеряют

- Compute bound — %% от операций/секунду
- Bandwidth bound — %% от ГБ/секунду
- Энергоэффективность — операций/ват

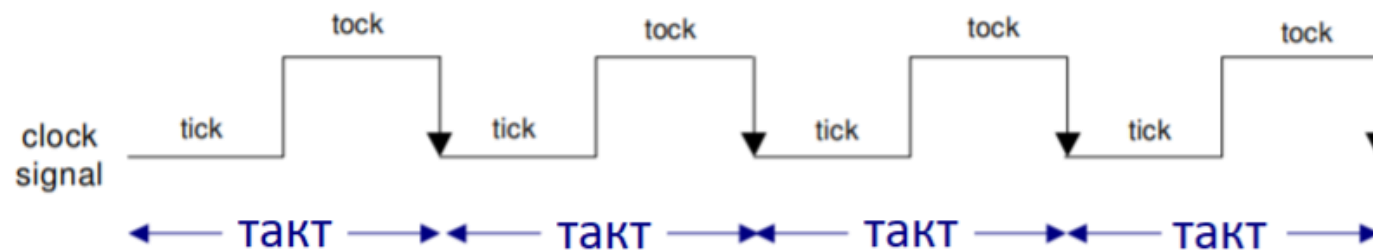


Для чего оптимизировать код?

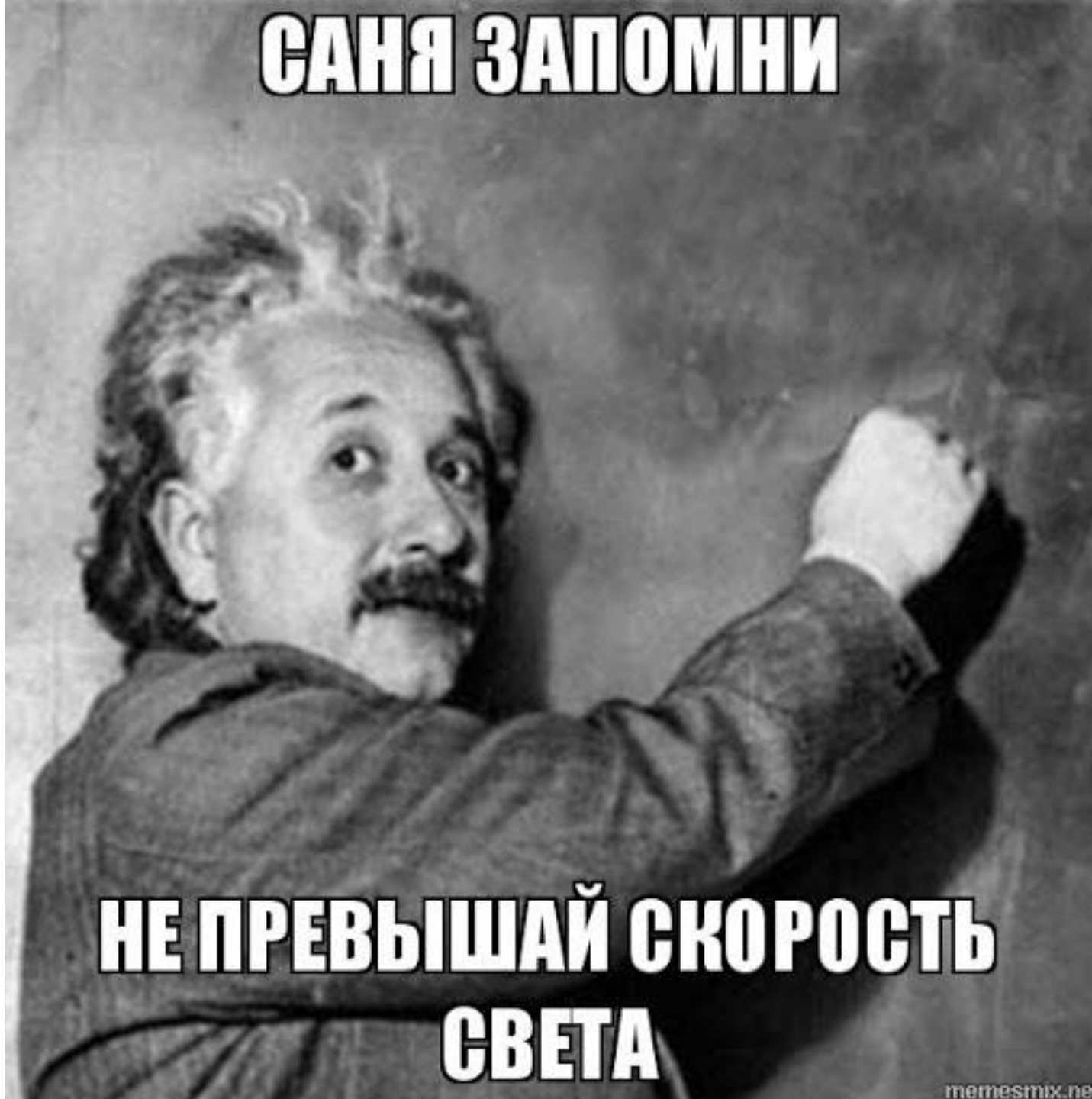
- Маркетологи же сказали что процессоры и так быстрее с каждым годом!

Speed limit

- ~3GHz – типичная частота ЦПУ
- Пока CPU делает 1 такт свет успевает преодолеть 10см



САНЯ ЗАПОМНИ



**НЕ ПРЕВЫШАЙ СКОРОСТЬ
СВЕТА**

Как же растёт производительность?

Как же растёт производительность?

Ядро современного ЦПУ(процессора):

1. Векторные инструкции: SIMD(SSE, AVX, NEON, SVE)

Как же растёт производительность?

Ядро современного ЦПУ(процессора):

1. Векторные инструкции: SIMD(SSE, AVX, NEON, SVE)
2. Конвейер

Как же растёт производительность?

Ядро современного ЦПУ(процессора):

1. Векторные инструкции: SIMD(SSE, AVX, NEON, SVE)
2. Конвейер
3. Несколько инструкций в 1 такт

Как же растёт производительность?

Ядро современного ЦПУ(процессора):

1. Векторные инструкции: SIMD(SSE, AVX, NEON, SVE)
2. Конвейер
3. Несколько инструкций в 1 такт
4. Широкий доступ в память разных уровней

Как же растёт производительность?

Ядро современного ЦПУ(процессора):

1. Векторные инструкции: SIMD(SSE, AVX, NEON, SVE)
2. Конвейер
3. Несколько инструкций в 1 такт
4. Широкий доступ в память разных уровней
5. (offtopic) и много ядер, ускорители, кластеры

Процессоры растут вширь – в параллельность разного уровня

Серверный процессор Huawei

Частота: 2.6 GHz

Архитектура: ARM v8.2 (64bit), 2019

Ядер: до 64

Память: DDR4 - 8 каналов (190Gb/s)

Когерентность: 1, 2 или 4 процессора

I/O: PCIe 4.0, CCIX, 100G, SAS/SATA 3.0

TDP: 195W

Тех процесс: 7nm, 20млрд транзисторов





Huawei Copyright

昆鹏

Kunpeng - имя могучего мифического
водного дракона с супер-способностями

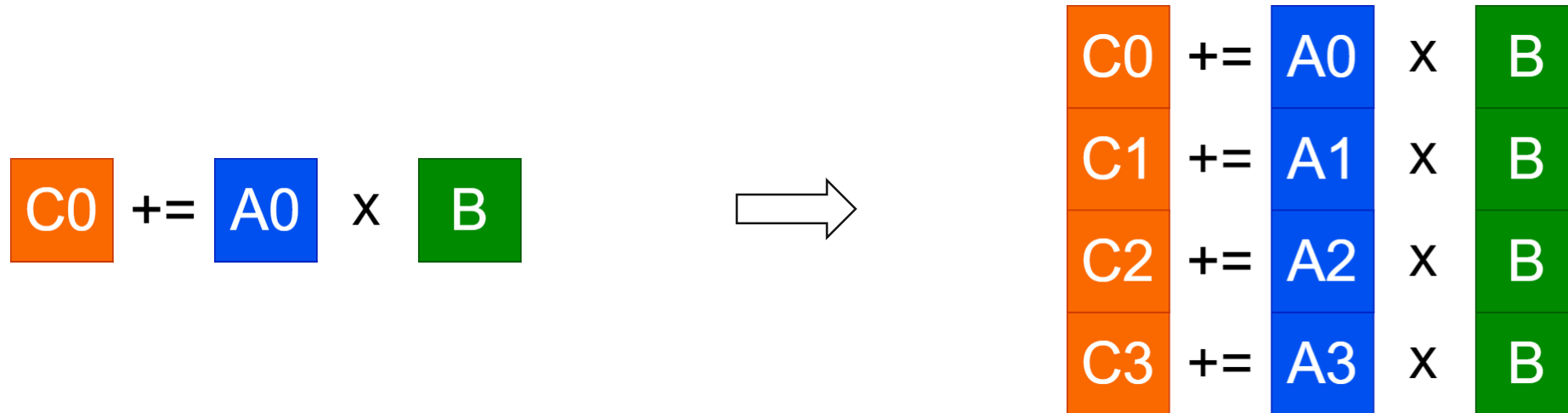


HUAWEI

Что же это там за параллельные
супер способности?

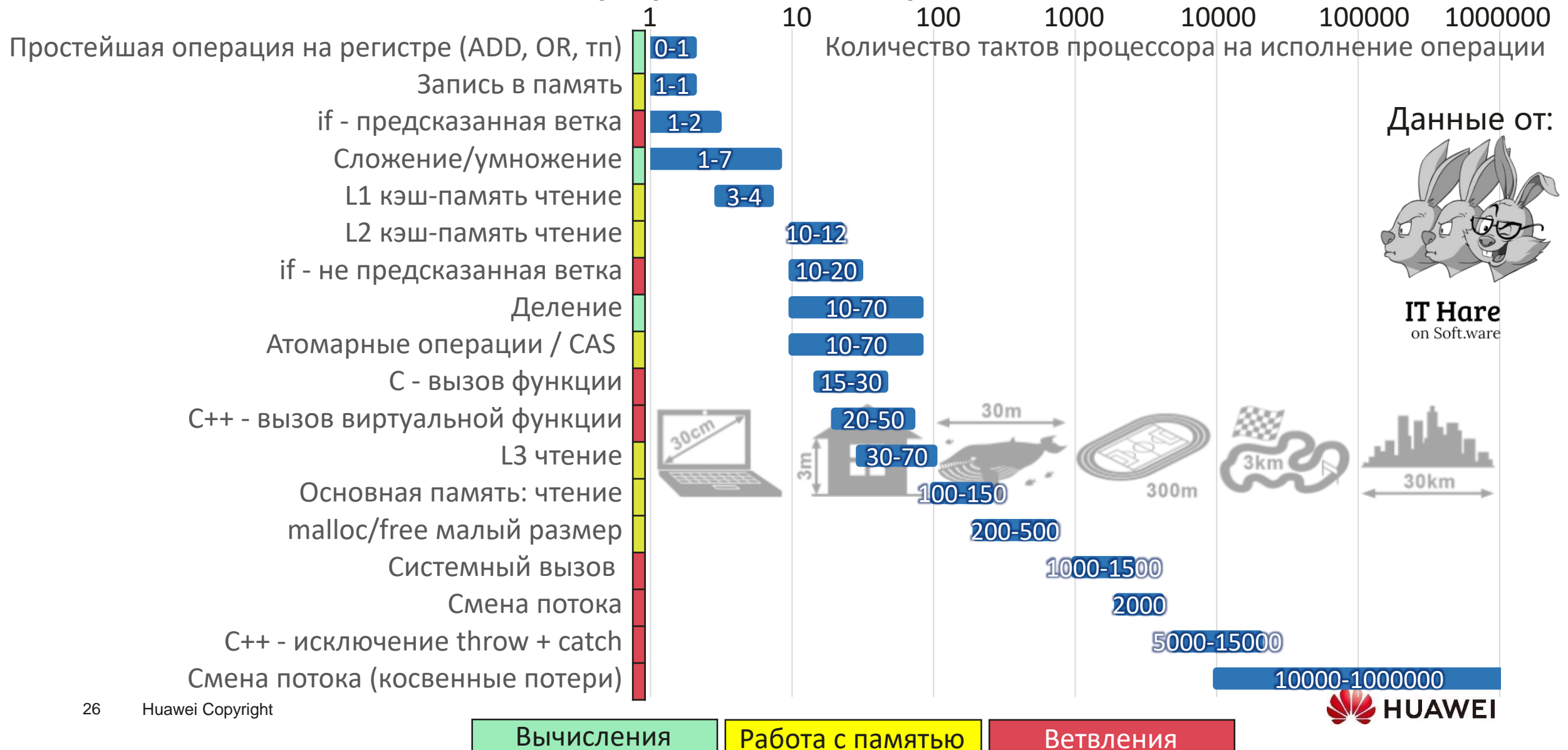
Векторные инструкции

- Одна инструкция – много данных (SIMD)

