

Числа, преобразование:

--	--	--	--	--	--	--

Норм. числа

Знак 64 32 16 8 4 2 1

--	--	--	--	--	--	--

Операт. числа

Как изображать число?

2) Другая система:

128 64 32 16 8 4 2 1

--	--	--	--	--	--	--

-128 или 127

$$(a+128) + (b+128) = (a+b+128) + 128 \leftarrow \text{нужно брать}$$

3) Двоичный дополнительный
(основание
числа)

-128	64	32	16	8	4	2	1

$$\begin{array}{r} -5 : 111111011 \\ + 9 : 000001001 \\ \hline 4 000000100 \end{array}$$

6) Форма с переносами: расширенная

0000	(0)	0
0001	(1)	-1
0010	(2)	1
0011	(3)	-2
0100	(4)	2
0101	(5)	-3
⋮	⋮	⋮

Дробные числа:

$$1) \frac{1}{4} \frac{1}{2} \frac{1}{4} \frac{1}{8} \dots$$

$$\dots \frac{1}{2^2} \frac{1}{2^1} \frac{1}{2^0} \frac{1}{2^{-1}} \frac{1}{2^{-2}} \frac{1}{2^{-3}} \dots$$

$$0,625 (-2)$$

$$\begin{array}{r} 1 | 0,225 \\ 0 | 0,500 \\ \downarrow 1 \end{array}$$

$$0,625 = 0,101$$

0	1
0	2
0	4
0	8
1	6
1	2
0	4

переиз

$$3,1_{10} = 0011,0(0011)_2$$

из 10 в 2 получаю либо
континуальные или перIOD. дроби
(или одна нечтная.)

Как изображать число?

2) Другая система:

128 64 32 16 8 4 2 1

--	--	--	--	--	--	--

-128 или 127

$$(a+128) + (b+128) = (a+b+128) + 128 \leftarrow \text{нужно брать}$$

(так идет-то)

-127	64	32	16	8	4	2	1

6) Система счисления с отриц. основанием

-128	64	32	16	8	4	2	1

$$(-2^0, -2^1, -2^2, -2^3 \dots)$$

7) Симметричные системы счисления

27	9	3	1
0	1	7	7

Минимальное значение

$$\begin{pmatrix} 2 \\ 0 \\ 1 \\ -1 \\ 0 \\ 1 \end{pmatrix}$$

Перевод FTZ / DAT

FTZ генерирует 0 (нуль)

DAT тоже.

2) exp = 1111 a) +int
-int - дескременировать

5) QNaN SNaN (нормализация
сигнала 1 г.)
↓
нормализация
ненулевое
значение

$$0/0 = NaN$$

$$+\infty - \infty = NaN$$

if ($a \neq a$) проверка на NaN 3.
~~float in [N]~~ Анонимно

float sum = 0, c = 0

for (int i = 0; i < N; i++) {

float y = in[i] - e;

float t = sum + y;

c = (t - sum);

sum = t;

Габариты в м.квадр.

if (fabs(a - b) < eps), где $\text{eps} = \epsilon$

float

10869 32 16 8 4 2 1

00000110

1111111

100000001

сменить

10/24

21.03.2013.

Создание собственного float

107,37 → float. → 2*42D6BD71

107|1

53|1

26|0

13|1

6|0

3|1

нормализация

1

1101011.01011101011000
010100

$\cdot 2^6$

$b_{(-127)} = 0 10000101$

0|37

0|74

1|48

0|96

1|92

1|84

1|68

1|36

0|72

1|44

0|88

1|76

1|52

0|84

0|08

0|16

0|37

$$0,125 = 0,001 \cdot 4 = 1 \cdot 2^{-3} = 0$$

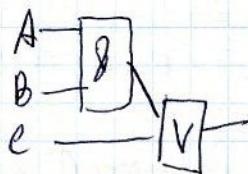
$0,250$
 $0,000$
 $1,00$

$(-3) + 127$

$$-3 = 124 =$$

$$(-3) + 127.$$

Mögliches memora
 $(A \text{ OR } (B \text{ AND } C))$

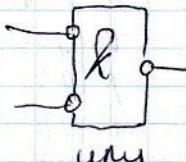
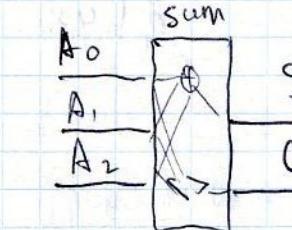


$\rightarrow D_0 - \text{ne}$

$\rightarrow D - u$

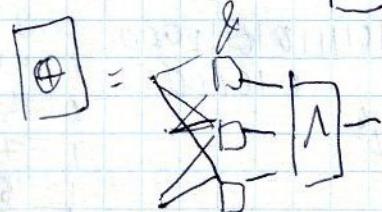


NAND



$$1+1=10$$

sc



Zeitvektor:

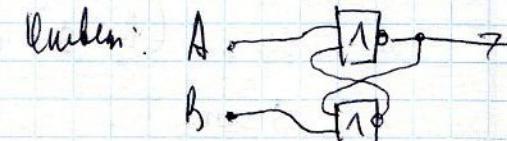
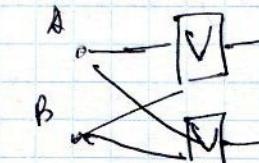
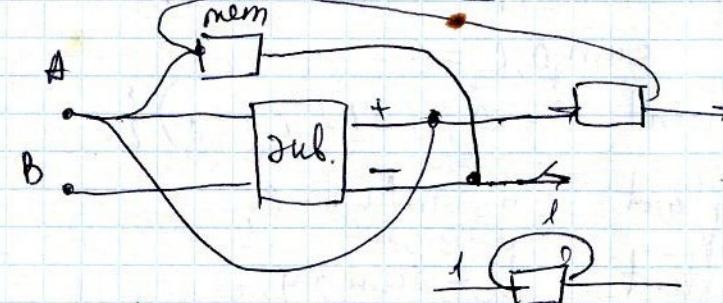
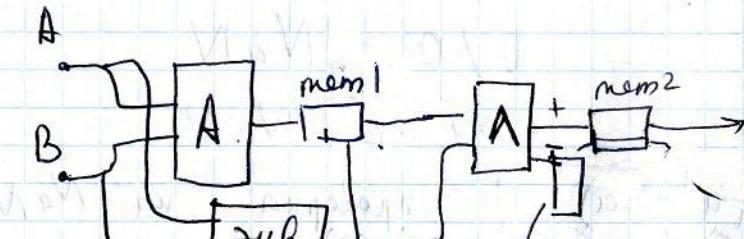
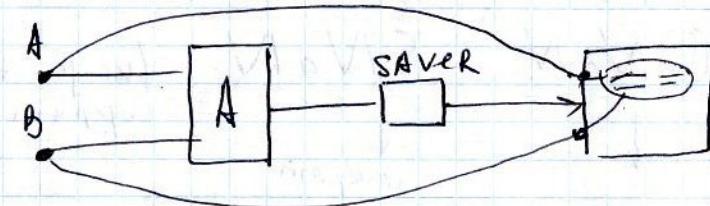
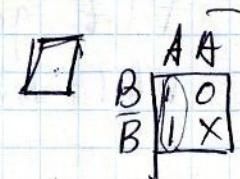
A B Q

0 0 corp. neg.

