ССЫЛКИ НА ВСЕ ИЗВЕСТНЫЕ МНЕ СТАРЫЕ(И НЕ ОЧЕНЬ) ДОКИ НА ВТОРОЙ СТРАНИЦЕ			СТАВЬ ЛОСОСЬ ЕСЛИ ЗВЁЗДОЧКА	***Самая важная дока***
если что-то не осилил				безбажные конспектики
Как раскрашены квадратики:	ОК. редактирование ячейки закрыто	**потоковый скаков**	<- рекомендовано к ознакомлению	Скаков - зайка!!!
	необходимо доделать	конспекты y2018 не очень		FAQ by нынешний второй курс
	не просмотрено	гуглопапка		дока препода 34-35 (есть картинки, но много багов
В ТАБЛИЧКЕ ТЕПЕРЬ РАЗРЕШЕНО ТОЛЬКО КОММЕНТИРОВАНИЕ				Какое-то наследие предков
ОЧЕНЬ ИНТЕРЕСНЫЕ ВОПРОСЫ:				Какое-то наследие предков(1)
Рассказать про альтернативные подходы к увеличению скорости доступа, кроме кэша (нет, это не треды. и не НВМ)				Какое-то наследие предков(2)
когда нам выгодно чтобы хазарды случались чаще?				конспекты Жени
Почему ячейки квадратные?)00)0))	Потому что гладиолус(наиболее правдоподобный ответ: хотим уменьшить длину проводов -> хотим уменьшить периметр ячейки -> хотим оптимизировать площадь ->>> делаем ячейку квадратной ***Мб потому что это упрощает производство? Оптимально расположить на плате* квадратики ИМХО проще чем кружочки, звездочки, хуи.** **Ну это-то да, но че не прямоугольники?** **Скаков Сказал, что он хз, почему они именно квадратные, они вполне себе могут быть и прямоугольниками, всё зависит от желания производителей** <- вообще нахождение этого вопроса вверху таблицы намекает, что это немного мем. но спасибо то что уточнили у зайки			Скаков 2к17
0.0 География				билеты by y2017
Что лучше: Уругвай или Парагвай?			Уругвай	еще билеты by y2017

1.1 Элементная база вычислительной системы: логические элементы, триггеры

1	Нарисовать неполный сумматор на заданном элементе	схема с NAnd, где S - бит суммы по модулю, C - бит переноса	
2	Почему jk триггер не может быть асинхронным?	Потому что если уберем сигнал синхронизации, то в состоянии 1 1 триггер будет постоянно менять значение на выходах	
3	Нарисовать RS, D, JK, T триггеры. Их синхронные версии, если есть	Вот тут есть описание всех триггеров	
4	Нарисовать полный сумматор	схема, где S - бит суммы по модулю, C - бит переноса	
5	Что есть (де)мультиплексор. Нарисовать заданной битности	Статья прекрасного человека, которую принял пигвин	
6	Нарисовать побитовый счетчик	http://neerc.ifmo.ru/wiki/index.php?title=Файл:Counter.jpg	
7	Рисовать логические элементы на полевых (мб биполярных) транзисторах	тык	
8	Как выглядит RS? Что будет, если хотим сделать RS на AND, а не на OR	ну придется еще входные сигналы инвертировать, и всё ок. выглядит так же, как RS на NOR'ах, только вместо NOR - NAND	
9	RS-триггер на И-НЕ	Картинка Тригера	
10	Триггеры: Зачем нужна синхронизация?	Так как электрический импульс доходит от выходов одного элемента до входов другого не мгновенно, возможна ситуация, когда мы подали на логический элемент две единицы, одна из них дошла раньше. Несинхронизированный логический элемент вернул результат от неверных входных данных. Это грустно, поэтому нужна синхронизация. Причем такт синхронизации должен быть достаточно длинным, чтобы все значения успели установиться на входах элементов	
11	Построить “умножитель (на входе A0, A1, A2, ,B0, B1, B2, которые задают 2 трехбитных числа, на выходе Q0-Q5 - результат их произведения)	Матричный умножитель	
12	Чем отличается двухтранзисторный инвертор от однотранзисторного?	В двухтранзисторном инвертере выход подключен либо к земле, либо к питанию. В однотранзисторном инвертере при подаче высокого напряжения выход будет ни к чему не подключен. Это очень плохо, поскольку в таком случае можем поймать какое-либо случайное напряжение от других транзисторов. Высокое энергопотребление и тепловыделение. Нужно вставлять резистор, чтобы не было короткого замыкания. Такая конструкция постоянно жрет энергию	тык
13	Нарисуй JK. Обьясни как сломать перемещением инвертора	пример прошлого года с пояснением	Здесь есть тест самого величайшего и он верный. стр 14-15. лень вставлять

1.2 Оперативная память: статическая/динамическая, организация.

1	Почему ячейки хранятся матрично, а не в линию.	Потому что много проводов - плохо. Хранение в линию 1МБ уже дает нам больше 10^6 проводов. А если хранить матрично, то всего около 10^3. Выбор очевиден	
2	Двухпортовая ячейка стат. и дин. памяти. Схемы ячеек памяти	ссылка	
3	Что идейно среднее между однопортовой и двухпортовой ячейками статической памяти?	Многобанковая память.	