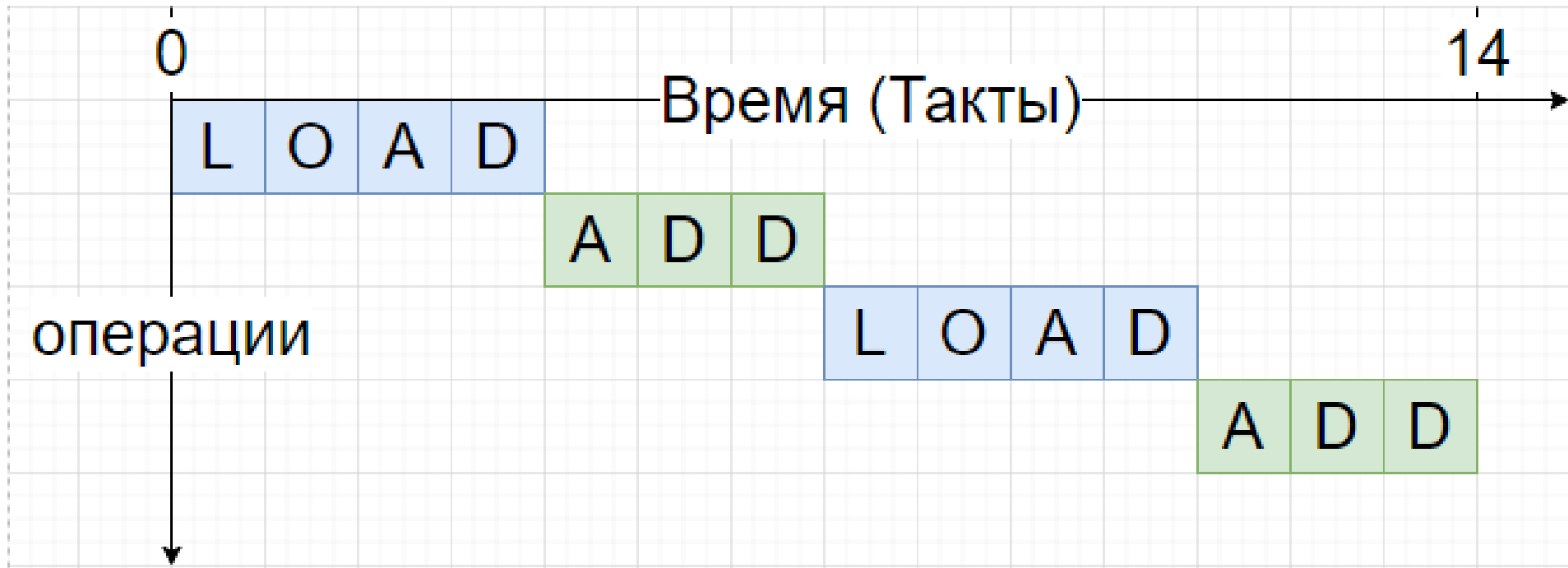


- Векторный регистр 16 байт (ARM NEON, Intel SSE)
 - 2x double; 4x float или int
 - 32-64 байта в Intel AVX1-2-512 и ARM SVE

Не все инструкции равны



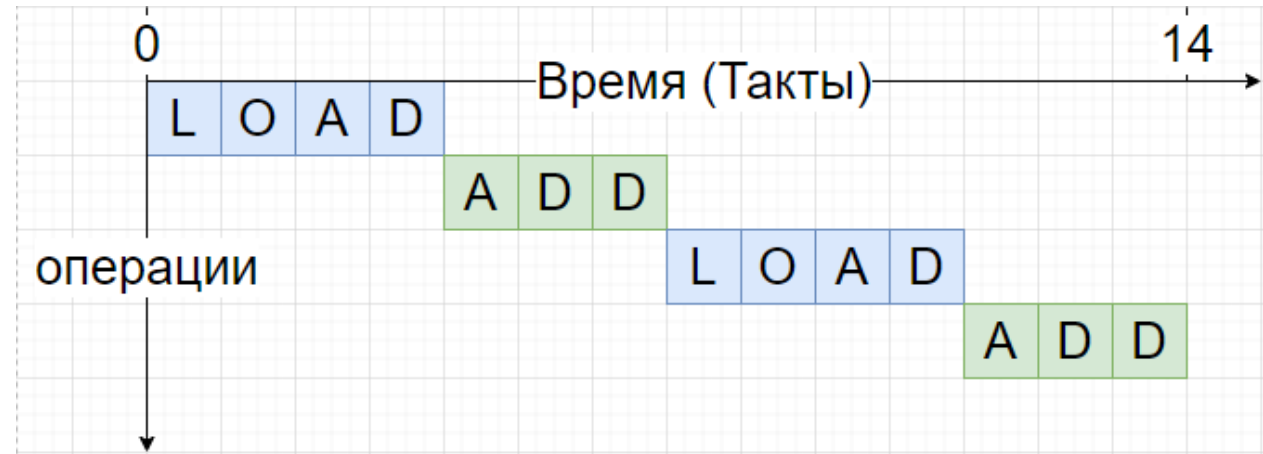
Исполнение операций



Исполнение операций

За 14 тактов:

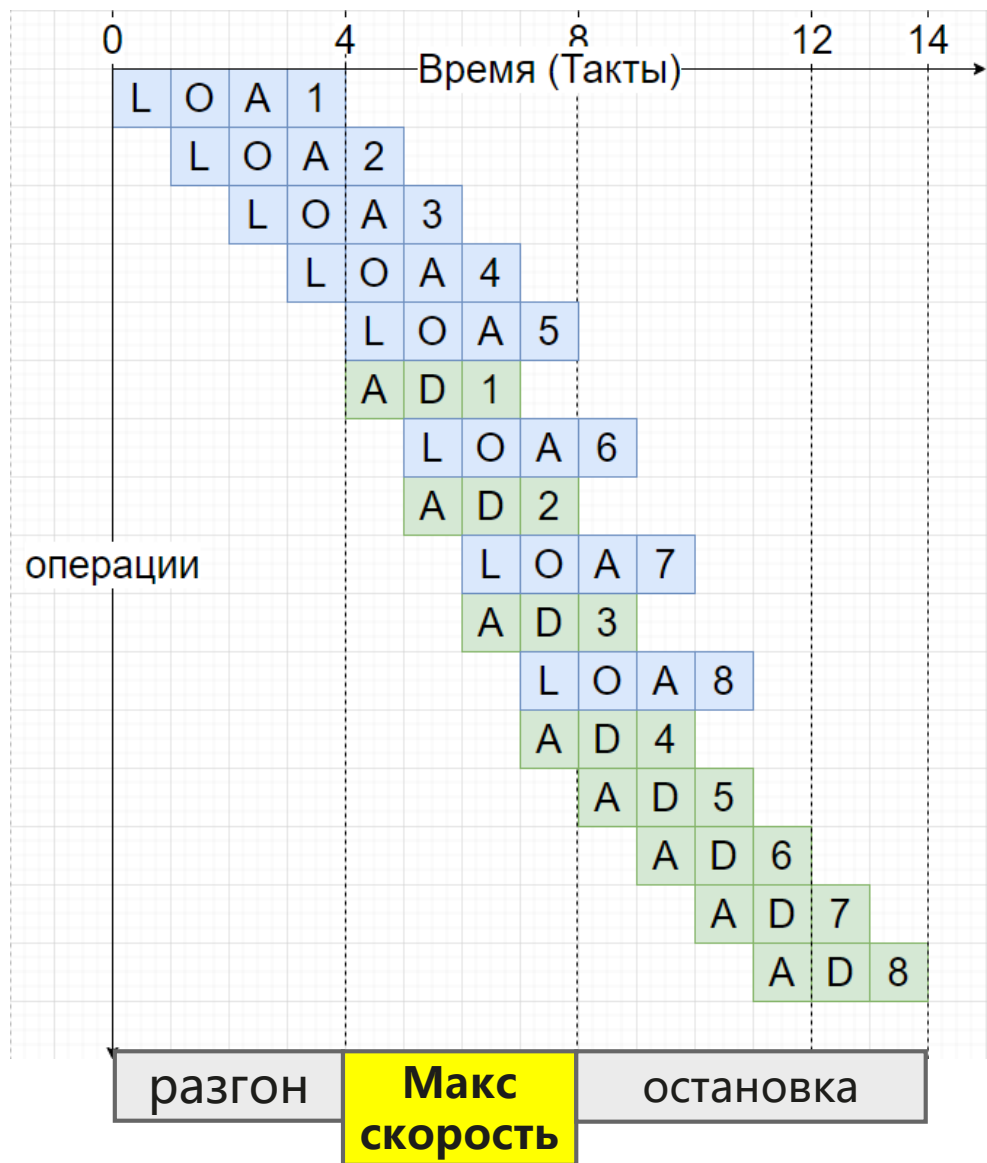
- 2 загрузки
- 2 сложения



С конвейером

За 14 тактов:

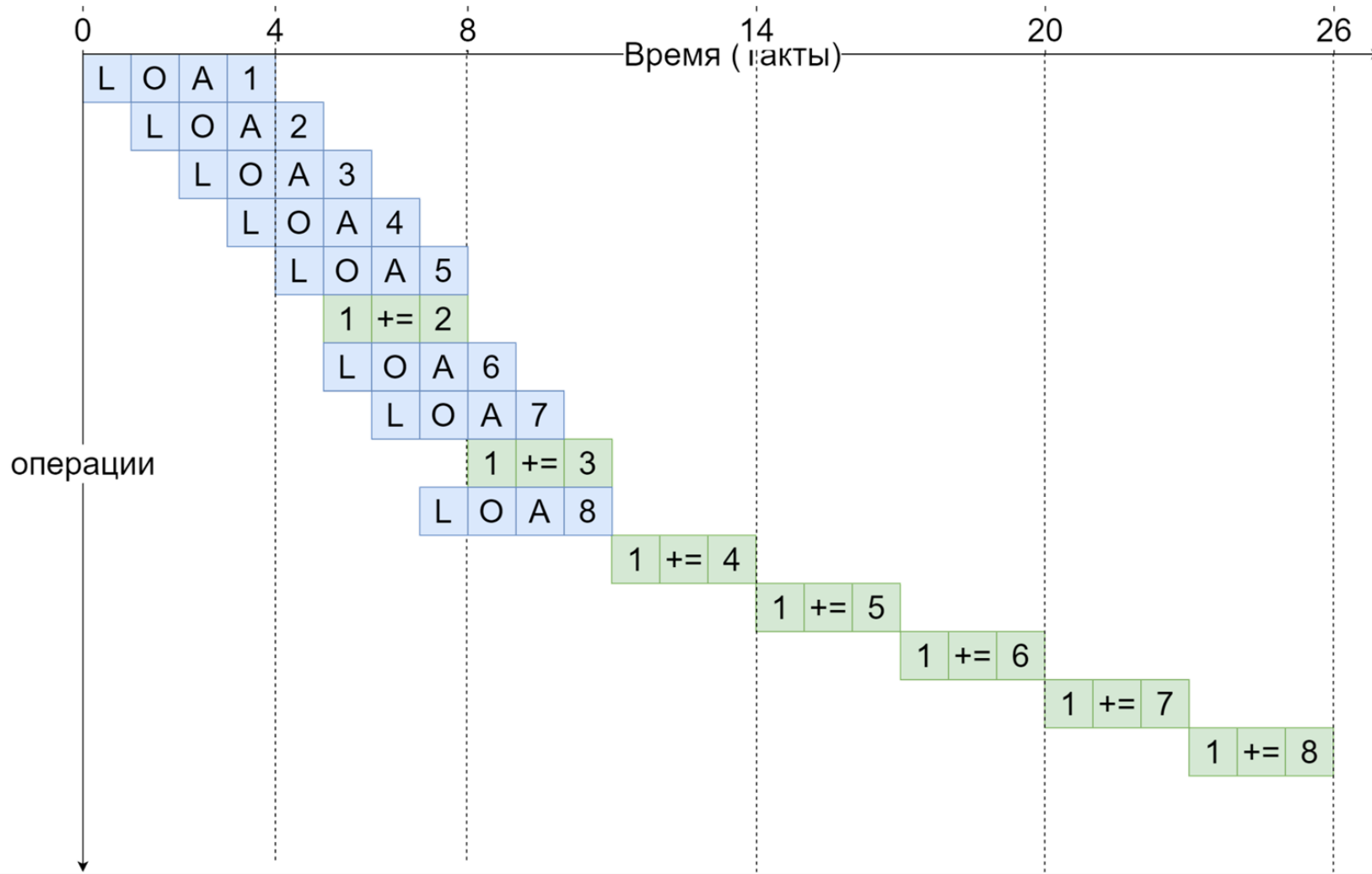
- 8 загрузок
- 8 сложений



Потенциально 2 инструкции каждый такт

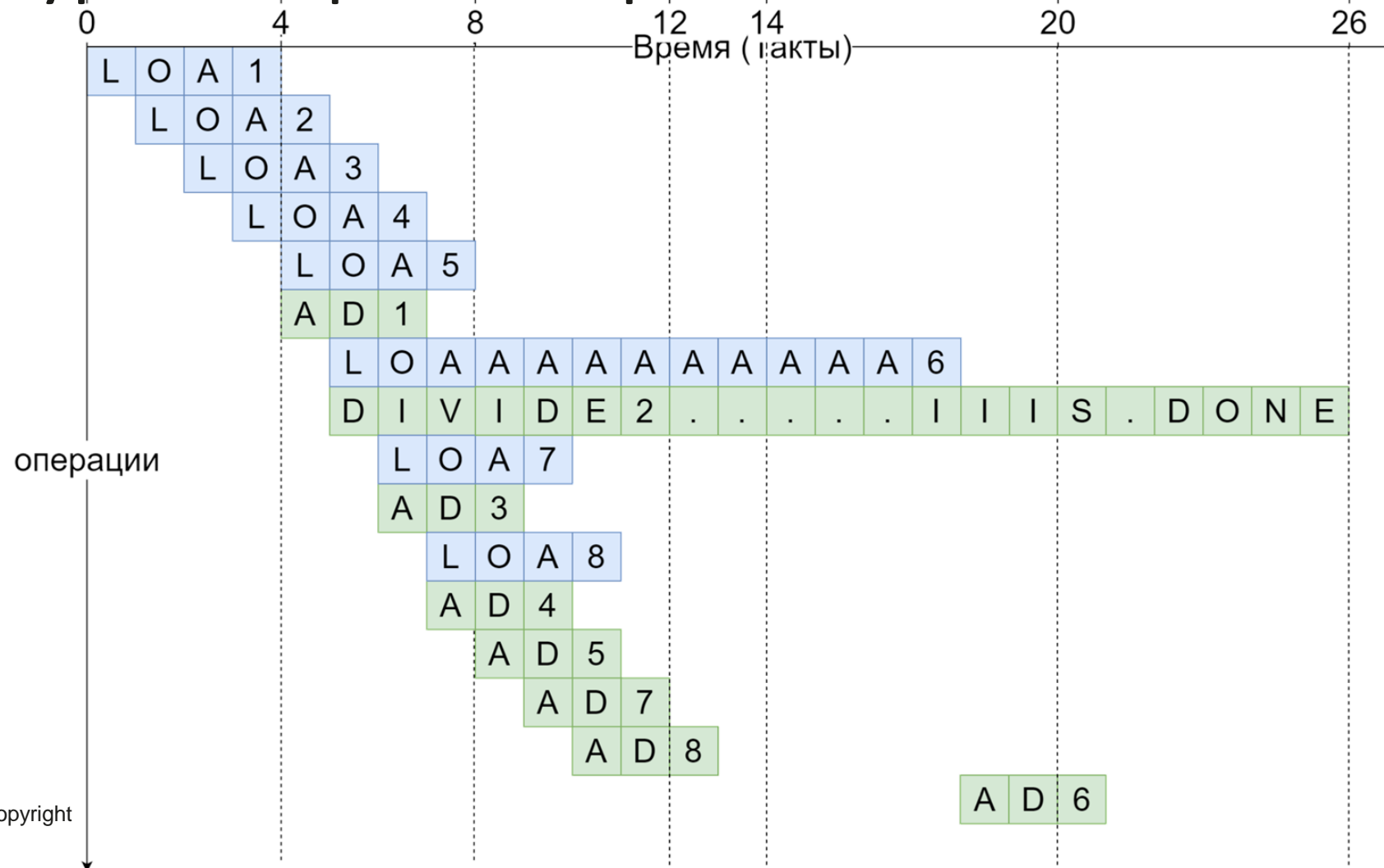
Помехи конвейера

Данные не готовы (считаются или загружаются)