0 1 0
1 0 1
1 1 1

0 0 negativ 1

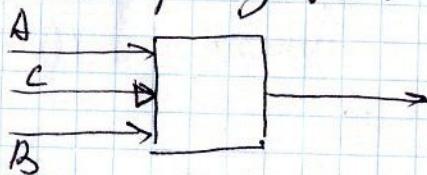
RS-triggen



$$\bar{C} \quad t = \frac{1}{V}, \quad S = \frac{C}{J}$$

Сигнал не приходит синхронизацию на
тревож. (10см / 1 такт нап-ра)

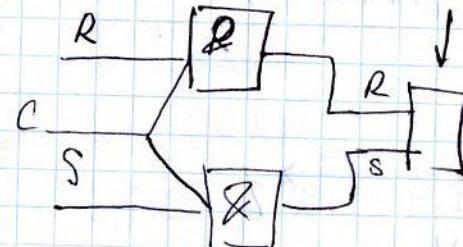
Синхронизация:



C - синхр. на борту

тревож.

Синхр:



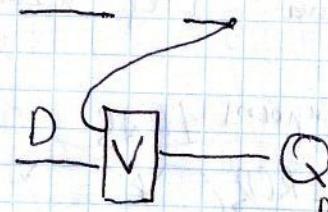
Алтерн.: (C)D - тревож., (C) - синхр.

C D | Q

1 0 | 0

1 1 | 1

0 X | сорп. нр.



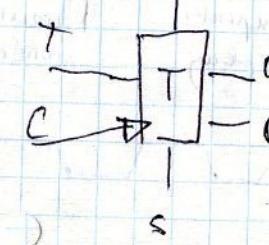
(C)T тревож

C T | Q

1 0 | сорп

1 1 | инверс.

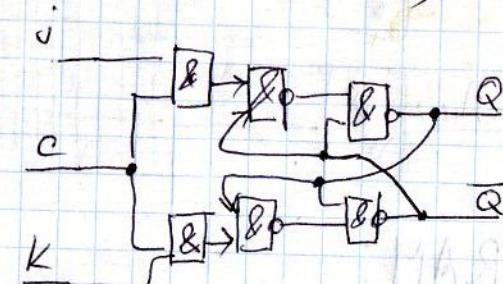
0 X | сорп.



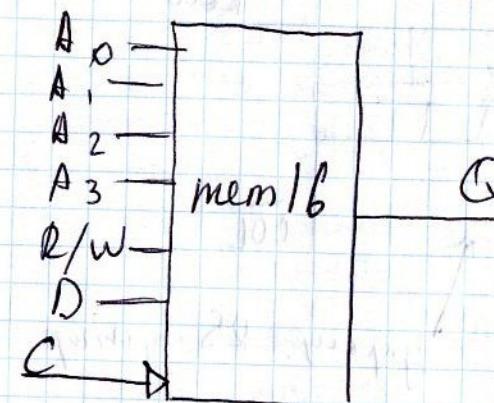
JK - тревож

J	K	Q
0	0	сорп
0	1	0
1	0	1
1	1	инв.

(продолжить)



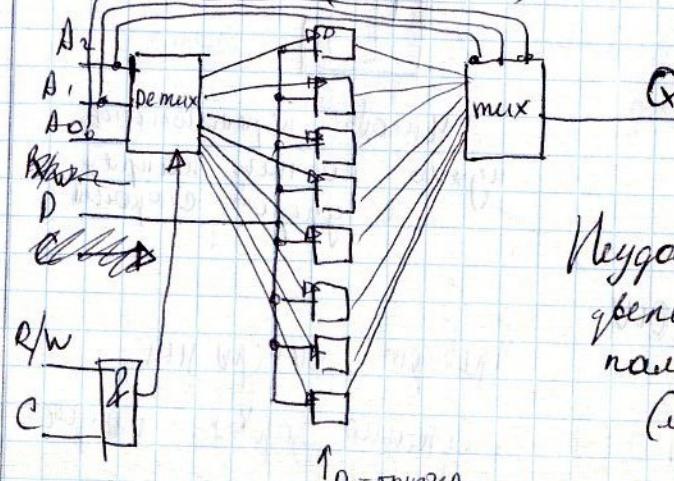
Логика



(дат. 3г)

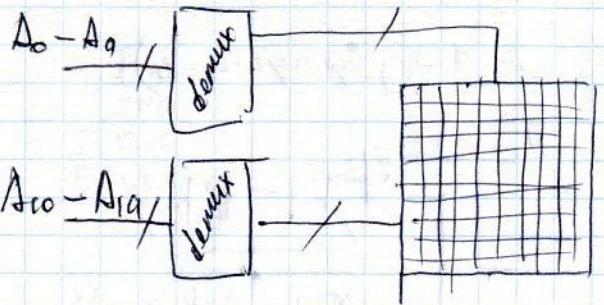
28.09.2013

Решение (mem 8)