4	Преимущества, недостатки динамической и статической ячеек памяти.	**Динамическая:** + дешево + занимает меньше места, чем статическая (транзистор + конденсатор вместо 6 или 8 транзисторов) - необходимо постоянно перезаряжать из-за утечки заряда из конденсаторов - необходимо перезаписывать после чтения, т.к информация при чтении теряется **Статическая:** + не нужно перезаписывать + работает быстрее - занимает много места - дорогая в производстве	Регенерация динамической памяти: - программист (давно-давно) - контроллер памяти (он связывает команды процессора и физическую память) - модуль памяти (лучший вариант, он не забивает шину своим говнищем)
5	Как лучше опрашивать память?Почему номера строк и столбцов лучше передавать последовательно, а не параллельно?	Потому что в два раза меньше проводов, без потери скорости. Так как память устроена матрично нет смысла передавать одновременно потому что нужно время для открытия строки. Номер столбца будет использоваться не сразу. Поэтому можно передавать последовательно уменьшив при этом шину.	
6	Когда бывает выгодно отправлять адрес строки и столбца одновременно?	Если у нас SRAM, то нам не нужно записывать строку в буфер, а значит можно одновременно передавать строку и столбец для ускорения	
7	Что схожего между ячейкой однопортовой и двухпортовой памяти?	они в целом практически одинаково устроены. Только немножко дополнительных транзисторов для второго набора проводов	
8	Может ли динамическая память быть двухпортовой?	да (см. 2 вопрос)	
9	А для чего используется правый провод (!COL)?	Почему существует линия not COL? Предположим, что мы убрали транзистор справа и линию not COL. 1)Тогда, например, если мы хотим записать значение 1 в ячейку, а в ней у нас хранился 0, то нижний инвертор будет мешать нам сделать это, ведь он будет пытаться вернуть значение 0. Если же у нас будет присутствовать линия not COL, мы можем легче подавить старое значение на инверторах, ведь меняем состояние сразу обоих инверторов. 2)Если мы хотим прочитать значение из ячейки без линии not COL, нам придется “честно” смотреть на значение на линии COL, а сигнал может быть недостаточно силен, чтобы отличить 0 от 1. А обычно мы смотрим на разницу в напряжении между COL и not COL	
10	Нарисовать игрушечный модуль памяти с триггерами на 8 бит	Также есть картинка в папке схемы	
11	Нарисуй ячейку SRAM и DRAM. Что за транзисторы в них?	тык.	
12	Зачем закрывать строку перед открытием новой?	Чтобы не потерять данные. (вспомним, что при чтении конденсаторы разряжаются)	
13	Почему не юзаем особо двухпортовую память?	Потому что есть многобанковая, а она дешевле и проще. Возможная параллельность не равносильна удорожанию стоимости и увеличению размеров ячеек.	Note: Двухпортовую юзают когда надо уметь параллельно читать и писать. Например видеопамять(Video RAM). Одновременно считывают чтобы показывать на мониторе и меняют чтоб картинка двигалась

1.3 Оперативная память: характеристики, типы динамической памяти, NUMA.

1	Есть две оперативки, 8-9-8-20 и 9-8-8-20, какая лучше?	При случайных обращениях к памяти - одинаковы. Если читаем несколько столбцов из одной строки, то первая лучше. Случайные обращения к памяти: tRP + tRAS - два последних числа В одной строке: CL - первое число	
2	Дробный тайминг	Вспомним, что DDR работает 2 раза за так, то есть возможна ситуация, что мы начали на фронте, а закончили на спаде, получаетя, что прошло какое-то целое число тактов + 0.5 такта.	В DDR мы увеличили внутреннюю шину и теперь у нас первая половина данных выплевывается сразу, а вторая ждет полтакта. Это же и есть дробный тайминг?
3	Что клёвого в DDR? Почему это лучше, чем увеличить частоту в два раза? что изменили в DDR2 и DDR3?	Передаем информацию два раза за такт. Лучше, потому что увеличение частоты требует больше энергии. В ddr2 увеличили частоту внешней шины и ширину внутренней, уменьшили напряжение питания. В ddr3 сделали все это еще раз.	
4	При каких улучшениях RAM улучшалась скорость передачи? Зачем закрываем строку?	1) FPM, EDO, BEDO, SDRAM, DDR, DDR2, DDR3 2) При чтении строки её содержимое копируется во внутренний буфер, при этом стирается само содержимое строки. При закрытии строки информация записывается из буфера в саму строку. (см ответ 1.2.12)	
5	Как мы получаем в ddr два раза инфу за такт?	Увеличили ширину внутренней шины. Теперь достаем из памяти в 2 раза больше информации, кладем во внутренний буфер, передаем первую половину, ждем полтакта, затем передаем вторую	
6	Почему не получится просто ставить везде выше скорость передачи, а не только в видеокартах?	В чем мем с видеокарточками. Им важна скорость передачи данных, а скорость доступа не так важна. Видеокарточки могут позволить себе отводить побольше тепла. Забивая на тепловыделение мы можем поднять скорость передачи(подробней в конспекте почитаете)	
7	Есть ли какая-то организация памяти многопроцессорной не NUMA?	внезапно, UMA (Uniform Memory Access)	
8	Нарисовать многопроцессорную не-NUMA архитектуру	он давал на лекции	
9	От чего зависит скорость передачи и скорость доступа?	Скорость доступа это время между тем когда мы подали команду и получением реакции на эту команду. Равно tRCD+tCL Зачастую мы просим не один кусочек информации, а несколько подряд. У памяти мы просим прочитать не один байт, а дать кусок памяти с такого то места и длиной столько то. Память передаст нам один кусочек и через время передаст второй, третий и т.д. Скорость передачи данных это время через которое мы получим второй кусок информации после получения первого. Например винчестер. Ему нужно время чтобы найти информацию, которую мы запросили. Это скорость доступа. Теперь когда он нашел данные и выдает подряд куски данных, которые мы запросили, ему не надо ее заново искать. Получается что скорость передачи данных существенно быстрее скорости доступа.	