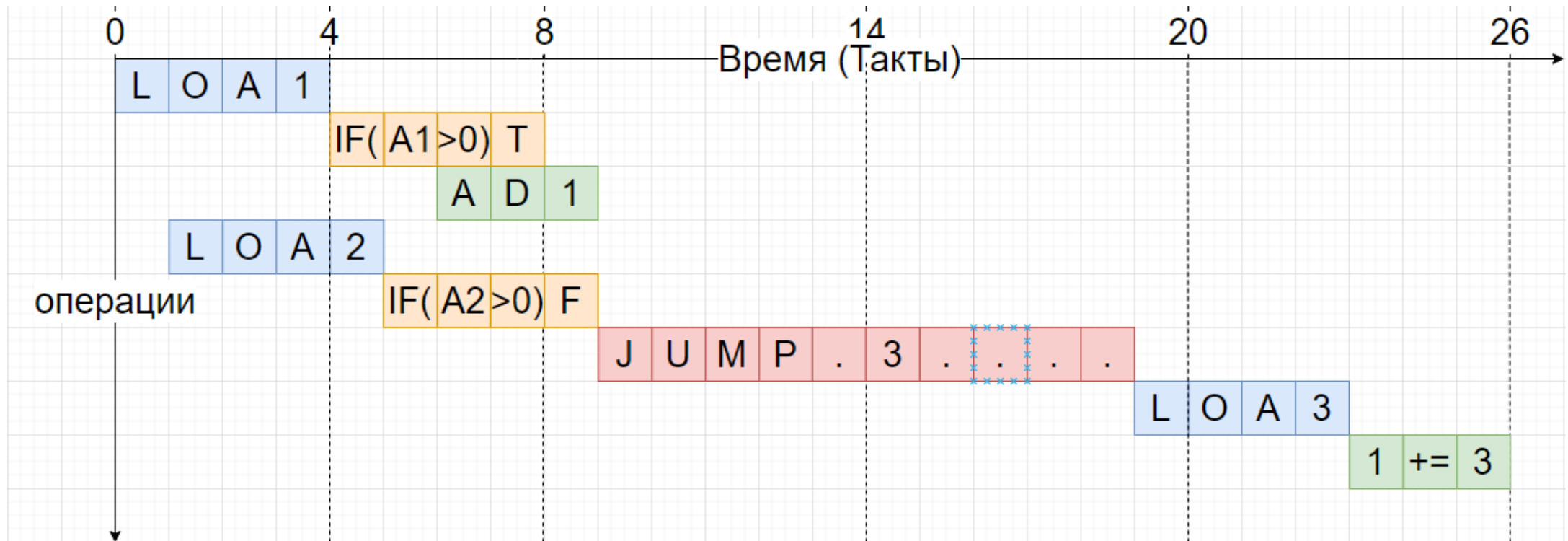
Помехи конвейера

Структурные – разное время исполнения команд

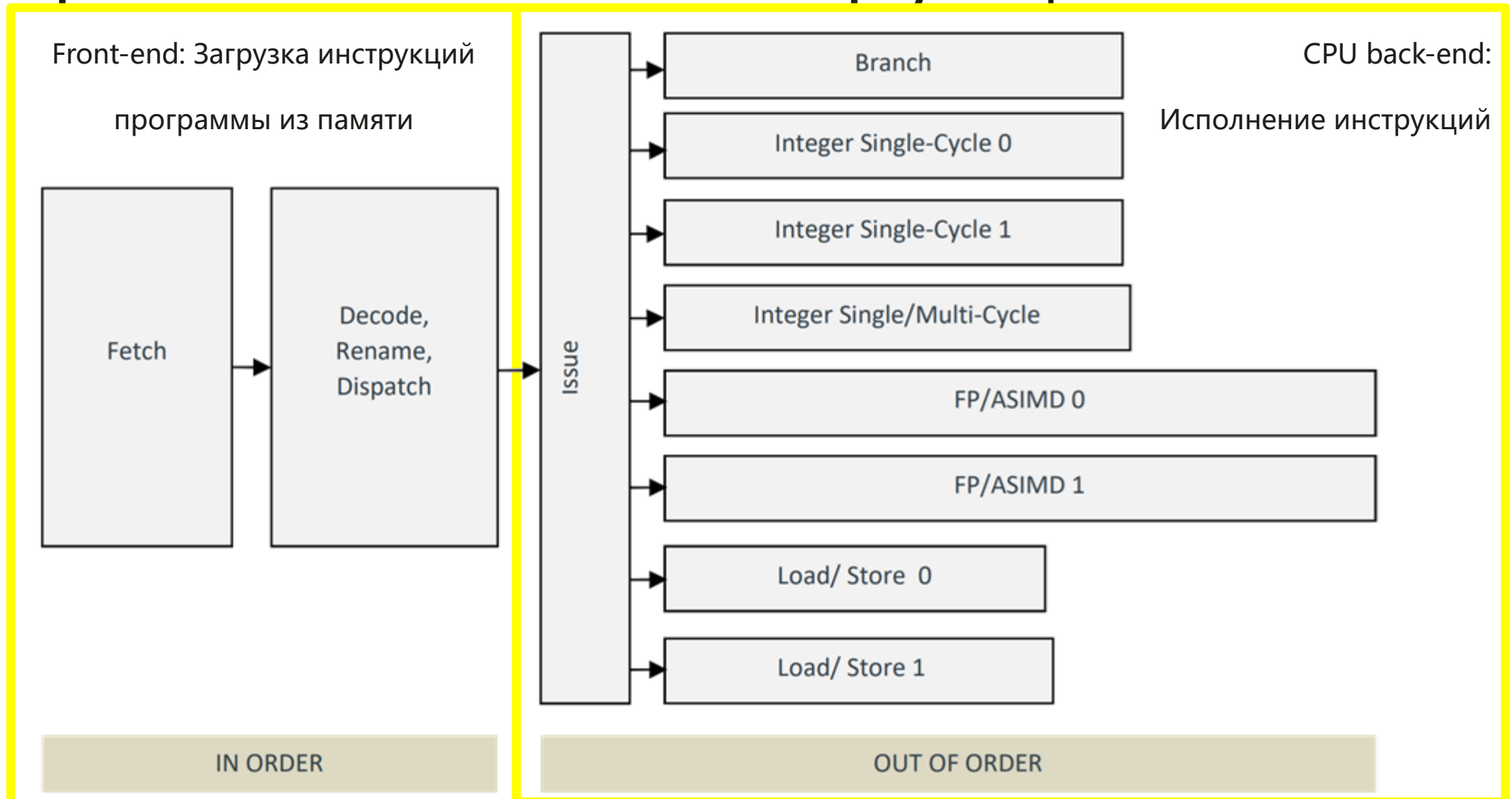


Помехи конвейера

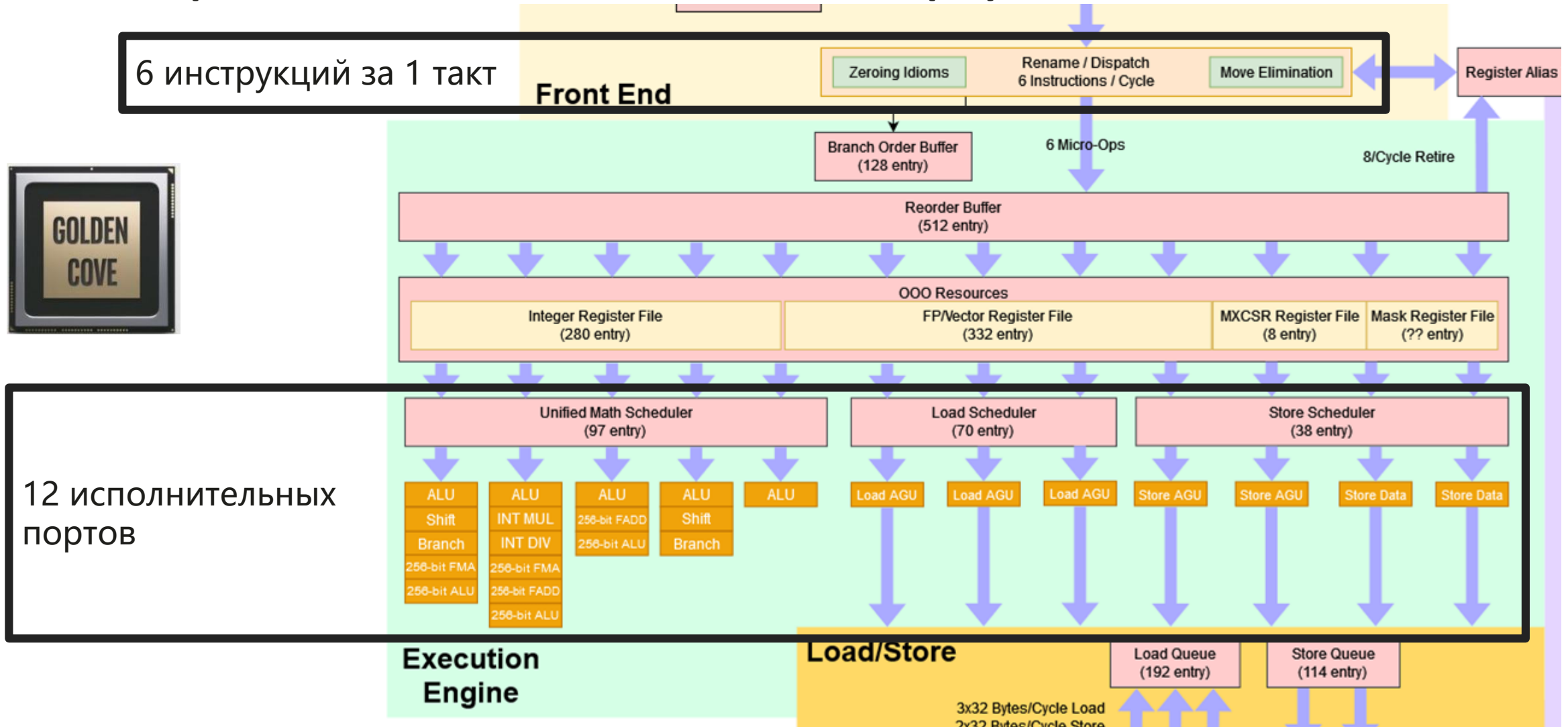
Управления – условные и безусловные переходы



Параллельные инструкции

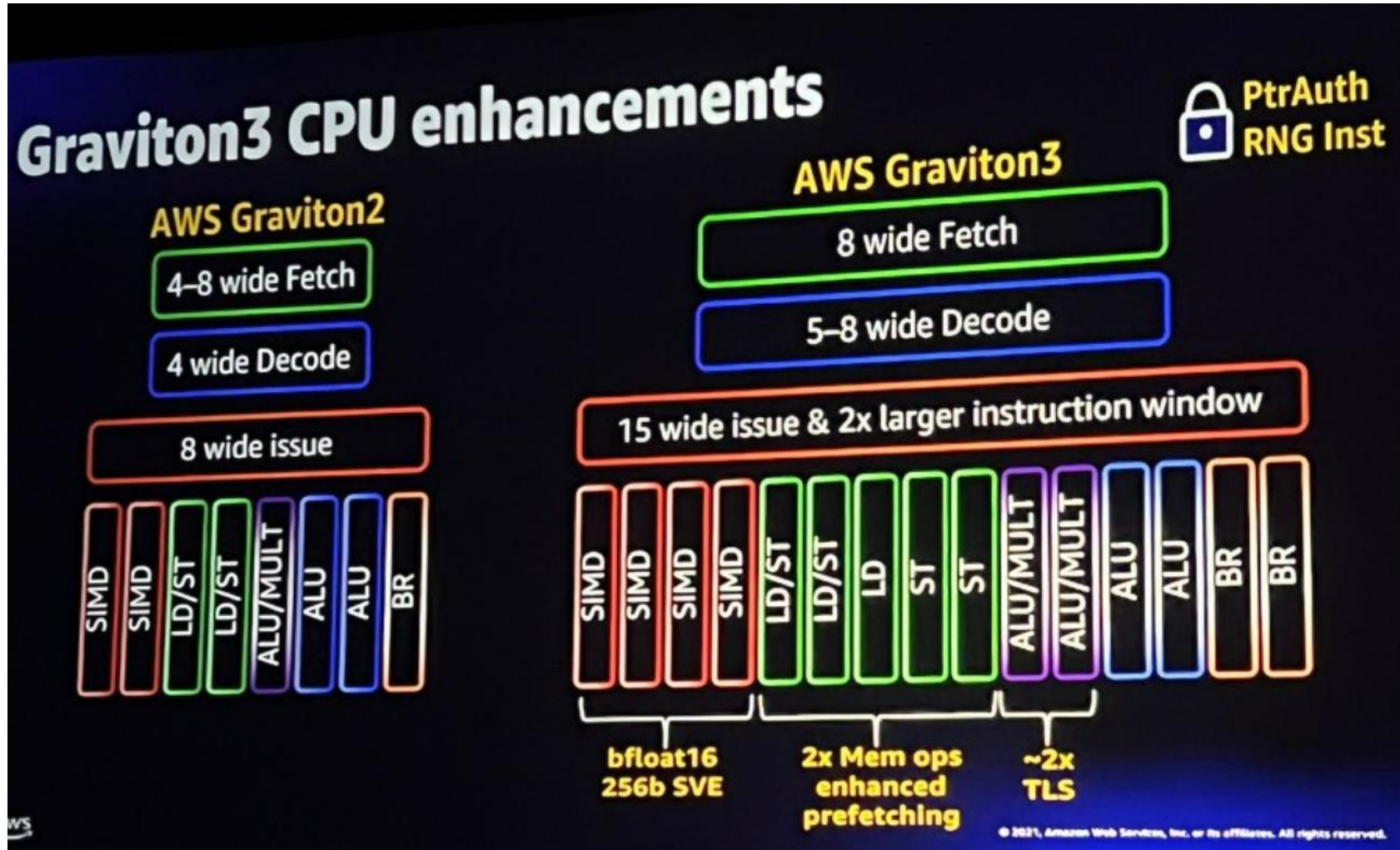


Параллельные инструкции Intel



12 исполнительных портов

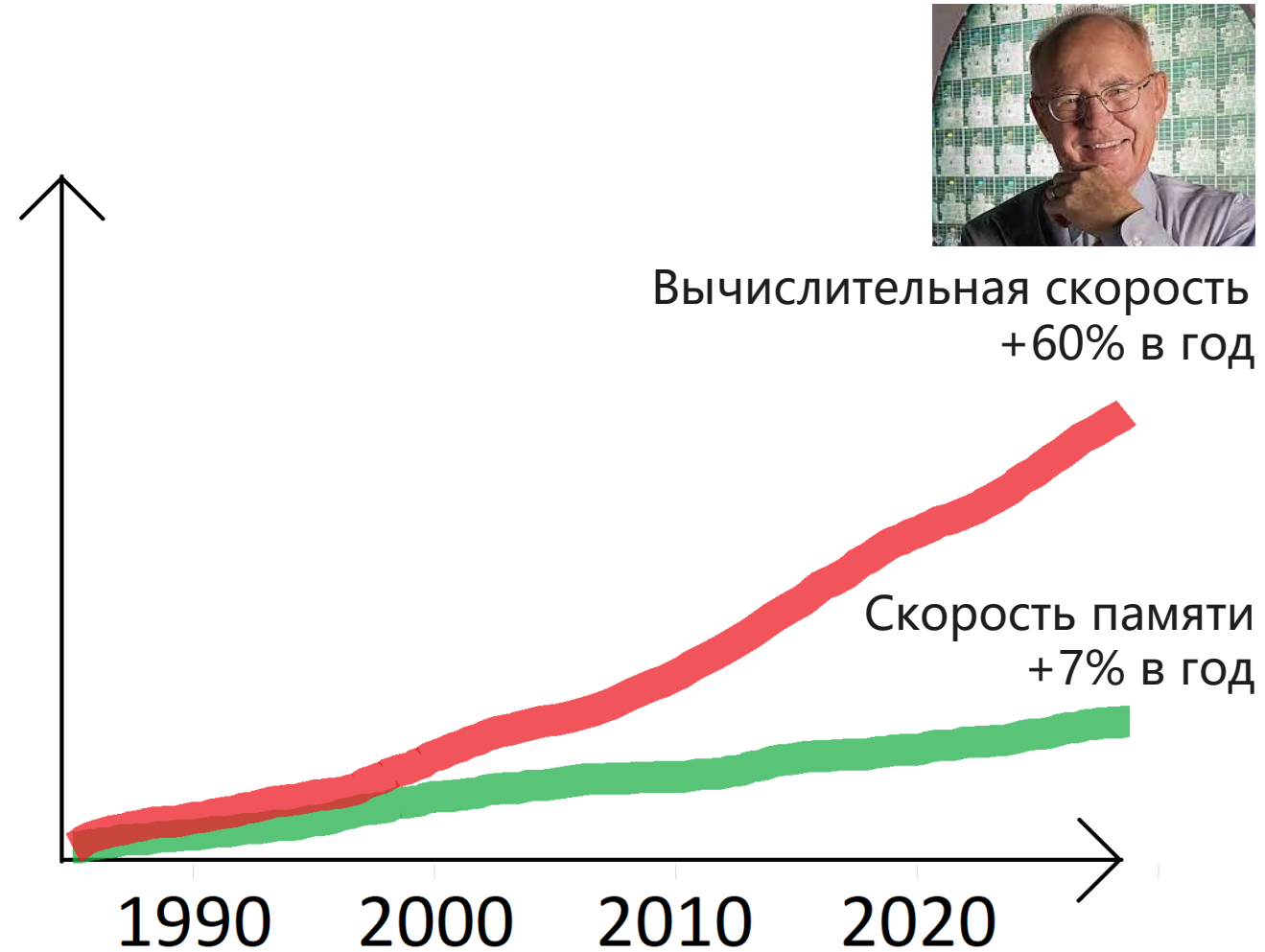
Amazon Graviton3



А как там с доступом к данным?