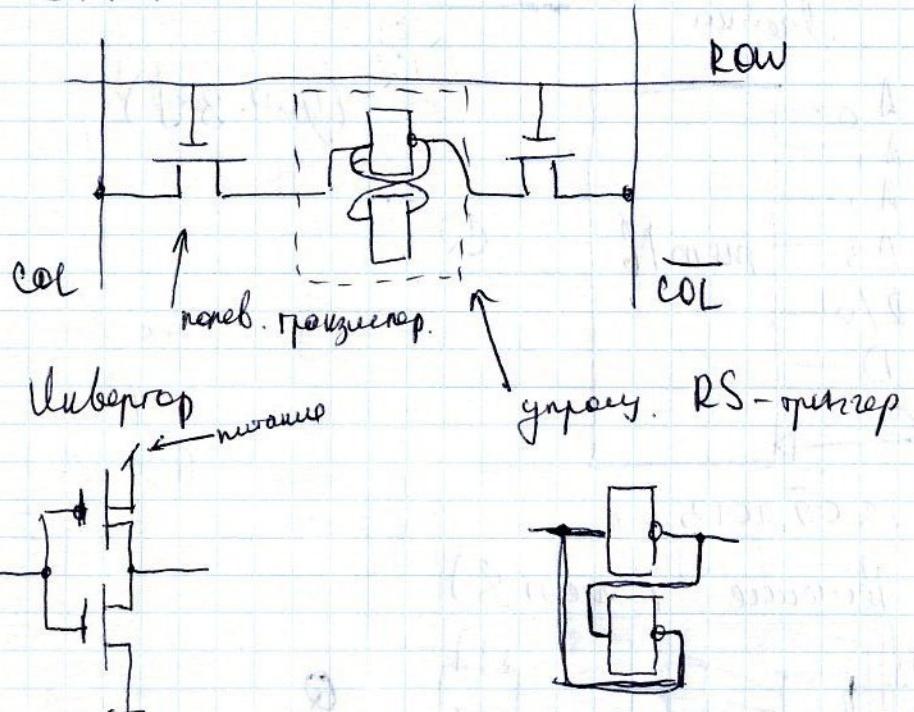


Недостаток, если
запись и
чтение
параллельно

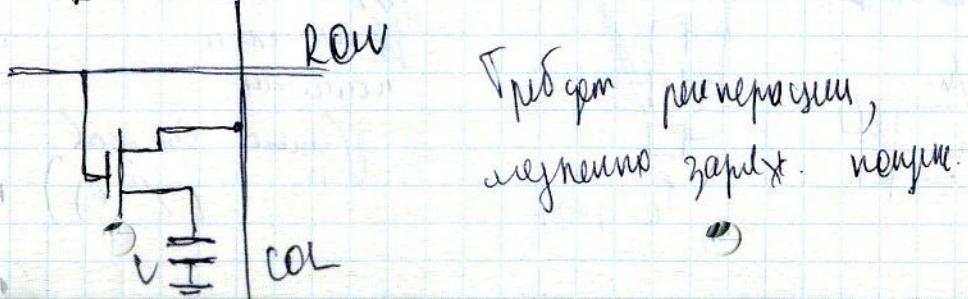
(использовать
биты вспомог.
(2^10 +))



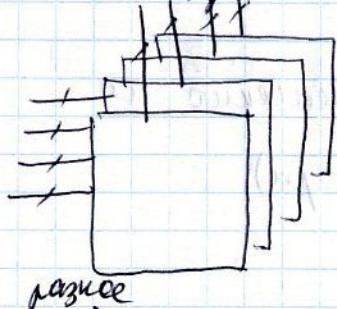
SRAM:



DRAM:



Уменьшение



многослойное наение

- машинное исполнение.

объем

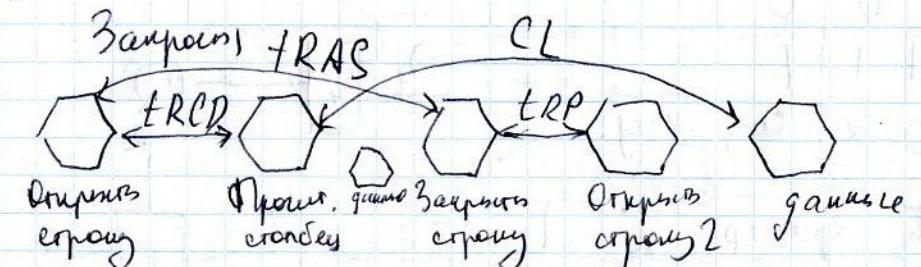
сроки разработки

от начинки до пуска
(ст. поиска)

Запом. Мята

Каждый из 2 ядер имеет 200 производственных операций, которые включают в себя 1000 единиц оборудования, увеличивающиеся в 2 раза

Запом. + RAS



Синхронизация



CL : CAS# latency

tRCD: RAS# row access strobe

CAS#

tRP: CAS# Precharge

tRAS: CAS# Precharge delay

1) I упрощение цикла:

FPM DRAM (Fast Page mode)

Если запрос на одну строку, то
она не зарабатывает

2) II упрощение DRAM

EDO DRAM (Extended Data output)

Синхрон с помощью буфера настр
состоит во время tRAS (просим)

3) III упрощение DRAM

BEDO DRAM (Burst Ext. Data out.)

Буфер состоит из $n + (n+1) + (n+2) + (n+3)$
от одной строки (у каждого из них
использовано время)

4) IV упрощение

SDRAM (Synchronization)

Действует синхронизацию отдельно
от контроллера и памяти

5) DDR (Double Data Rate)

+ битность увеличена

Pause между строк. времена-
напряжение прошлют
DDR двух. вспомогательные
на подложке и на чипе
чипа (чип и чип) - зазоры
за Raum.

За время убывания разряда
может начаться генерация
на убе растяг (первая половинка
на чипе, вторая откладывается
на контактах)

6) DDR2

Убраны разрядные лаги

Убраны общий recovery отработки

7) DDR3

Все так же как DDR2

Пример:

DDR3

1600 MHz
(800-пин.)

9-8-8-23-CR1

CL-tRCD-tRP-tRAS - ?

5.10.13

GDDR3 .. GDDR5

(DDR2)

(DDR3)

G-graphic (где граф. адаптеров)

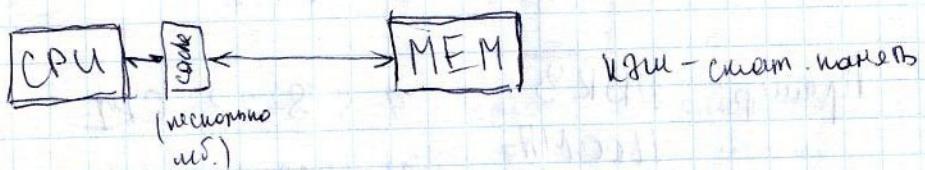
GDDR оптимизирована на высокую со-
 временность пропускания (100, 200 нс/нс) высокопро-
 токольную), но не может отключать.

В PSLI GDDR5 имеет более памяти.
(Быстро 100-200 нс/нс Atom (np.), не сильно
уменьшить GDDR5 память)

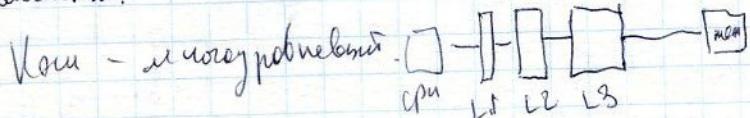
LPDDR3 Low-palette где меньших
устройств. (анализации кратчайших путь
изменение энергопотребления)

ECCDDR - мерид. памяти на избыточно
вероятности ошибок. Добавление новой кор-
рекции, исправляющей ошибки ошибок.

Кэш-память



Кэш заменяет меньшее время времени доступа
переносами.



(VRAM L4 - каш. каш-пам.)