10	Почему синхронизация в SDRAM - хорошо?	Мем в том, что раньше тактовые частоты на памяти и на остальных устройствах не совпадали, поэтому приходилось ждать столько тактов, сколько нужно + ещё чуть-чуть, чтобы гарантированно получить корректные данные. Потом умные чуваки решили посадить все устройства на одну и ту же тактовую частоту, поэтому теперь можно ждать ровно столько, сколько нужно	
11	DDR, чем он лучше, почему просто не увеличить частоту у SDRAM?	Если сравнивать только скорость передачи, то ничем, но надо понимать, что если увеличить частоту в 2 раза, это приведёт к увеличению энергопотребления. То есть в DDR не происходит увеличение энергопотрибления, я в SDRAM с удвоенной частотой происходило бы.	
12	Изменения в DDR2, DDR3, DDR4?	DDR2: Удвоили ширину внутренней шины до 256 бит и частоту внешней шины => удвоение скорости передачи. Уменьшили напряжение питания до 1,8В DDR3: То же самое, удвоили ширину внутренней до 512 бит и частоту внешней => опять удвоение скорости передачи. Уменьшили напряжение питания до 1.5В DDR4: Удвоили кол-во банков памяти (до 16) + улучшена энергоэффективность (опять уменьшили напряжение питания, ~1.3V), чуть подняли частоты, но не засчет удвоения внутренней шины	Скаков не рассказывал про удвоение банков, и точно говорил что скорость передачи не увеличилась
13	Отличие SDRAM, BEDO и DDR	// Точно такой вопрос? Может сразу все виды опишем, начиная с FPM? тык	
14	Что такое NUMA? Нарисуй.	NUMA - NotUniformMemoryAccess. Устройство памяти компьютера в мультипроцессорной системе. При таком устройстве скорость доступа к памяти зависит от ее расположения относительно процессора. Контроллер памяти вшит в процессор (важно) Пример NUMA-архитектуры на картиночке (сбоку). У каждого процессора быстрый доступ к своей памяти, более долгий к памяти соседних процессоров и еще более долгий к памяти процессора расположенного по диагонали	картинка
15	Основные характеристики оперативки	CL - tRCD - tRP - tRAS CL - CAS (Column Address Strobe) Lattency - число тактов между отправкой адреса столбца и началом получения данных. tRCD - RAS (Row Address Strobe) to CAS Delay - минимальное число тактов, которое нужно подождать при открытой строке, прежде чем открывать столбец. CL + tRCD = время доступа при закрытой строке tRP - Row Precharge - минимальное число тактов, которое нужно подождать между закрытием строки и открытием новой. CL + tRCD + tRP = время доступа, при открытой неправильной строке tRAS - Row Active Strobe - минимальное число тактов между открtыием и закрытием строки.	
16	Что за буфер в EDO памяти?	Буфер адреса столбца. Быстрее считываем столбцы, если они из одной строки	
17	Чем GDDR отличается от DDR? Как мы их можем сравнивать?	GDDR оптимизирована на максимальную скорость передачи данных, но она дорогая, много жрет энергии, хуже скорость доступа	
18	Какие характеристики RAM влияют на скорость работы кэша?	скорость передачи + ширина шины + когда в DDR стали выкидывать данные два раза за такт, кэшу стало приятно (кэш-линия в 64КБ стала заполняться за 2 такта, а не за 4)	Какой то посос. Кешу на самом деле до лампы на скорость передачи(вроде). А вот скорость доступа ему сильно нужнее. Это не точно но с этим надо разобраться

1.4 Кэш-память.

1	Уровни кеша	Обычно есть три уровня кэша. L1, L2 и L3. L3 обычно общий для всех ядер. L1 обычно делится на L1i (кэш для инструкций) и L1d (кэш для данных)	

2	Когда кэш бывает вреден	Такая ситуация возможна, если у нас кэш с write-allocate (т.е. при кэш-промахе при записи мы сначала подгружаем кэш-линию в кэш, а потом уже пишем туда значение). В таком случае, если у нас случайные обращения к памяти, то мы будем делать две операции с памятью (чтение и запись) на каждый запрос. А без кэша или с no-write-allocate (при промахе при записи - просто пишем в память), мы бы делали только одну операцию (запись). - это интеллектуальный ответ, ответ by SKKV выглядит так: "если мы используем данные, лежащие в разных линиях (например с шагом в размер линии), тогда мы передаем всю кэш линию (лишние 64 байт)"			
3	Что такое ассоциативность. Чем плохи и хороши 0 и 1 ассоциативности.	см. вопрос 7			
4	Когда мы записываем в память значение из кэша при Write-Back	Когда выкидываем кэш-линию.			