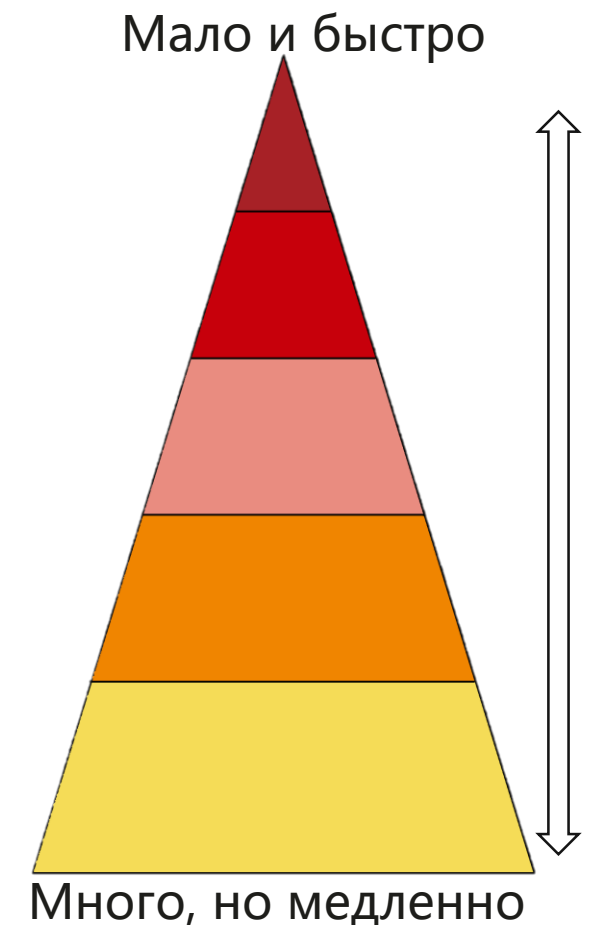
Скорость памяти

- Процессоры ускоряются быстрее чем память

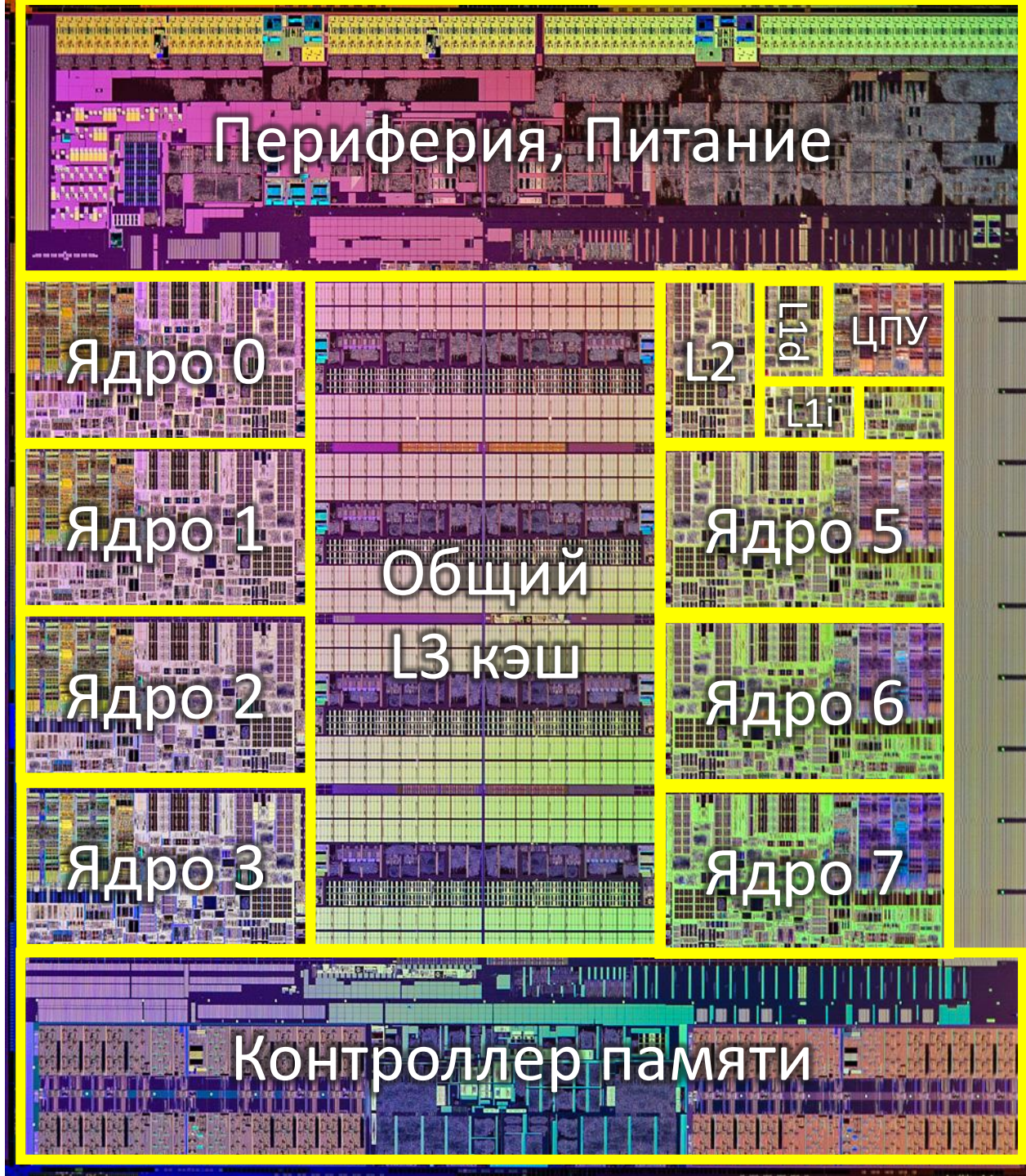


Иерархия памяти

Вид	Объём	Пропускная способность	Задержка циклы ЦПУ
Регистры	512b вект 256b общ	250 Gb/s	0
L1Data	64 Kb	80 Gb/s	4
L1Instructions	64 Kb	80 Gb/s	
L2	512 Kb	60 Gb/s	12
L3	1 Mb	40 Gb/s	30-70
Память RAM	Много	16 Gb/s	150-200 500 (NUMA)



Intel Haswell в разрезе



The picture has an illustrative purpose and is not exact

128-Bit DDR4-3200 / DDR5-4800 Physical Layer (PHY)



10nm ESF/Intel 7 Alder Lake die shot (~209mm²) from Intel via Andreas Schilling on Twitter:
<https://twitter.com/aschilling/status/1453391035577495553>

Die shot interpretation by Locuza, October 2021



Как данным попасть в кэш?

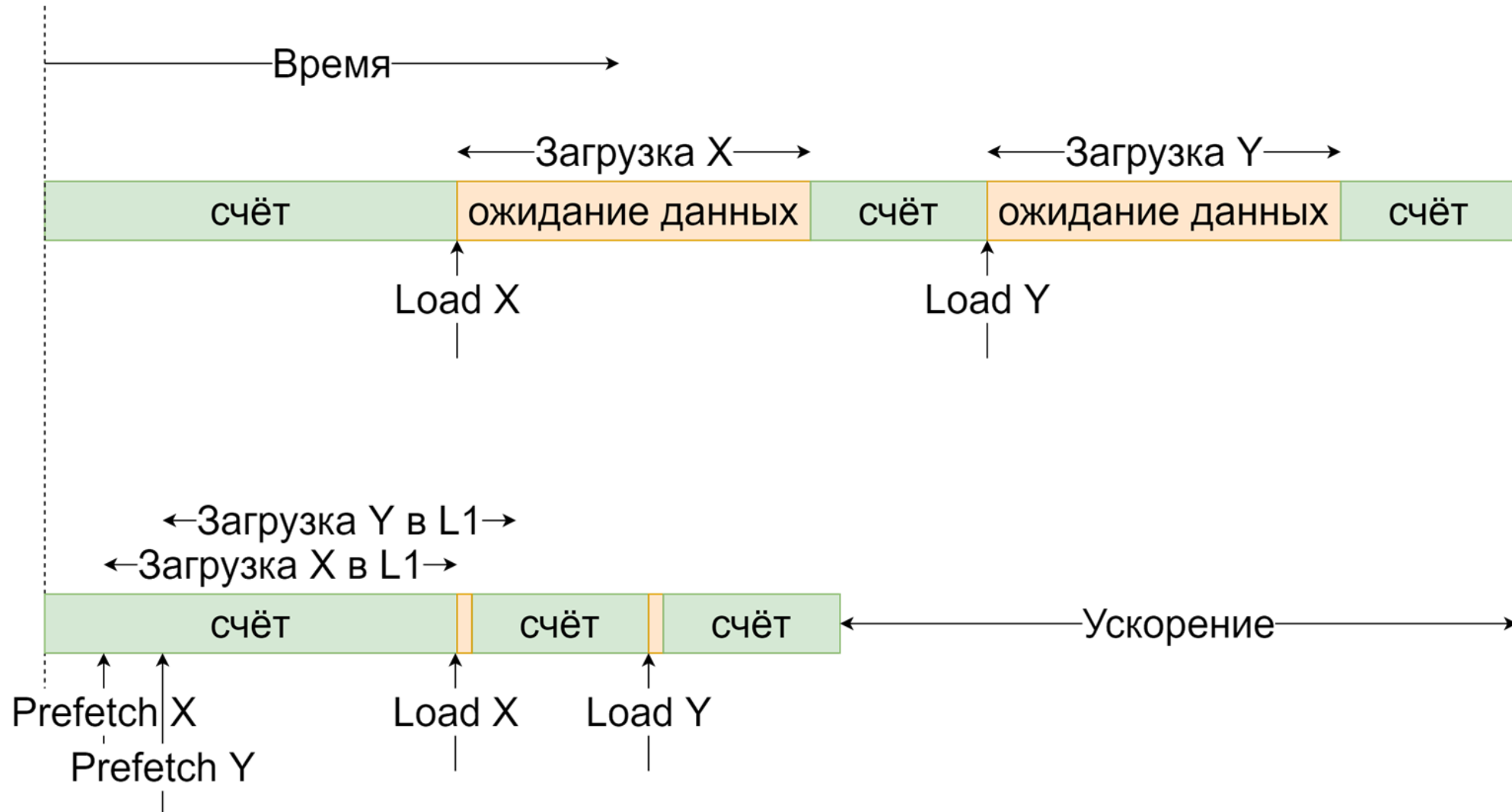
Как данным попасть в кэш?

1. Там уже недавние данные

Как данным попасть в кэш?

1. Там уже недавние данные
2. Предзагрузка (prefetch)

Предзагрузка данных (Prefetch)



Предзагрузка данных (Prefetch)

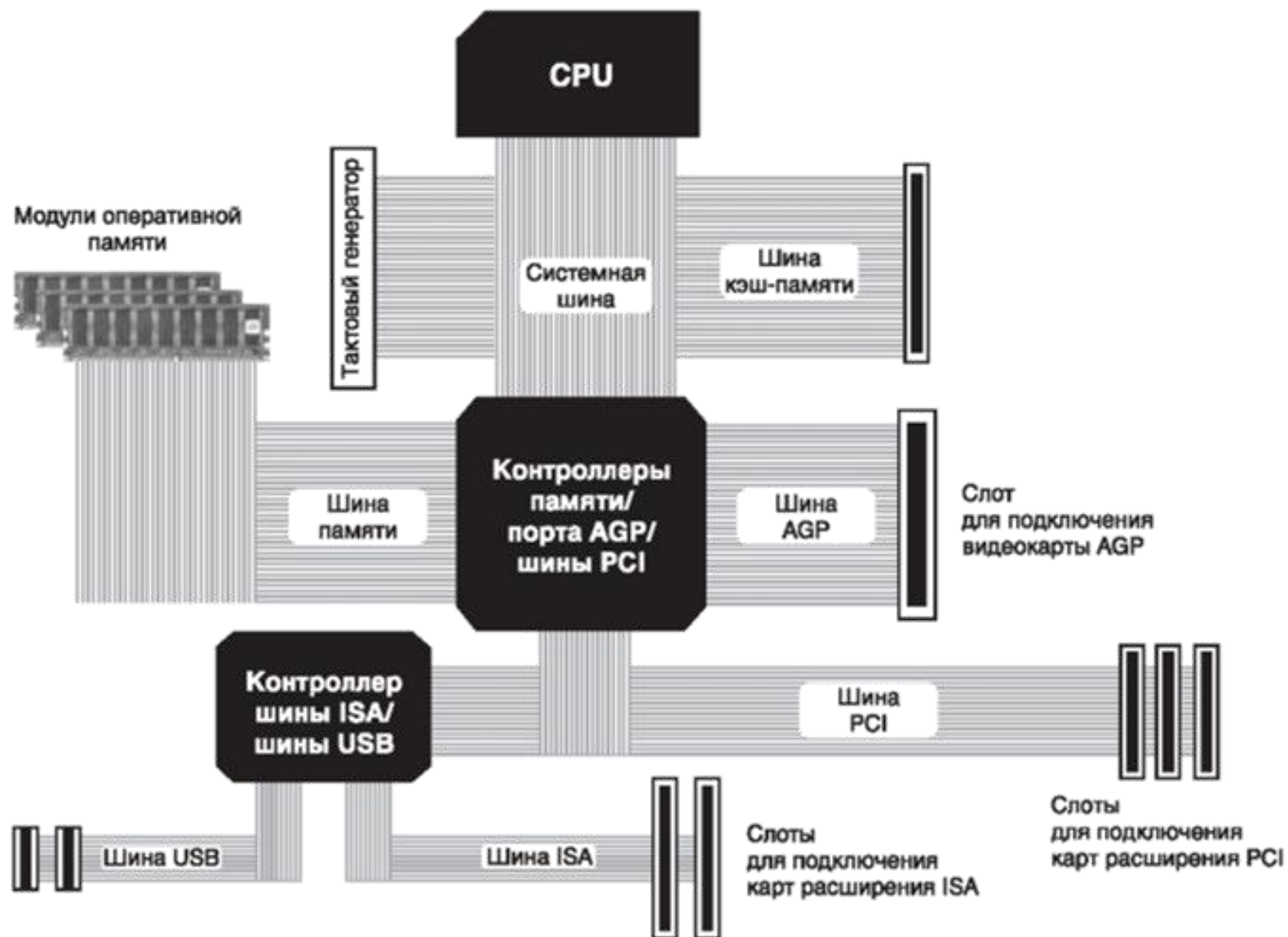
- Автоматическая (процессором)
 - При линейном доступе к данным
 - При доступе с шаблоном (Big cores)