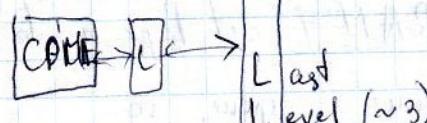
Регистры памяти каш-пам.

R: 1. look through
2. look aside
(through + CPU → mem)

W: 1. write through
CPU → k → mem
2. write back
(не пишет в mem, новая ком не передана)



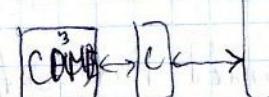
Соответствует!



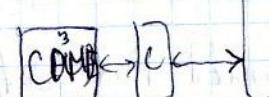
32kB cache L1 16 ячеек



256kB cache L2 128 ячеек



8MB cache L3 30 ячеек



MEM 100-200 Гбайт.

Кэш с новыми ассоциативностью, - не acc.

Принцип обратного.

- acc.
acc.

Ассоциативность кэша.

L1 4
L2 8
L3 16

(10, 8, 4 ячейки доступа к памяти
новые ones.)

8 ячейк обработка

Проблема непримитивная кэша (нужен зеркальный
а, потому нет б. оверх., но он в
памяти все (write-back))

1. Конфликтность с памятью. Диспетчинг. —
один изображенный вами изображающие
согласия имена

2. Протокол конфликтности, если не было
имен.

1) MSI — находит одинаковую & 3x
координату
modified shared invalid

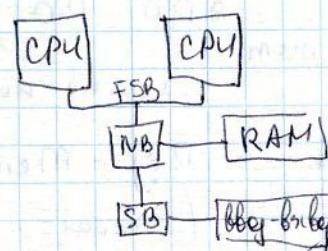
Координаты
invalid — некая
shared — хранит некую MEM
modified — то, что не имеет go MEM

Результат если память & мод. — право
имеет
если & shared — все операции вспомогательны —
некий протокол, некие общие соглашения
modified, а & с вспомогательной — invalid

Если & invalid

2) MESI — MSI + Exclusive — shared,
no equivalent.
Exclusive → modified
modified + write → exclusive
(shared из-
за того, что они
не могут)

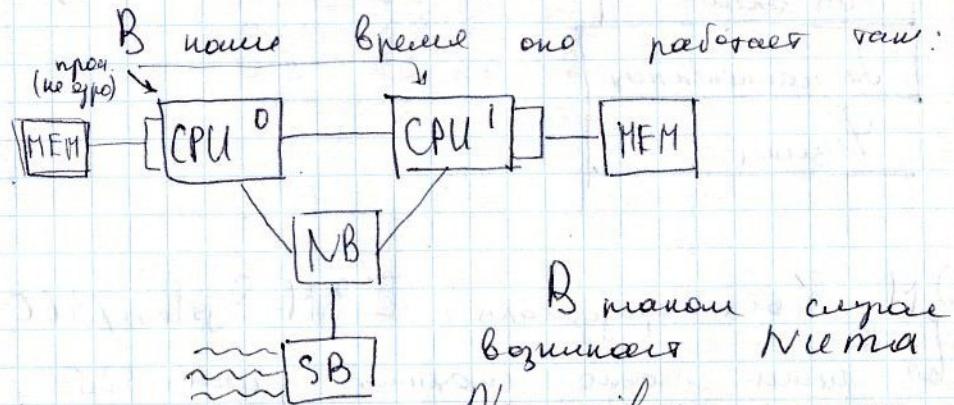
Pause



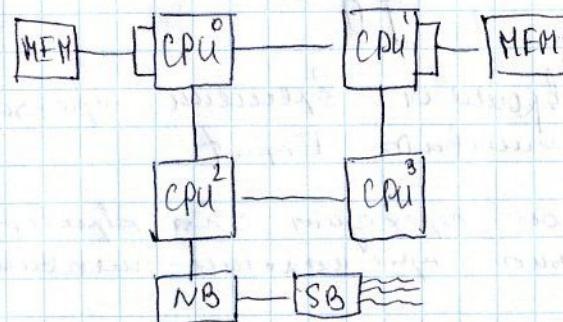
Использование
Frontside
Bus
(Frontside
Bus)

Чипсет — это набор
вспомогательных чипов
процессора

NB — North Bridge за отображение
SB — South Bridge за управление.

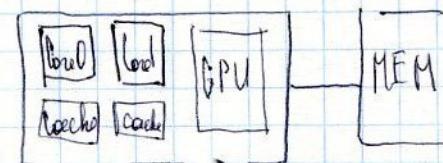


Важное
вопросы
Numba
Non-uniform memory access.
Две разные метки
использованием оп-м
для разных операций
или ячеек.



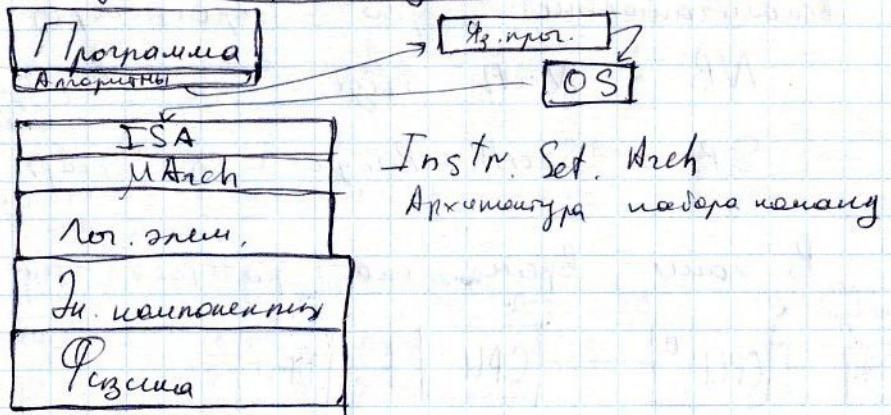
[NB] берет
на модуль CPU

(Processor = ядро + контроллер памяти + кэш + ...)



ISA / MA архитектура (многосервисная.)