5	Бывают ли алгоритмы, для которых безразлично - есть кэш или нет?	Да, те, которые читают данные, расположенные в случайных местах памяти Пример: сортировка кучей		Алгоритму не может быть безразлично наличие кэша : он либо полезен либо вреден, как он может не влиять на скорость? Ну и да, пример алгоритмов плес	"любой алгоритм с более-менее случайным доступом к памяти" (ц) скаков
6	Зачем нужен кэш; типичные размеры кэшей разных уровней	Каждый раз читать из памяти долго (чтение занимает порядка 200 тактов). Поэтому давайте заведем небольшую быструю память поближе к процессору. 1-ый уровень ~2x32 Кб и ~4 такта, 2-ой уровень ~(256-512) Кб ~ 12 тактов, 3-ий уровень ~6-8 Мб и ~50 тактов		Почему доступ к памяти именно 200 тактов занимает, это можно как-то высчитать? Это стоило спрашивать на парах. инфа про 200 тактов by Скаков и он дропал её раз 100	
7	Что такое ассоциативность? Рассказать про типичные ассоциативности, адресацию в них, преимущества и недостатки	Ассоциативность кэша - это когда мы разделяем кэш на n множеств, в каждом из которых хранится m линий. Блок в памяти сначала соотносится с множеством, а потом заносится в любую кэш-линию этого множества. 1-ассоциативность (direct mapping) - в каждом множестве ровно одна кэш-линия. У такого подхода к организации кэша есть преимущества: быстро ищем данные, дешевая аппаратная часть (сравниваем только один раз). И недостатки: низкий hit range. Т.е если мы часто достаем из памяти данные, которые попадают в одно множество, то нам грустно: кэш-линия у нас всего одна и данные будут постоянно заменять друг-друга. Выигрыша в скорости особо не получим. Полная ассоциативность: если можем писать любой блок памяти в любую кэш-линию. Очень долгий и дорогой поиск по кэшу. В итоге как-то пытаются балансировать между двумя крайностями. Обычно юзается 8-ассоциативность.		Зая говорил что обычно 8-ассоциативность.. **А что там по адресации?** <- в конспектах почитай, там слишком нужны картинки, чтобы объяснить	
8	Как процессор воспринимает кэш? Видит он его или нет?	Контроллер памяти в процессоре общается с памятью через шину памяти и сам по себе не знает, кто именно отвечает на его запросы. На этой шине могут сидеть другие устройства: другие процессоры, контроллеры со стороны оперативной памяти и, в том числе, кэш-подсистема. На уровне кода кэширование абсолютно прозрачно. Однако с кэш-подсистемой возможно общаться, если она это поддерживает. В современных системах действительно можно явно попросить кэш-подсистему что-нибудь закэшировать, однако это не значит, что она моментально исполнит этот запрос (это рекомендация, а у кэш-подсистемы могут быть свои приоритеты). Более того, она вообще не обязана слушать эти рекомендации и может не исполнять рекомендации извне. Выбрать, откуда взять или куда сохранить данные явно -- нельзя. С точки зрения "скорости работы" для процессора есть заметная разница: обмен данными при наличии кэша в разы быстрее.			
9	Какой недостаток look-aside?	Если нужные данные найдены в кэше, то нужно послать сигнал отмены в память. В современном мире кэш-попадание случается процентах в 90 случаев. А значит, жирная шина между кэшем и CPU дала бы больше выигрыша в скорости, чем немного более быстрый доступ к памяти. А в look-aside сделать жирную шину не получится.		https://stackoverflow.com/questions/34001002/look-through-vs-look-aside пруф так сказать. предыдущие ответы были похожи но не точны	ну и зачем ты совсем стер то, что было(
10	Сравни write-through и write-back.	Write-through: + проще реализовать - забиваем шину запросами к памяти	Write-back: + пишем в память только когда вытесняем линию из кэша - сложнее реализовывать (т. к. нужно хранить кэш линия содержит измененные данные памяти или нет) - если делаем запрос на чтение и промахиваемся, то зачастую обращаемся к памяти дважды: сначала чтобы записать вытесненную строку в память, а затем чтобы нужные данные (все равно лучше, чем write-through)	Второй минус Wb: а разве нужно записывать в память, я думал кэш дублирует память, разве нет? ыыы чел, там write-back, он часто хранит "грязные" данные (т.е. не совпадающие с данными в памяти)	
11	Что быстрее write-through или write-back?	WB, так как когда у нас в память прилетит запрос на чтение не закэшированных данных, перед ним не будет километровой очереди из бессмысленных дублирующих запросов на запись			
12	В чем плюсы и минусы инклюзивной и эксклюзивной архитектур?	Если все кэш-линии кэша присутствуют в кэшах более высокого уровня - то этот кэш является инклюзивным. Если кэш-линии кэша не присутствуют в кэшах более высокого уровня, то он эксклюзивный.			