Предзагрузка данных (Prefetch)

- Явная

- *void __builtin_prefetch (const void *addr, ...)* – загрузка кэш линии (64 байта)
- Имеет смысл при косвенной адресации `a[index[i]]`

Параллельная шина данных



Параллельная шина данных

- Любое обращение к памяти грузит одну кэш линию (64байта), даже если нужен 1 байт.
 - Выгодно: обратившись к одному байту воспользоваться всеми соседними!
 - Структура с массивами быстрее массива структур



X86 vs ARM (vs RISC-V)

	X86	ARMv8	
Количество базовых регистров	8 – X86 16 – X86_64	30	★
Количество векторных регистров	32 – AVX{1,2,512}	32	
Набор инструкций	Complex Instruction Set Computer (CISC)	Reduced Instruction Set Computer (RISC)	
Длина инструкции	1-15 байт	4 байта	★
Разработка	Intel, AMD, VIA(no longer)	Qualcomm, Apple, Huawei, Nvidia, Samsung, VIA, AMD, and so on	
Год создания	1978-2021	2011	

X86 vs ARM – Fight!

	Частота	Вычислит ельных инструкц ий за такт			Ширина вектора	Операций на инструкц ию	Итого теоретич еских GFLOPS	Реальные GFLOPS на SGEMM*	Эффекти вность
Intel SKX AVX512	2.6Ghz	*	2	*	16 float	* 2 (fma) = 166		135	81%
Huawei ARM v8	2.6Ghz	*	2	*	4 float	* 2 (fma) = 41.6		39.7	95%

* SGEMM – матричное умножение: Intel MKL 2020.1.217 vs OpenBLAS 0.3.13

Как же воспользоваться
процессором на 100%?

Код, ненавистный CPU

1. Вызов мелких функций (без inline)

1. -сохранение состояния регистров, и восстановление по возврату из функции



Код, ненавистный CPU

1. Вызов мелких функций (без inline)

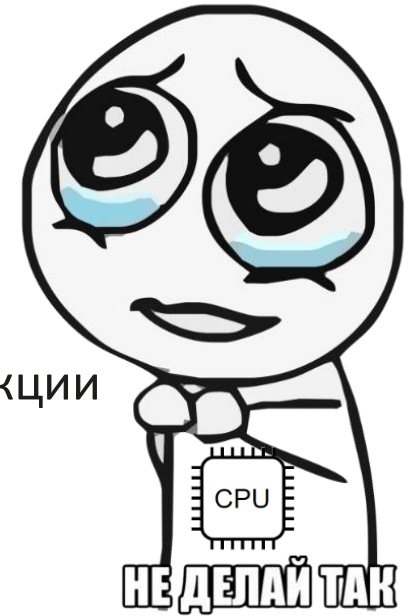
1. -сохранение состояния регистров, и восстановление по возврату из функции

2. условие в цикле

1. -векторизация, -спотыкач на предсказании ветвлений
2. +AVX512, SVE2 получили маски для векторных инструкций



Код, ненавистный CPU



1. Вызов мелких функций (без inline)

1. -сохранение состояния регистров, и восстановление по возврату из функции

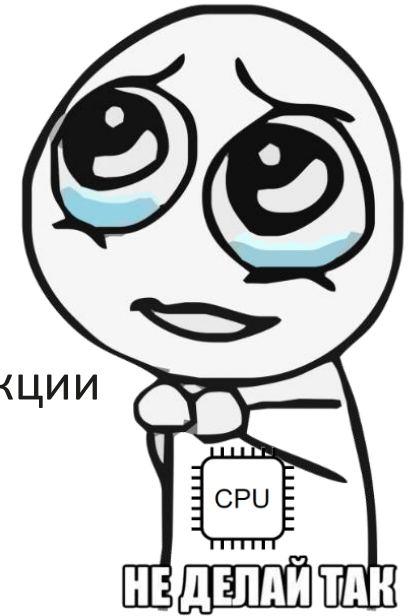
2. условие в цикле

1. -векторизация, -спотыкач на предсказании ветвлений
2. +AVX512, SVE2 получили маски для векторных инструкций

3. Косвенная адресация: $a[index[mapping[encode[i]]]]$

1. -предзагрузка, +лаги на ожидание данных

Код, ненавистный CPU



1. Вызов мелких функций (без inline)

1. -сохранение состояния регистров, и восстановление по возврату из функции

2. условие в цикле

1. -векторизация, -спотыкач на предсказании ветвлений
2. +AVX512, SVE2 получили маски для векторных инструкций

3. Косвенная адресация: $a[index[mapping[encode[i]]]]$

1. -предзагрузка, +лаги на ожидание данных

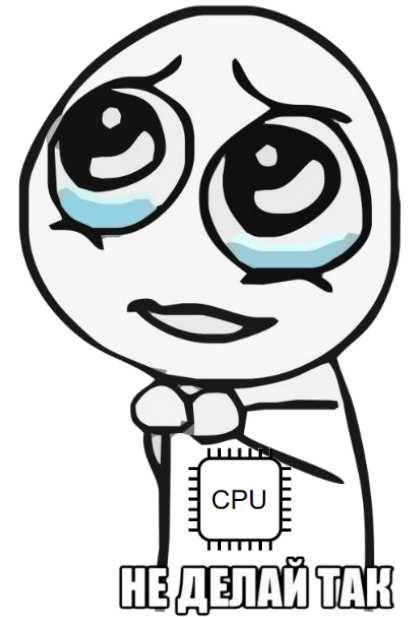
4. Зависимость по данным у соседних строк кода

1. -конвейер, +ждём окончания предыдущей инструкции

НУ ПОЖАЛУЙСТА

Код, ненавистный CPU

- 5. Много мелких виртуальных функций
- 6. Много операций деления
- 7. Много атомарных операции



Любимый код компилятора и CPU

- Небольшой цикл на много итераций с повторяющимися независимыми последовательными данными
 1. Код цикла в L1
 2. Повторяющиеся данные в L1 и L2
 3. Независимые данные => конвейер и суперскалярность
 4. Последовательные данные – векторизация и prefetch
 5. Матрицы – AMX, SME
- Функций либо нет, либо все inline

Пример кода

```
1 void scale16(float alpha, float *a) {  
2     for(int i = 0; i < 16; ++i)  
3         a[i] *= alpha;  
4 }
```

gcc-12.1 -O3 -msse4

```
1 scale16(float, float*):  
2     movups    (%rdi), %xmm1  
3     shufps    $0, %xmm0, %xmm0  
4     movups    48(%rdi), %xmm2  
5     mulps     %xmm0, %xmm1  
6     movups    %xmm1, (%rdi)  
7     movups    16(%rdi), %xmm1  
8     mulps     %xmm0, %xmm1  
9     movups    %xmm1, 16(%rdi)  
10    movups    32(%rdi), %xmm1  
11    mulps     %xmm0, %xmm1  
12    mulps     %xmm2, %xmm0  
13    movups    %xmm1, 32(%rdi)  
14    movups    %xmm0, 48(%rdi)  
15    ret
```

Эмулятор ЦПУ

gem5.org:

- Clock accurate
- Out-of-order emulation



Исполнение

```

1 void scale16(float alpha, float *a) {
2     for(int i = 0; i < 16; ++i)
3         a[i] *= alpha;
4 }

```

```

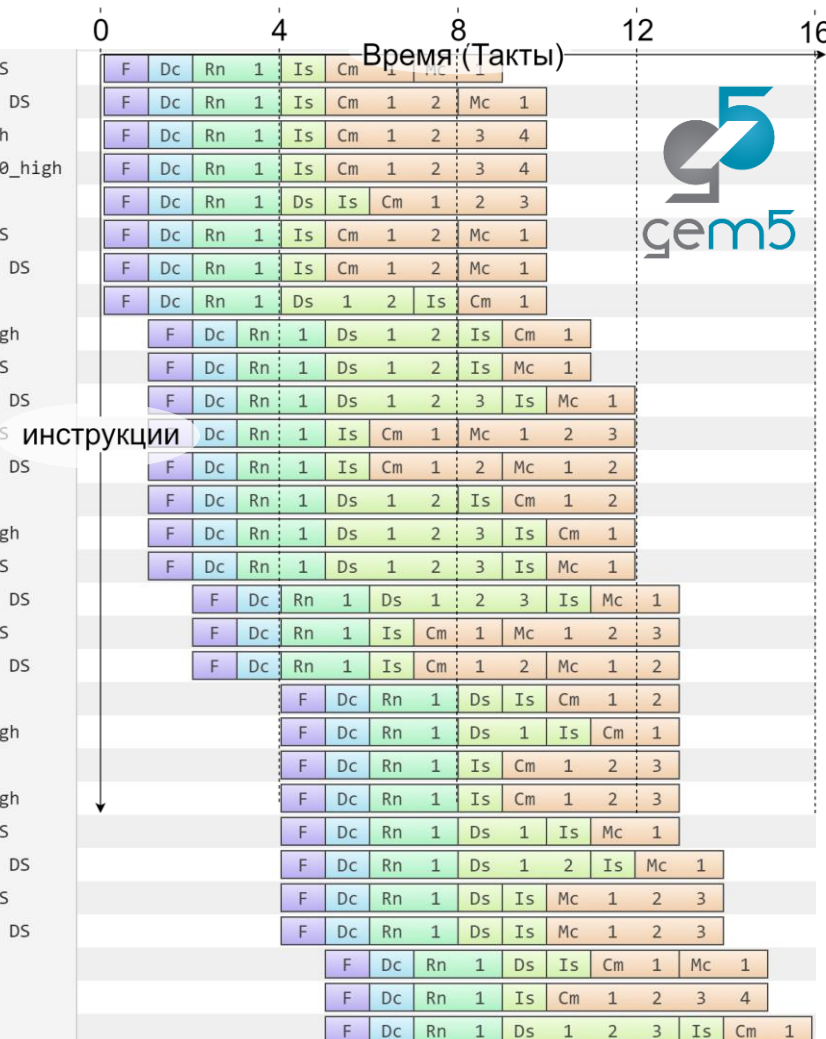
1 scale16(float, float*):
2     movups    (%rdi), %xmm1
3     shufps    $0, %xmm0, %xmm0
4     movups    48(%rdi), %xmm2
5     mulps     %xmm0, %xmm1
6     movups    %xmm1, (%rdi)
7     movups    16(%rdi), %xmm1
8     mulps     %xmm0, %xmm1
9     movups    %xmm1, 16(%rdi)
10    movups    32(%rdi), %xmm1
11    mulps     %xmm0, %xmm1
12    mulps     %xmm2, %xmm0
13    movups    %xmm1, 32(%rdi)
14    movups    %xmm0, 48(%rdi)
15    ret