12.10.13



IBM б 60х предупредил IBM System/360

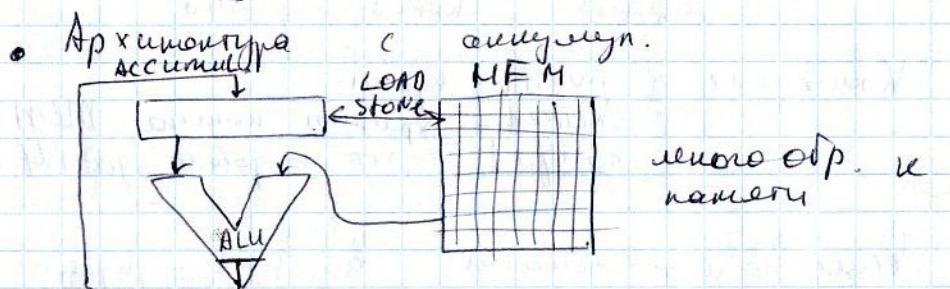
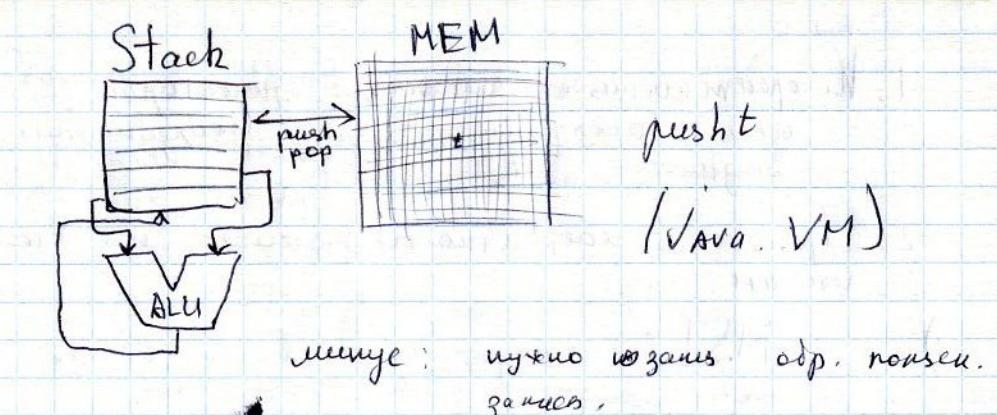
До 25. меса можно създади операт 325
архитектур за 1 мес и не 2мес 165
архит. за 2 меса.

Способи взаим. с IO

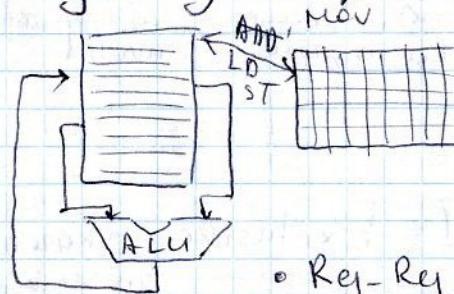
1. Polling : прос. брауз от времени прерывает биоса. < 100ms ограничение Input

2. Прерывание: прос. переход на обработку прерывания только при настолни междуне, например.

• ISA : • основная архитектура.



• Reg - Reg approx. (ARMv7)



• Reg - Reg 3 apr:

$$\begin{aligned} \text{ADD } R2 & R2 R3 \\ R2 &= R2 + R3 \end{aligned}$$

• Reg - Reg 2 apr: ADD R2 + = R3

$$\text{MOV } R2 R1 \Leftrightarrow R2 := R1$$

• Reg - Mem ADD R0 R1 - Reg - Reg 2
|| ADD R0 t - Reg Mem
(x86,) →
Reg - Reg + mem
если память (не более
одной операции)

• Mem - Mem = Reg - Mem + mem
(если не более одной операции)
можно использовать
generators

Байт - это минимально адресуемая единица памяти.

Память ~ байт (8 бит)

1 KB = 1024 байт

1 KB/c ^{но} = 1000 байт/c, memory rate

Memory (байты/c)

Примеры ISA

$$z = (a + b \cdot c) / (c + d \cdot c - e)$$

Stack	Acc	Reg-Mem
4 registers	1	6
push/pop	LD/ST	MOV

Минимум: ADD, SUB, MUL, DIV

push B	load B
push C	mul C
mul	add a
push a	store x
add, pop Y	load d
push d	mul c
push c	add a
push e	sub e
push a	store y
add	Load x
push e	div y
sub, pop X	store z
push Y, push x	
div	
pop Z	

R-R3

```

LD R0 a
LD R1 b
LD R2 c
MUL R1 R1 R2
ADD R1 R1 R0
LD R2 d
LD R3 e
MUL R2 D2/R3
ADD R2 R0 R2
SUB R2 R2 R3
DIV R0 R0 R2
ST Z R0

```

| R-R3 MUL LD| R2 S a
 LD R1 b
 LD R2 c
 MUL R1 R2
 ADD R2 R1
 LD D R1
 LD C R2
 LD E R3
 MUL R1 R2
 SUB R3 R1
 LD A R1
 ADD R6 R3
 DIV R0 R6
 ST Z R0

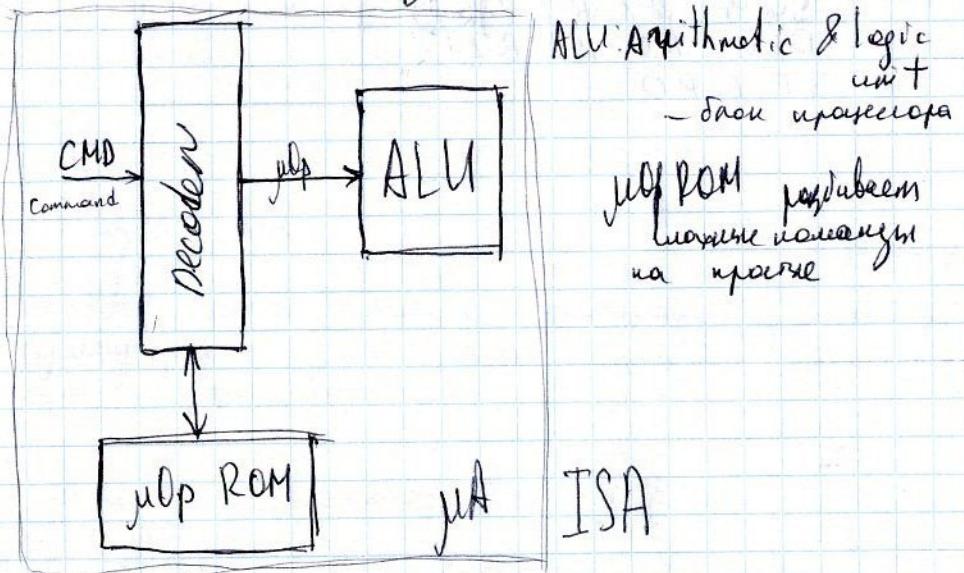
19.10.13.