		Эксклюзивная (AMD) Кэш-линия хранится только в одном кэше + быстрее добавляем в кэш + тратим меньше места - дольше ищем	**Инклюзивная (Intel)** Кэш-линия хранится в более чем одном кэше + быстрее выкидываем из кэша (не нужно просевать в кэши большего уровня, тк там уже присутствует копия этой кэш-линии) + быстрее ищем - тратим больше места		

1.5 Протоколы когерентности кэш-памяти

1	Зачем нужны протоколы когерентности? Как обойтись без них?	Представим себе ситуацию, что у нас несколько ядер на процессоре. У каждого ядра есть свои кеши L1, L2 и общий кеш L3 (примерно так всегда на современных процессорах). Допустим, что одно ядро взяло данные из RAM и изменила их, другое ядро тоже хочет взять данные по тому же адресу и может взять уже старую, неактуальную версию этих данных. По этой причине и существуют протоколы когерентности, которые контролирует такие баги. Обойтись без этих протоколов нельзя, кроме как сделать один кеш на все процессоры (см.88 строчку)		Можно обойтись без когерентности еще одним способом. Когда у нас идет взаимодействие между двумя разными тредами, то давайте будем сбрасывать кеш нужных данных в каждом треде, когда они там используются. Еще можно вынести кеши за шину, но тогда проигрываем по скорости Инфа из прошлогодних аудио	5
2	Как заработать состояние Owned (изначально кэши пусты)?	MOESI: Пусть к шине подключены два ядра со своими кэшами. Первое ядро отправляет на шину памяти запрос на чтение ячейки X. Так как этой ячейки нет ни в одном из кэшей (она в состоянии Invalid), то запрос отправляется в оперативную память. Когда оттуда приходит ответ, в кэше первого ядра ячейка X появляется в состоянии Exclusive (в кэше второго ядра она Invalid, поэтому эксклюзивная в единственном варианте). Затем первое ядро должно отправить на шину памяти запрос на запись в ячейку X. Эта ячейка есть в его кэше, поэтому он обрабатывает данный запрос: записывает в неё новые данные, переводит её в состояние Modified. Так как она присутствовала только в кэше первого ядра, то в остальных кэшах ничего не изменяется. Далее второе ядро отправляет на шину памяти запрос на чтение ячейки X. В его кэше эта ячейка находится в состоянии Invalid, однако в кэше первого ядра она присутствует в состоянии Modified, поэтому кэш первого ядра отправляет на шину её значение, а сам переводит её в себе в состояние Owned. После этого кэш второго ядра принимает ответ от кэша первого, отдаёт его второму ядру, и записывает ячейку X себе в состоянии Shared.		(* Где-то тут нужно указание на картиночку с переходами между состояниями в MOESI, которая на листе с картинками *)	
3	Расскажи про состояние "E" в MESI	E - Exclusive. Данный блок присутствует только в текущем кэше и является "чистым". Может быть изменен на Shared при получении запроса на чтение. Может быть изменен на Modified при получении запроса на запись в этот блок.			
4	Процессор хочет записать в памяти строку: MESI (все случаи) и MOESI (мы попали в O)	**MESI:** 1. Попали в Modified - изменяем локально и все ок 2. Exclusive - переводим в состояние Modified, пишем локально, все ок 3. Shared - просим все кэши выкинуть свои копии, переводим в состояние Modified, пишем, все ок 4. Invalid - просим все кэши выкинуть свои копии, читаем себе в кэш, изменяем. новое состояние строки - Modified	**MOESI:** Owned - пишем данные локально, переводим в состояние Modified, сообщаем остальным кэшам удалить свою копию этой кэш-линии		
5	В чем отличие msi от mesi, чем плох msi	MSI плох тем, что мы забиваем шину ненужными запросами. А именно: каждый раз при изменении состояния с Shared на Modified мы просим другие кэши выкинуть данные (вдруг они у них есть), но чаще всего данные есть только у нас в кэше. Давайте избавимся от этой проблемы, добавив состояние Exclusive. Теперь мы можем изменять с E на M без каких-либо оповещений других кэшей.			