```

```

MOVUPS_XMM_M : ldfp    %xmm1_low, DS: ldfp    %xmm1_low, DS
MOVUPS_XMM_M : ldfp    %xmm1_high, DS: ldfp    %xmm1_high, DS
SHUFPS_XMM_XMM_I : shuffle    %ufp1, %xmm0_low, %xmm0_high
SHUFPS_XMM_XMM_I : shuffle    %xmm0_high, %xmm0_low, %xmm0_high
SHUFPS_XMM_XMM_I : movfp    %xmm0_low, %ufp1
MOVUPS_XMM_M : ldfp    %xmm2_low, DS: ldfp    %xmm2_low, DS
MOVUPS_XMM_M : ldfp    %xmm2_high, DS: ldfp    %xmm2_high, DS
MULPS_XMM_XMM : mmulf    %xmm1_low, %xmm1_low, %xmm0_low
MULPS_XMM_XMM : mmulf    %xmm1_high, %xmm1_high, %xmm0_high
MOVUPS_M_XMM : stfp    %xmm1_low, DS: stfp    %xmm1_low, DS
MOVUPS_M_XMM : stfp    %xmm1_high, DS: stfp    %xmm1_high, DS
MOVUPS_XMM_M : ldfp    %xmm1_low, DS: ldfp    %xmm1_low, DS
MOVUPS_XMM_M : ldfp    %xmm1_high, DS: ldfp    %xmm1_high, DS
MULPS_XMM_XMM : mmulf    %xmm1_low, %xmm1_low, %xmm0_low
MULPS_XMM_XMM : mmulf    %xmm1_high, %xmm1_high, %xmm0_high
MOVUPS_M_XMM : stfp    %xmm1_low, DS: stfp    %xmm1_low, DS
MOVUPS_M_XMM : stfp    %xmm1_high, DS: stfp    %xmm1_high, DS
MOVUPS_XMM_M : ldfp    %xmm1_low, DS: ldfp    %xmm1_low, DS
MOVUPS_XMM_M : ldfp    %xmm1_high, DS: ldfp    %xmm1_high, DS
MULPS_XMM_XMM : mmulf    %xmm1_low, %xmm1_low, %xmm0_low
MULPS_XMM_XMM : mmulf    %xmm1_high, %xmm1_high, %xmm0_high
MULPS_XMM_XMM : mmulf    %xmm0_low, %xmm0_low, %xmm2_low
MULPS_XMM_XMM : mmulf    %xmm0_high, %xmm0_high, %xmm2_high
MOVUPS_M_XMM : stfp    %xmm1_low, DS: stfp    %xmm1_low, DS
MOVUPS_M_XMM : stfp    %xmm1_high, DS: stfp    %xmm1_high, DS
MOVUPS_M_XMM : stfp    %xmm0_low, DS: stfp    %xmm0_low, DS
MOVUPS_M_XMM : stfp    %xmm0_high, DS: stfp    %xmm0_high, DS
RET_NEAR : ld    t1, SS: ld    t1, SS
RET_NEAR : addi    rsp, rsp, 0x8
RET_NEAR : wripi    t1, 0

```



Пример кода

```
1 double sum16(double *a) {  
2     double sum = 0;  
3     for(int i = 0; i < 16; ++i) {  
4         sum += a[i];  
5     }  
6     return sum;  
7 }
```

gcc-12.1 -O3 -msse4

```
1 sum16(double*):  
2     pxor    %xmm0, %xmm0  
3     addsd   (%rdi), %xmm0  
4     addsd   8(%rdi), %xmm0  
5     addsd   16(%rdi), %xmm0  
6     addsd   24(%rdi), %xmm0  
7     addsd   32(%rdi), %xmm0  
8     addsd   40(%rdi), %xmm0  
9     addsd   48(%rdi), %xmm0  
10    addsd   56(%rdi), %xmm0  
11    addsd   64(%rdi), %xmm0  
12    addsd   72(%rdi), %xmm0  
13    addsd   80(%rdi), %xmm0  
14    addsd   88(%rdi), %xmm0  
15    addsd   96(%rdi), %xmm0  
16    addsd   104(%rdi), %xmm0  
17    addsd   112(%rdi), %xmm0  
18    addsd   120(%rdi), %xmm0  
19    ret
```

gcc-12.1 -O3 -msse4 -ffast-math

```
1 sum16(double*):  
2     movupd  32(%rdi), %xmm1  
3     movupd  48(%rdi), %xmm3  
4     movupd  (%rdi), %xmm0  
5     movupd  16(%rdi), %xmm4  
6     addpd   %xmm3, %xmm1  
7     movupd  80(%rdi), %xmm5  
8     movupd  96(%rdi), %xmm2  
9     addpd   %xmm4, %xmm0  
10    addpd   %xmm0, %xmm1  
11    movupd  64(%rdi), %xmm0  
12    addpd   %xmm5, %xmm0  
13    addpd   %xmm0, %xmm1  
14    movupd  112(%rdi), %xmm0  
15    addpd   %xmm2, %xmm0  
16    addpd   %xmm0, %xmm1  
17    movapd  %xmm1, %xmm0  
18    unpckhpd %xmm1, %xmm0  
19    addpd   %xmm1, %xmm0  
20    ret
```


25 ticks

VS

18 ticks

[illegible]

MOVUPD_XMM_M	:ldfp	%xmm1_low, DS:ldfp	%xmm1_low, DS
MOVUPD_XMM_M	:ldfp	%xmm1_high, DS:ldfp	%xmm1_high, DS
MOVUPD_XMM_M	:ldfp	%xmm3_low, DS:ldfp	%xmm3_low, DS
MOVUPD_XMM_M	:ldfp	%xmm3_high, DS:ldfp	%xmm3_high, DS
MOVUPD_XMM_M	:ldfp	%xmm0_low, DS:ldfp	%xmm0_low, DS
MOVUPD_XMM_M	:ldfp	%xmm0_high, DS:ldfp	%xmm0_high, DS
MOVUPD_XMM_M	:ldfp	%xmm4_low, DS:ldfp	%xmm4_low, DS
MOVUPD_XMM_M	:ldfp	%xmm4_high, DS:ldfp	%xmm4_high, DS
ADDPD_XMM_XMM	:maddf	%xmm1_low, %xmm1_low, %xmm3_low	
ADDPD_XMM_XMM	:maddf	%xmm1_high, %xmm1_high, %xmm3_high	
MOVUPD_XMM_M	:ldfp	%xmm5_low, DS:ldfp	%xmm5_low, DS
MOVUPD_XMM_M	:ldfp	%xmm5_high, DS:ldfp	%xmm5_high, DS
MOVUPD_XMM_M	:ldfp	%xmm2_low, DS:ldfp	%xmm2_low, DS
MOVUPD_XMM_M	:ldfp	%xmm2_high, DS:ldfp	%xmm2_high, DS
ADDPD_XMM_XMM	:maddf	%xmm0_low, %xmm0_low, %xmm4_low	
ADDPD_XMM_XMM	:maddf	%xmm0_high, %xmm0_high, %xmm4_high	
ADDPD_XMM_XMM	:maddf	%xmm1_low, %xmm1_low, %xmm0_low	
ADDPD_XMM_XMM	:maddf	%xmm1_high, %xmm1_high, %xmm0_high	
MOVUPD_XMM_M	:ldfp	%xmm0_low, DS:ldfp	%xmm0_low, DS
MOVUPD_XMM_M	:ldfp	%xmm0_high, DS:ldfp	%xmm0_high, DS
ADDPD_XMM_XMM	:maddf	%xmm0_low, %xmm0_low, %xmm5_low	
ADDPD_XMM_XMM	:maddf	%xmm0_high, %xmm0_high, %xmm5_high	
ADDPD_XMM_XMM	:maddf	%xmm1_low, %xmm1_low, %xmm0_low	
ADDPD_XMM_XMM	:maddf	%xmm1_high, %xmm1_high, %xmm0_high	
MOVUPD_XMM_M	:ldfp	%xmm0_low, DS:ldfp	%xmm0_low, DS
MOVUPD_XMM_M	:ldfp	%xmm0_high, DS:ldfp	%xmm0_high, DS
ADDPD_XMM_XMM	:maddf	%xmm0_low, %xmm0_low, %xmm2_low	
ADDPD_XMM_XMM	:maddf	%xmm0_high, %xmm0_high, %xmm2_high	
ADDPD_XMM_XMM	:maddf	%xmm1_low, %xmm1_low, %xmm0_low	
ADDPD_XMM_XMM	:maddf	%xmm1_high, %xmm1_high, %xmm0_high	
MOVAPD_XMM_XMM	:movfvp	%xmm0_low, %xmm1_low	
MOVAPD_XMM_XMM	:movfvp	%xmm0_high, %xmm1_high	
UNPCKHPD_XMM_XMM	:movfvp	%xmm0_low, %xmm0_high	
UNPCKHPD_XMM_XMM	:movfvp	%xmm0_high, %xmm1_high	
ADDPD_XMM_XMM	:maddf	%xmm0_low, %xmm0_low, %xmm1_low	
ADDPD_XMM_XMM	:maddf	%xmm0_high, %xmm0_high, %xmm1_high	
RET_NEAR	:ld	ti, SS:ld	ti, SS
RET_NEAR	:addl	rsp, rsp, 0x8	
RET_NEAR	:wripi	ti, 0	
OR_R_R	:or	rcx, rcx, rcx	
MOVSD_M_XMM	:stfvp	%xmm0_low, DS:stfvp	%xmm0_low, DS

F	Dc	Rn	1	Is	Cm	1	Mc	1					
F	Dc	Rn	1	Is	Cm	1	2	Mc	1				
F	Dc	Rn	1	Is	Cm	1	2	Mc	1				
F	Dc	Rn	1	Is	Cm	1	2	Mc	1				
F	Dc	Rn	1	Ds	Is	Cm	1	Mc	1				
F	Dc	Rn	1	Ds	Is	Cm	1	2	Mc	1			
F	Dc	Rn	1	Ds	Is	Cm	1	2	Mc	1			
F	Dc	Rn	1	Ds	Is	Cm	1	2	Mc	1			
F	Dc	Rn	1	Ds	1	2	Is	Cm	1				
F	Dc	Rn	1	Ds	1	2	Is	Cm	1				
F	Dc	Rn	1	Ds	Is	Cm	1	Mc	1				
F	Dc	Rn	1	Ds	Is	Cm	1	2	Mc	1			
F	Dc	Rn	1	Ds	Is	Cm	1	2	Mc	1			
F	Dc	Rn	1	Ds	Is	Cm	1	2	Mc	1			
F	Dc	Rn	1	Ds	1	2	3	Is	Cm	1			
F	Dc	Rn	1	Ds	1	2	3	Is	Cm	1			
F	Dc	Rn	1	Ds	1	2	3	Is	Cm	1			
F	Dc	Rn	1	Ds	Is	Cm	1	2	Mc	1			
F	Dc	Rn	1	Ds	1	2	3	Is	Cm	1			
F	Dc	Rn	1	Ds	1	2	3	4	Is	Cm	1		
F	Dc	Rn	1	Ds	1	2	3	4	Is	Cm	1		
F	Dc	Rn	1	Ds	1	2	3	4	5	Is	Cm	1	
F	Dc	Rn	1	Is	Cm	1	2	Mc	1	2	3		
F	Dc	Rn	1	Is	Cm	1	2	Mc	1	2	3		
F	Dc	Rn	1	Ds	1	2	3	Is	Cm	1	2		
F	Dc	Rn	1	Ds	1	2	3	Is	Cm	1	2		
F	Dc	Rn	1	Ds	1	2	3	Is	Cm	1			
F	Dc	Rn	1	Ds	1	2	3	Is	Cm	1			
F	Dc	Rn	1	Ds	1	2	3	4	Is	Cm	1		
F	Dc	Rn	1	Ds	1	2	3	4	5	6	Is	Cm	1
F	Dc	Rn	1	Ds	1	2	3	4	5	6	Is	Cm	1
F	Dc	Rn	1	Is	Cm	1	Mc	1	2	3	4	5	
F	Dc	Rn	1	Is	Cm	1	2	3	4	5	6	7	
F	Dc	Rn	1	Ds	1	2	Is	Cm	1	2	3	4	
F	Dc	Rn	1	Is	Cm	1	2	3	4	5			
F	Dc	Rn	1	Ds	1	2	3	4	Is	Mc			



Другие инструменты

1. `-O3 -mtune={native,sse,avx,neon,...}` - автовекторизация
2. `#pragma simd`, `#pragma unroll` – авто конвейеризация
3. `Intrinsic` – векторные типы и операции
4. `if(__builtin_expect(..))` – подсказки ветвления
5. `restrict`, `const`, etc...
6. PGO, LTO
7. OpenMP – удобная многопоточность
8. Готовые оптимизированные библиотеки/продукты

Другие инструменты

- Профилировщики
 - Intel VTune, ARM MAP, AMD μ Prof,
 - Perf, OProfile, valgrind, MSVC, TAU, Caliper, HPCToolkit, vampire,...
 - **printf**, rdtsc, и т.п.
- Compiler explorer ([Godbolt.org](https://godbolt.org))
- [GEM5](#) + [Konata](#)

Ещё почитать/посмотреть

- Intel® 64 and IA-32 Architectures Software Developer's Manual
- Agner Fog's блог по оптимизации
- IT Hare: operation costs in CPU
- Prof. Dr. Ben H. Juurlink on Youtube

Подытожим

- 1. Эффективный код = много параллельности в операциях**
- 2. Надо переиспользовать данные**

Thank you.