CISK/RISK (μ A)

Conflict instruction set computer - x86

Reduced instruction set computer - ARM

lysus в уменьшении производительности
надежности наряду с уменьшением размера



- CISK - архитектура определена ADD [reg + Reg + 4 + off], reg (Reg-Reg)
- RISK - простая, но неудобная архитектура, основанная в концепции μ Op. (Reg-Reg)

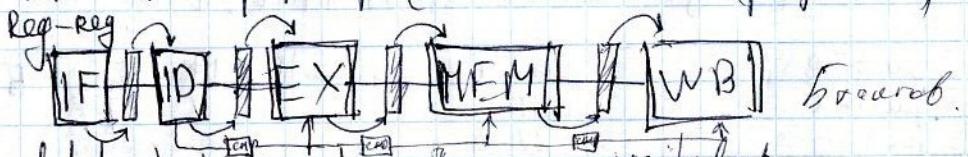
- В x86 группа команды 1..15 байт [инструкция регистра назов]
- CISK - мак. группа команды назов

- В RISK - одна группа 4 байт

// Рассмотрим прошуше оптимизацию

Profile Guided Optimization - PGO

PGO (премиум) подразумевает на этапах быстрое, неоптимизированное выполнение, которое включает в себя генерацию, компиляцию и тестирование различных версий программы на производительность (не уверен?). (Быстро, f, разные версии). (Чем меньше времени требуется на генерацию, тем лучше). (Чем меньше времени требуется на генерацию, тем лучше)

Pipeline - последовательная архитектураMIPS - микропроцессор (PS1 PS2 (playstation))

inst. fetch decode execute mem writeback

1. Извлечение из ячейки
2. Декодирование
3. Выполнение
4. Возврат к памяти

Решение Pipeline - когда одна команда на 2 этапе, но 1 этапе. заблок. работе.

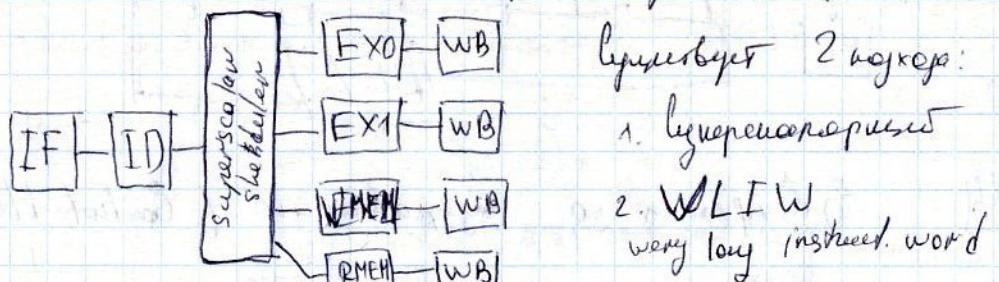
- latency 1, + промеж. час. 1.

Выполненные буферы - фиктивный

9.11.13

Нужно сделать суперкор? SpeedUP!

Ненужные нен-бы ALU / Рассмотрим



VLIW - процессор одновременно выполняет many наряды, побуждая не параллельно, но последовательно.

VLIW можно считать суперкором 4

Суперкоромархитектура - в scheduler заменяется на менеджеромархитектурой. Superscalar: in-order / Out-of-order
in-order корпоративный наряды не ордера
out-of-order вырабатывает из своих нарядов не корпоративные наряды для выполнения (width range ~ 60)

VLIW: +: просто сгенерировано

-: требует дополнительных временных гнезд параллельности
-: при выдаче загружает наряд не один (задача None), а два блоков - параллельно.

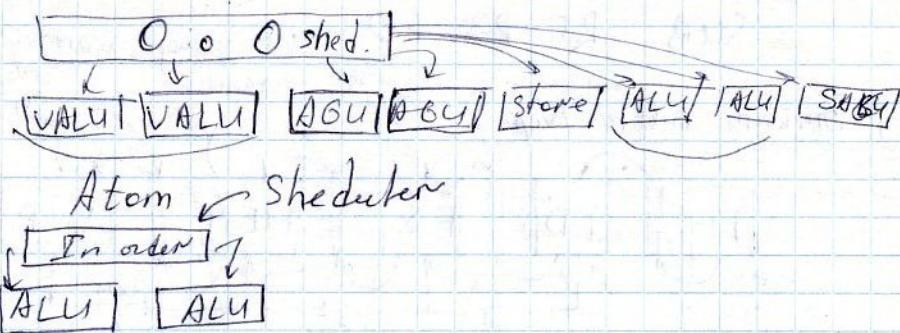
SuperScalar: +: есть вспомогательное управление
+: засоряет память, это несет в себе, слева наряды из своих параллельных +: не нужны переключения. Всегда есть один идентичный архитектур.

VLIW: itanium (intel), and scalar (AMD, nVidia)

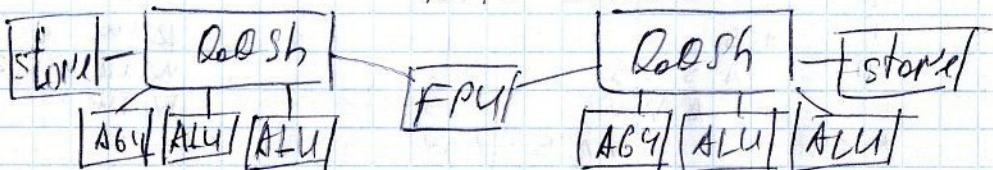
Superscalar: In-order: atom
Out-of-order: x86, ARM

Современная суперкоромархитектура

Haswell II (небольшой Intel)



AMD Bulldozer



2-core - AMD ~ 1-core - Intel

16.11.13 Многопоточная архитектура

В同一 время в индустрии появляются ядерные ядра с параллельным управлением нарядами speed = clock × IPC instruction per clock.

Хороший пример - Pentium 4 (высокий разъем, параллельные ядра, IPC высокий - около 42)

P4 - pentium, superscalar, ядерный

PPro - pentium pro, superscalar OoO → P2 →

→ P3 → P4 - асинхронно между ядерами (ядерный) → PM - где каждое ядро (ядерный) работает отдельно.

(Pentium II билинг)

Когда дин. приложение, intel генерирует PM

Core 2. Потом свою intel генерирует синхронизацию ядер, потому что это ходоподчинение они не делают.

IPC увеличивается. Внужден в Synchronization > 3 ядерных, будет больше потрачено и на 3-4 ядра. управление в виде - регионов.

Тогда регионы генерят многоядерное приложение. это "пресекор блокирование" для каждого ядерного блока. блок-блока, блок-блока и приводят к заблуждению.

Процессор - ядро + подсистема (mem, GPU)
+ процесор - генератор процессора ядерные
У каждого ядра свои регионы и они могут ограничиваются и определять превращение.

Они могут быть и противоположными другим ядерам, другие - могут быть (также)

Thread - генератор, пресекор" по времени ядер.