6	Какие ошибки могут возникать в системе из двух кэшей при отсутствии когерентности?/Привести код, который ломается без когерентности	крч есть два треда: Первый читает переменную a = 5 из памяти и циклится while a == 5 Второй такой a++ и циклится while a != 5 С одной стороны, а не может быть одновременно равна и не равна 5, а с другой, без когерентности именно так и получается. Если добавим когерентность, то зациклится только второй			
7	Как эти ошибки исправить не используя когерентность? Как эти ошибки исправляет когерентность?	При когерентности первый тред получит значение a = 6 и цикл прервется ***Не*** используя когерентность - в тех местах программы, где происходит обмен данными между тредами(или хотя бы потенциально может происходить) надо ставить команду(ручками) "сбросить кеш"			
8	В каких случаях когерентность не возникает при трех кешах?	Если данные, которыми пользуются кэши, не пересекаются (Тут надо сказать, что разные КЕШ-ЛИНИИ, т.к. я могу одним кешом попросить записать в адрес 228, а другим в 229, то я двумя кешами попал в одну кеш-линию и проиграл, хотя, по факту, адреса разные)			
9	Можно ли придумать такое устройства КЭШа, для которого не нужны протоколы когерентности	Да, один общий кеш на все процессоры			
10	Почему проблему когерентности не решают WT кэшем?	пусть у нас есть 2 треда и 2 кеша соответственно, в оба из них подгружено a = 3, тогда если один трад изменит значение a, скажем a = 5, то даже если кэш WT другой кэш никак не заметит это изменение пока не выкинет свое значение и не подгрузит его заново. То есть даже при WT в кэше будет лежать не актуальное значение			

1.6 Носители информации: магнитные, оптические и на основе флеш-памяти, RAID

1	SLC и MLC, что такое, + и -	SLC - Single Level Cell - есть два уровня напряжения: высокое и низкое. Храним один бит в клетке. MLC - Multi Level Cell - есть четыре уровня напряжения. Храним два бита в одной клетке. **SLC:** + выше скорость записи + большая износоустойчивость клетки + ниже энергопотребление - дорого **MLC:** + выше плотность записи за счёт того, что храним 2 бита в одном транзисторе + стоимость ниже - Ниже скорость записи и чтения			
2	Когда не работает TRIM	из прошлогоднего конспекта: 1) если это RAID 2) если подключен по USB (ограничение протокола) 3) если диск зашифрован программно (тогда нужны соседние блоки для декодирования) 4) не поддерживается ОС 5) не поддерживает устройство 6) если диск виртуальный		еще про TRIM	у TRIM есть явные проблемы с RAID. Для исправления этого интел выпустил специальный патч, но он только для интел..
3	Сравнение винчестера и SSD. Почему скорости могут плавать у обоих;	У винчестера разное кол-во секторов в начале(с краю) и в конце(в центре), а угловая скорость - const поэтому скорость вначале может быть даже в 2 раза больше чем в конце, а у SSD если мы записываем на чистый сектор, то стирать не надо, а если перезаписываем(стираем + записываем), то будет явно дольше. Версия 2: Сравнение скоростей: скорость доступа у SSD выше. Скорость передачи у HDD выше.Сам Паша говорил, что скорость чтения/записи(пишем поверх без очистки) у HDD выше. Плавать скорость у HDD может например: 1) при чтении(и записи наверное тоже) при увеличении номеров сектора(при движении от края к центру) скорость чтения снижается. 2) можем перезаписывать не по 4 Кбайт, а меньше. Тогда должны прочитать весь сектор, вставить туда новый фрагмент, пересчитать CRC/ECC и обратно записать. 3) плохо разбиваем HDD на кластеры(надо, чтобы раздел на диске начинался с сектора, номер которого, кратен 8 (4 Кбайт(физическое разбиение на сектора) / 512 байт(логический размер сектора, вспинаем про эмуляцию 512e) + размер кластера у файловых систем кратен 4 Кбайт) У SSD: 1) зависит от режимов транзисторов(SLC MLC TLC QLC в порядке падения скорости записи, чтения) 2) скорость записи плавает от «чистоты» SSD (на новый все пишется быстро) 3) крутость контроллера 4) кол-во свободных ячеек для маневров контроллера(все время перестраивает, чистит)(чем больше места, тем быстрее работает SSD, т.к. контроллер может быстрее обработать запрос(в том числе повышая скорость доступа), когда(если) завершит свои делишки)		что значит плавать? Не иметь константного времени доступа? именно	
4	Достоинства и недостатки флеш-памяти Почему время поиска у flash меньше, чем у жестких дисков	Потому что производители хитрые и сделали их тупо параллельными, т.е. де-факто у нас не один, а много таких дисков Чел сверху(я снизу) прав, но параллелизм это про скорость передачи, а поиск == скорость доступа, она у flash меньше, потому что там не надо "попадать в сектор" то есть покрутить диск + передвинуть головку, там просто есть адрес памяти по которому мы идём			
5	Что такое trim?	TRIM - команда, которая позволяет ОС сообщить SSD какие блоки данных уже не используются и их можно удалить. Появилась скоро после появления SSD, т.к. скорость записи в непустой блок у SSD очень грустная. (надо сначала стереть, потом записать)			
6	Что такое write amplification и почему оно может быть больше/меньше 1. Стандартные значения WA	Вспомним, что в SSD есть интеллектуальный контроллер, который равномерно распределяет нагрузку, то есть если он видит, что в какую-то ячейку часто записываем, а в каку-то нет, то очередной запрос пойдёт в ту, в которую реже(логически он пойдёт куда должен, но физически в ту область которая менее нагружена, потому что контроллер перемапит), чтобы не потерять данные которые были в редко используемой ячейке, мы должны их тоже перезаписать, отсюда и появляется > 1 Версия 2 Есть хороший первоисточник<---->(через ячейку) Если кратко: поверх записать не можем. Но минимальный блок для стирания гораздо больше минимального блока для записи. Хотим поменять небольшой блок файла. Итог: должны прочитать много, стереть много(а стирать, по словам Скакова, медленнее, чем писать), исправить часть и записать снова много. Вот тут и 1<WA = Данные, реально записанные на флеш-память / Данные, отправленные на запись хостом. Все это дело