把数字世界带入每个人、每个家庭、
每个组织，构建万物互联的智能世界。

Bring digital to every person, home and
organization for a fully connected,
intelligent world.

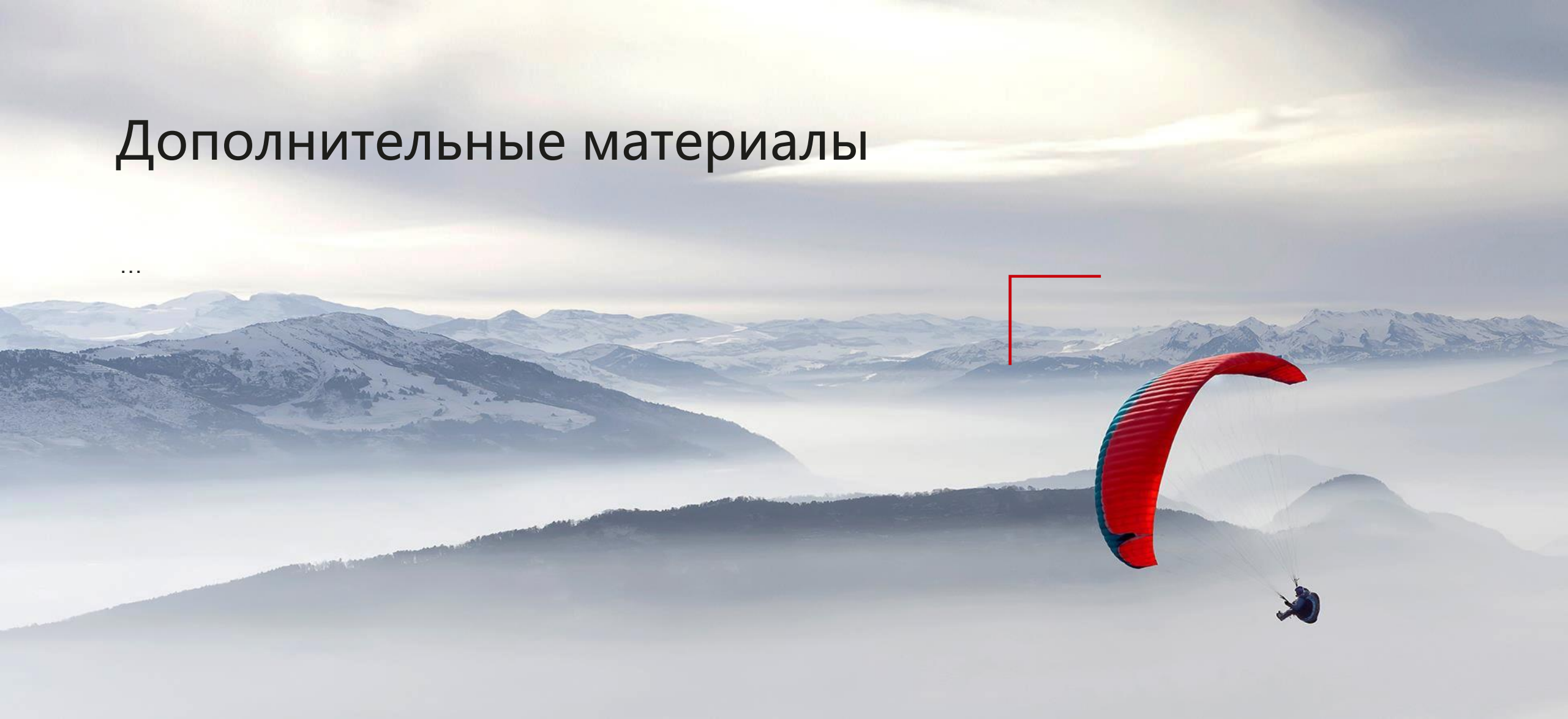
**Copyright©2020 Huawei Technologies Co., Ltd.
All Rights Reserved.**

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.

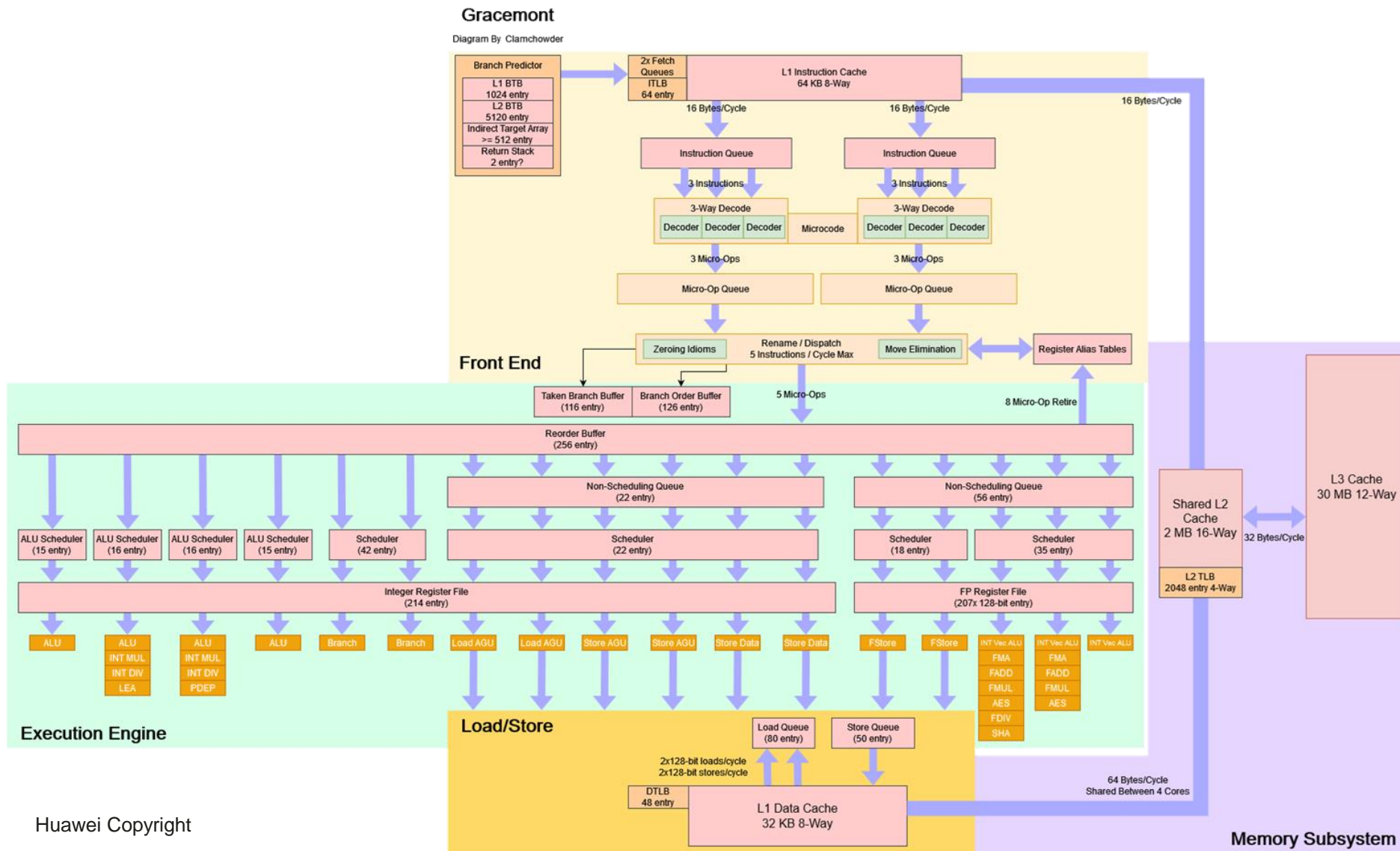


Дополнительные материалы

...

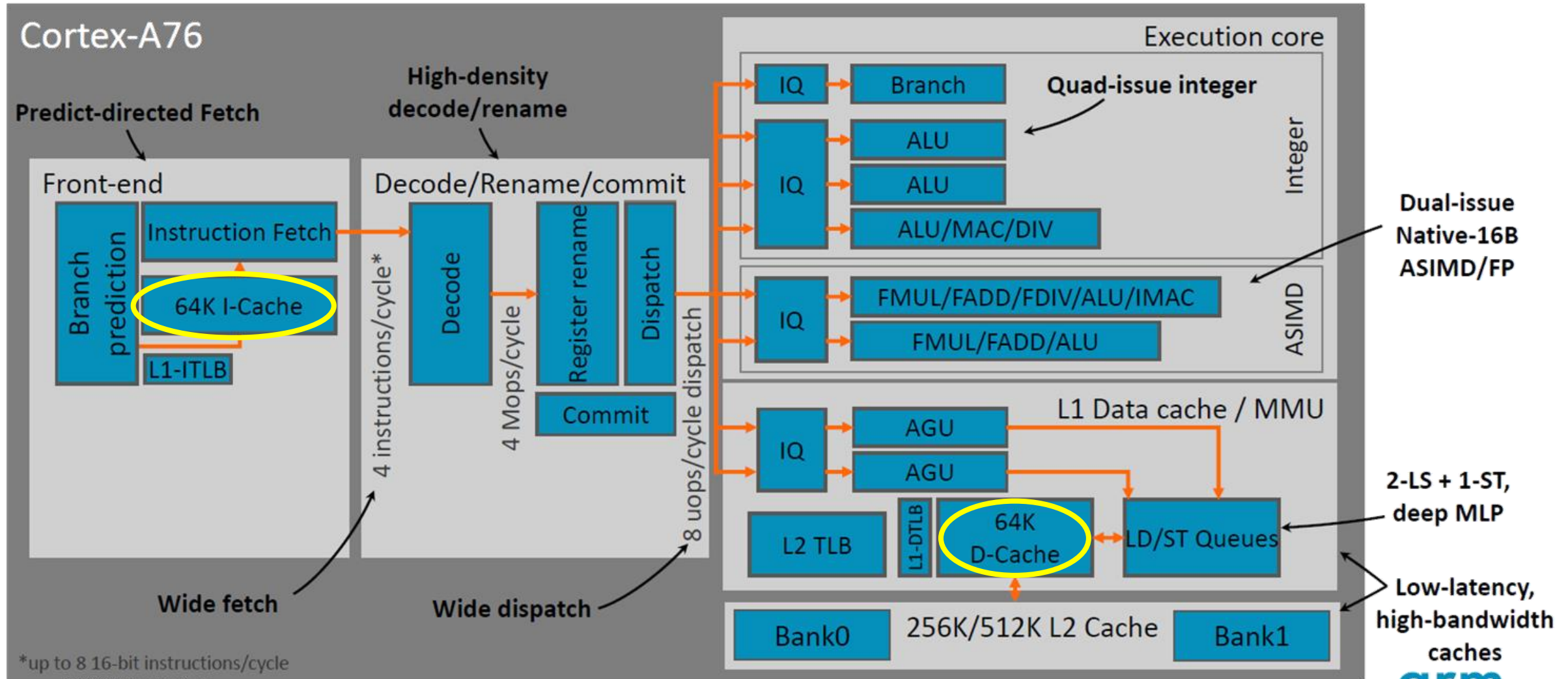


Gracemont core – экономное ядро





ARM: кэш на схеме



Пример кода AVX512

```
1 void scale16(double alpha, double *a) {  
2     for(int i = 0; i < 16; ++i) {  
3         a[i] *= alpha;  
4     }  
5 }
```

```
1 scale16(double, double*):  
2     vbroadcastsd    %xmm0, %zmm0  
3     vmulpd    (%rdi), %zmm0, %zmm1  
4     vmulpd    64(%rdi), %zmm0, %zmm0  
5     vmovupd    %zmm1, (%rdi)  
6     vmovupd    %zmm0, 64(%rdi)  
7     vzeroupper  
8     ret
```

gcc-12.1 -O3 -mavx512f