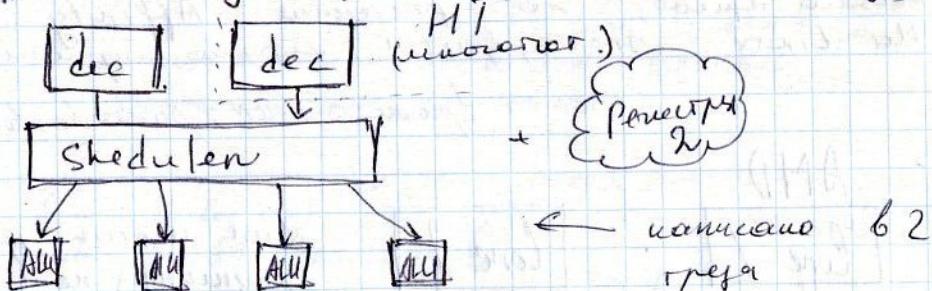
То есть, это непрерывное переключение пресекоров.

Все же генератор ядер, один из которых группой не может. (тогда один из них может, он переключает другое ядро)

Некоторое разделение на ядра - задача L1 cache.
создание ядерных архитектур, несущих, приемлемые производственные технологии, которые не являются экспериментальными.

В итоге устройство многоядерных систем ядер (такие как Standby и Main) - big.LITTLE

Hyper-Threading (Simultaneous Hyper-Threading)



Но память, которую имеет 2 ядра, не зависит друг от друга (разделение)

HT даёт выигрыши от этого до 2x раз в 6 ядерном ядре

В самом деле генераторы выигрывают. Суммарно HT даёт прирост в 20-30% производительности, и HT создаёт меньше.

Когда ядерный ядро занят задачами, то HT запускает $i_1 = 2^k$ задачи за раз. Первое ядро HT работает в Pentium II и работает параллельно.

В Core 2 не знаю HT (но у меня есть замечания), потому что Core 2 не генерирует

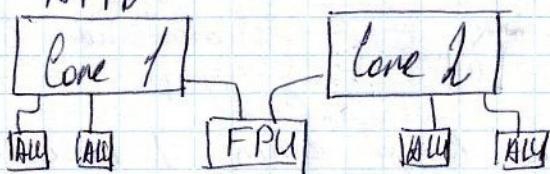
$i_1 = i_2$ из HT, $i_3 = 3$ ядра.

LinPack - надо писать на языке C с HT пользованием и т.д.

На старых Atom HT имеет и он хорошие производительности за счет того, что он In Order и минимизирует тупоть

Windows распределяет потоки на ядра
безопасично, это называется Affinity.
На Linux это аргумент, можно писать так:
(process_affinity/process_nice)

AMD:



Две ядра поделены на один FPU, но если одна ядро работает, то оно делает все операции, а другое ядро не может. Intel это делает иначе.

Для суперкомпьютеров используют SMT 16x, например. Стандартный переход по ядру, когда один ядро делает все операции в памяти. Но при этом ядро не использует языки.

Для видеокарточек очень много ядер, но дешевле, и все основные ядра, кроме FPU, делятся на ядра, которые производят различные переходы ядер. За счет этого производительность процессора высока по FLOPS.

Разница в Tesla и Geforce гравитирует.
Всё равно! И это низкоаналогично.

Open-source драйвера лучше и Nvidia
и отрабатывают их из-за монополии.

Драйверы ядерных ядер работают безопасно,
хотя это BRRATs мессенджерами.

Но в видеокартах есть более
дорогие, но их ядра работают драйверами.

Что такое Архитектура GPU

GPU хуже обобщенных CPU тем что медленнее. Например, запускать ОС она не умеет. Системная разработка доступна не реализована на ядре GPU, потому что GPU - копроцессор.

Но попытка предпринималась — PCI-E Knights by Intel. Такие чипы называются coprocessors вместо GPU, но имеют из x86 ядро (40-60 ядер), которое имеет HTLV4 (~200 threads). Работают сама система (внутри Unix). Разработка доступна есть. Предлагаемые программы тоже есть. Записи под * и + (float).

Начем float: Наличное представление числа число.

NAD (FMA) NAD означает норм + FMA в самом понятии.

Существуют такие архитектуры с теми же самыми ядрами ядер — добавленные новые ядра.

У Nvidia это warp, AMD — wavefront. Например, GeForce серия 32 ядра и выполнение одновременно. Это и то же SPMD — single program multiple data.

При условии перехода выполнимый код меняется. А вот вычисления — нет.

Еще есть — APUs, USA (от AMD)

UAS - unified Address space

(Модно заложение якого GPU RAM кот прок в пам'яті - лише по ширині, не висоті - лише післян). На функціональних GPU підтримується пам'ять, що використовується (RAM bus). На відеокартах пам'ять - це окрім прескалера буджетне оптимизоване і звичайне звичайний десктоп. процесори з UAS використовують.

Динамічні проф. — 120W — TDP ^{Thermal design power?}

GPU up to 300W (2x8+PCIe) (після 2x8)
Після 2x8 GPU використовує звичайну пам'ять.

FurMark — LinPack гру GPU. Результат можна зробити будь-який.

"Turbo" — TDP ↑

"Throttling" — Freq. ↓ of TDP ↑

Принцип Рон-Нагмана

1] Використання пам'яті зглинистого типу (супер професій 10.5)

зглинистий
зглинистий
зглинистий
зглинистий

2] Принцип програмного управління

3] Принцип агресивності (рекур. падіння)

4] Принцип однорідності пам'яті — хранення коду і даних в пам'яті

Opposite — Гарвардське

- + писати нечісно проще ніж зглинисті
- + генерація кода в більшість разів
- маємо більше пам'яті за потребу
- маємо більші можливості використання серверів чи баз даних

У Гарвардській архітектурі пам'яті, погодженої звичайної в ній розрізняється, як процесори, так і пам'яті

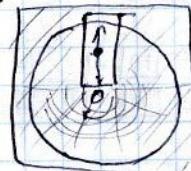
✓ 86 — Ron-Nagman's rule з тими зразами і програмами, як 2 кіла.

Так юз вже Гарвардська. MIPS, якщо відмінно.

5] Принцип пам'яті: іспользовані — погоджені, сірий — підведений.

Кошики Ганкінса

Мониторинг: одна діяльність на пам'яті. оптимізація і симетрія



Розділення на сектора пам'яті, які не мають звичайної зглинистості.

Однією з проблем є проблема зглинистості — пам'яті в пам'яті пам'яті. Є її певність — напр. суми — CRC. А є її — ECC — висока надійність пам'яті. Сумана страждає? ECC, які використовують високоякісні CRC?



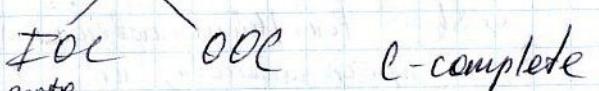
Висновок: пам'яті розподіляють зглинисті

14.12.13

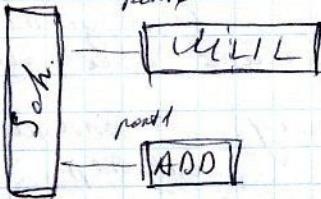
Early Writeback

В динамике может возникнуть
race condition в процессоре.