слишком долгое. Поэтому контроллер пишет на заранее очищенные ячейки и меняет информацию о фактическом нахождении части файла в таблице трансляций и помечает старый блок «доступным, но не готовым». По мере жизни SSD все больше таких простаивающих блоков получается. Что делать? Избавляемся от груза, пока никто не видит(пока нет задач на запись и чтение), но тогда может быть, делаем лишнюю работу, так как при стирании ненужных блоков мы должны перезаписывать данные, которые потом(возможно) удалятся(еще вспомним что кол-во перезаписей ограничено). Зато последующая запись на очищенное заметно быстрее. Это фоновая очистка. А можем делать непосредственно во время записи новых нужных данных(явная очистка). В итоге стараемся комбинировать два способа очистки, снижая ущерб от обоих. По поводу стандартных значений – не понятно. Паша говорил что то про WA>1, 2... и < 1, если контроллер умеет сжимать данные.	он чето говорил, что какие то бенчмарки тестировали на наборе 0 и было бешенное кэширование Разве Write Amplification это не просто отношение количества информации, которое реально записывается на диск к количеству информации, которое должно быть записано? И Write Amplification > 1 когда, например, мы записываем что-то на область SSD, которая занята. Тогда мы стираем большой блок, а потом восстанавливаем его с измененным сектором и получается, что надо было записать 1 сектор, а мы записали целый блок секторов, а < 1 когда у нас диск может в скатке данных и вместо большого объема пишет меньший, скав его	Хорошо написано
7	Почему размер сектора жесткого диска ровно 4Кб?	переход на секторы размером 4 КБ позволил увеличивать плотность записи данных и емкость жестких дисков, а также повышать надежность исправления ош Рассмотрим первый случай: 8 секторов по 512Б. У каждого из них существуют М (?Байт?) служебной инфы, с помощью которой можно в каждом блоке восстанавливать до М / 2 ошибок. Во втором случае: 1 сектор 4КБ и 8М служебной информации. Заметим, что суммарно по 8 блокам в первом случае можно исправить 4М ошибок, а также во втором случае - суммарно 4М ошибок. Но так как сектора в первом случае по 512Б, то отдельно в каждом секторе можно исправить не более М / 2 ошибок, а значит, что если в каком-нибудь секторе возникнет, например, 2М ошибок, то их исправить мы уже не сможем. Размер в 4КБ связан с тем, что у файловых систем кластер в 4Кб.		переход на секторы размером 4 КБ позволил увеличивать плотность записи данных и емкость жестких дисков, а также повышать надежность исправления ошибок. Допишите про ФС	<- а насколько подробно ты хочешь про ФС?
8	Почему запись на диск происходит посекторно, а не побайтово?	Поскольку оборудование неидеально, то иногда некоторые битики могут записываться неправильно, поэтому существуют ECC и CRC, которые фиксят ошибки. Очевидно, что сделать такое для секторов будет проще, а также будет занимать меньше места служебная информация, чем для каждого байтика писать ещё дохера байтиков для того, чтобы гарантировать корректность. Ну и из устройства DDR-памяти ясно, что передовать сразу несколько байтиков быстрее, чем один (эта одна из модификаций)			
9	Когда чтение диска не происходит?	Мы захотели прочитать, то есть прочитали якобы то, что записали, потом сделали ECC, оно типо исправило ошибки, потом пересчитали CRC и сравнили с тем, которое есть, если они совпали, то мы прочитаем, если нет то ничего диск нам не отдаст (слушайте диджея зайку, трек "2.6_2 и немного 1.6" таймкик 34:29)		Поясните за вопрос плиз у тебя есть диск(магнитный или оптический), ты хочешь его почитать, но он не даёт тебе данные, такое бывает когда они там ошибочные на лекции был пример с телеком, что тот вначале (когда мало ошибок) фиксит их, а потом резко выключается, так это потому что он больше не может их фиксить, а ошибочную информацию не выдают, также и в дисках	

10	Главные отличия магнитных и оптических носителей	Способом хранения информации: магнитный диск хранит информацию с помощью магнитных полей, а оптический - с помощью ямочек на дорожках	
11	В чем схожи оптические накопители и flash-память	Нет так таковой операции перезаписи: ты вначале всё стираешь подчистую (возможно, тебе придётся записать куда-то данные, которые ты не хочешь терять), и потом на чистое записываешь то, что считаешь нужным	кажется неполным ответ.. (сделайте пометку правее если что-то изменится плиз)
12	Чему равен сектор при записи аудио?	2 352 байта, т.к. при записи аудио нам всё-равно, что некоторые битики будут неправильными (человеческое ухо не поймёт разницы), поэтому мы отдаём привычное место ECC и CRC под звук. Но также есть специальные требования к самому аудио файлу: стерео, 16 битное кодирование, 44 100 Гц частота	
13	Есть два диска в черных коробках, один - магнитный, другой - оптический, как определить, где какой?	Есть тупое решение, как например положить их в микроволновку. Оптический диск будет жариться с потрескиванием, и слабыми искрами (которых в коробке то и не увидишь). А вот магнитный выдаст неплохую порцию огня и вообще беги от этой микроволновки хорошее решение :D (оно одобрено? Серьезно??? (с) Автор) одобренное покрасилось бы в зеленый) Ещё есть версия попробовать использовать магнит (но нужен очень мощный магнит), а также потрясти (но я, что-то, не верю, что это рабочий способ)	Стоит сравнить, как изменяется скорость считывания. Далее нужно понимать, что скорость считывания сектора линейно зависит от (длина сектора) / (длина окружности, на котором находится сектор) при фиксированной угловой скорости Оптический диск: однослойный - скорость будет равномерно только увеличиваться двуслойный - скорость равномерно увеличивается, затем равномерно уменьшаться. Связано это с тем, что у нас одна дорожка (в однослойном диске), начинающаяся в центре диска, в которой длины секторов одинаковы Магнитный диск: скорость считывания уменьшается "скачками", т. к. считываем, начиная с внешней дорожки, и дорожки разбиты на "группы", где каждая дорожка разбита на одно и тоже количество секторов.