In Order: InOrderEX

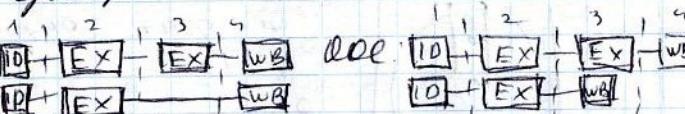


IOE:



IOE - кэширует результат работы (результат). В примере IOE синхронизирует 2 результата и очищает единовременное

IOE: MUL
ADD



Superscalar:

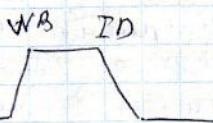
IOEX OOEEX

Тут не менее, 2го разделяем ресурсы, когда есть параллельные вычисления и обмены и памяти.

Но есть баги.

WAR и WAW hazards:

Если выполнение команды не коммутативно, а первая уже занесена в регистр, первая ее коммутация в другом регистре.



коммутация коммутации в регистре, одновременно, но занесено в регистр;

WAW x86 доступно в процессоре

Процессор не приводится по синхронизации и результату, или настолько низкое, потому что он мало.

Избавление от них - добавление
много процессоров (!параллелизм). Планаризация
всегданики про-авн. ресурс, если будет
Hazard, то переключается в другой регистр,
и т.д.

То есть программа процессоров синхронизируется, но аппаратно бывает, с гор. архитектурой, работает як.

Register Renaming

Read A - 2 раза
Write A - 2 раза Происходит не то

Обработка прерываний

Exception Zero Divide портает прог. при ошибке.
Если у нас перенаправление не обрабатывается
& → то возникает прог. прерывание.

Обмен между регистрами:

$$A = A + B$$

B = A - B Тут можно возможную перенаправление, но

A = A - B в итоге есть вероятность обрыва, и это
не является hazard прерыванием.

А там если обработка обесточит рабочие коммутации - перенаправление - ошибка в программах.



Прерыв. иск. также где нужны root/level
0..0.

Така же нужно это-то сделать от
root (один с вынесением), тогда он
вызывает прерывание и это делает 31
нено 00.

Предположим, что есть чистый компьютер,
на нем аппаратное прерывание, но
если можно обработка и отладка не
занять

Куда вернуться из прерывания?
прерыв.



Решение: прер. хранит свой блок
выхода из прерывания
и он отсылает
указанные выше
помощники процессора и
только потом за-
нимает прерывание.

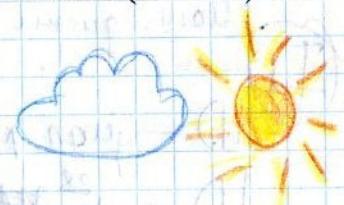
Блок процессора — помощник
Регистр — стописание
вместе с ядром



Блок

Если в один момент > n WB блоков
то n — WB блоки. Тогда можно — то можно
откладывать на память.

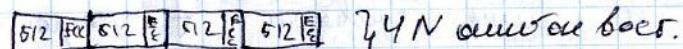
Процессор имеет более блоков
распараллеливания: Intel xBox One 16 штукерных
блоков программы, но 18 аппаратно.
Если браузерный блоков меньше 2x, то
все ок.



Back To Data

Максимальное количество итог.

Самое большое размер сектора 512
512 байт и это и самое привычно.
Помимо этого увеличивается, но
максимум больше. А ECC потому выживает.
Помимо увеличения размер сектора до ЧКВ



4096 [8N]

Разница в том, что 1 вариант восстановл.
по N ошибкам в всем секторе если весь
в 1 секторе будет ^{ЧКВ} то ничего не восстано-
вится, а то 2 варианта все ок.

Но ЧКВ-данные в это время привыкают
512-байтами. Это
помогает при работе с файлами, но
не секторами секторами, которые состоят из секторов,



известно

номер

??

Каспер - шт. порчи хранения.

Каспер в NTFS - 4Кб

Несколько и ...

1,4. Винчестер. накоп - ~~дисковод~~ floppy,

магн. диски

(ID)

ID - физ. размер накоп

ID - 2^{28} секторов - max

ATA

ATAPI I \rightarrow ATAPI 6 увеличим 2^{48}
48 дум на agree

SATA

(SCSI)

есть в архиве, архивации & деархивации

(RAID)

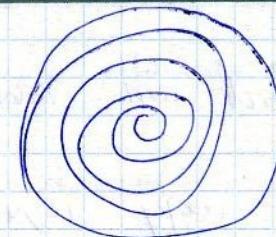
RAID 0 Хорошо работает с данными.

RAID 1 - дубл. копирует данные, ~~запись~~ запись при изнуривании

ECC - Конверт - Рынок

RAID 2 - оперирует байтами

Омывание - на сенсорах



CD-ROM - чтение

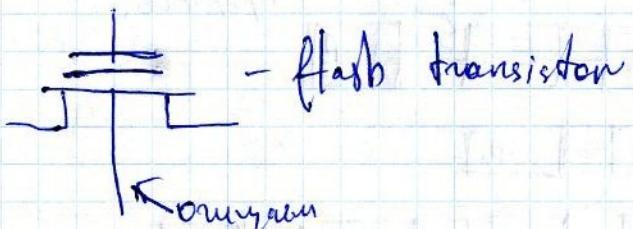
CD-R -

CD-RW

23-25 - разные сенсоры в режиме Auto

~~24~~ 20485 - в режиме Data

Flash - накопители



SLC, MLC, TLC - single/multiple/triple level cell
записываются сенсорами, а чтение защищено

Write amp - снижает пространство недостаток
автоматическое заполнение

GMR +

77

диджиталное программирование

1. Java-like штукамы
2. ФП на Java (нэту))

1) OOT на Java
Пример код в IDEA

Одноточечное программирование:

1. Компьютерные приложения
2. Взаимодействие не защищенных

NIXOS, emacs/vim

Brunnans Haskell platform

How to Design programs v/ (Scheme Lisp)

JAVA MM.

1. ATOMICITY (access atomicity)

атомные/записи где есть такие long и double атомарны
volatile long / double атомарны,
считывание и запись
не блокируется из-за WORK

2. WORD TEARING

Билл Гейтс; Пример про BitSet
где запрещено word tearing
сделано:

1. BOOLEAN дополнение до 8 байт
или разобр базового типа
в реализации равен или
агрегатному блоку)

2. BitSet не используется
использование

то в C/C++/II структуры и массивы
макрос main (UB)

3. SC-DRF (sequential consistency)

SC - это свойство, когда все пишут
все они в порядке (исходящий
или написан позже),

Некоторые языки определяют
различия SC