14	RAID 0,1 , 0+1, 4, 5, 6, их + и -, отличия от остальных + че такое Рид-Соломон	**RAID** - Redundant Array of Inexpensive Disks - Избыточный массив недорогих дисков. Альтернатива SLED - Single Large Expensive Disk. **RAID 0** - просто несколько дисков, данные распределяются по всем. Не использует ни бит четности, ни какой-либо ECC. Вообще-то так себе RAID, т.к. никакой избыточности в нем нет. Поддерживает диски разного объёма, но объём памяти, который добавляет в массив каждый диск будет равен объёму наименьшего из дисков. Теоретически, если у нас есть n дисков, то мы улучшим скорость передачи в n раз. **RAID 1** - каждый диск - точная копия первого. Массив может быть настолько большим по памяти, насколько большой у нас самый маленький диск. Не использует бит четности, Рида-Соломона и всё такое. Повышает скорость чтения, но ухудшает скорость записи. **RAID 4** - (n-1) диск представляет собой RAID 0 массив, а n-ный диск хранит в себе бит четности всех дисков. Неплохая скорость чтения, но с записью всё плохо. Кроме того, диск с битами четности используется сильно чаще остальных и потенциально отвалится первым. **RAID 5** - теперь блок с битами четности распределен по всем дискам. **RAID 6** - теперь у каждого диска есть еще и блок с кодами Рида-Соломона, что повышает надежность. **RAID 0+1** - RAID 1 массив из RAID 0 массивов **RAID 1+0** - RAID 0 массив из RAID 1 массивов Коды Рида-Соломона позволяют при добавлении к сообщению длины N доп.информации длины M исправить M/2 ошибок из всех (N+M) битов.	Таненбаум, страница 115 а на википедии еще и картиночки красивые есть
15	Почему флешка работает медленнее HDD?	Ну потому что менять состояние транзисторов (особенно если это MLC или TLC гораздо дольше, т.к. надо точнее подать напряжение, чем поменять состояние с помощью магнитного поля. На SSD проблема решается тем образом, что используют дохера планок, и мы получаем какую-то пародию на параллельность	Возможно потому что во флешку не запихнуть умный контроллер
16	Как лучше поцарапать оптический диск: по дуге или вдоль радиуса	Если поцарапать диск по дуге, то часть данных пропадет и не восстановится. Если вдоль радиуса, то код коррекции должен справиться.	
17	Что покрывает что: ECC или CRC?	ECC покрывает CRC, поскольку ECC может поисправлять ошибки, а CRC лишь чекает, что ошибок нет. Для хомяков: сначала делают CRC, затем ECC. CRC - хэш-функция, ECC - код коррекции.	
18	Зачем сектор во флеш	"а как без них?". Побайтовая передача неудобна, ибо отправив большой запрос (адрес) мы получим в ответ только один байт. Т.е. чтобы считать тот же 4-килобайтный файл потребуется забить шину запросами. Медленно и неудобно. + стоит почитать в таненбауме/википедии про виртуальную память. без сектора грустно. + " чтобы использовать интерфейс жестких дисков" (ц) Скаков	-> Это унаследованный от жестких дисков интерфейс <- и че, поч новый не сделали? -> Хз, возможно чтобы для системы выглядело как HDD, если контроллер памяти на процессоре, ему иначе пришлось бы объяснять, почему пропали сектора и как этим пользоваться, а так он ничего не заподозрит Не забывайте, что переписывание данных происходит таким образом: "Запомнить что было в секторе до этого, стереть всё, записать на чистое, не теряя при этом нужных старых данных и добавив новых", мб поэтому они и нужны по флеш Н: Стирание не посекторное, а поблоковое, блок из многих секторов, в теории ему сектора и не особо нужны для стирания и перезаписи. Еще вариант, сектора нужны, потому что ECC и CRC так накладывать разумнее Мб запишем несколько версий. И давайте каждую новую реплику выделять отличным цветом. И ты уверен(а)(о), что запись не происходит не посекторно (см. 96 строчку) -> напиши @gleb12 Н: Почему читать по одной ячейке разносит ресурс памяти? Мы же все равно и так и так читаем блок ячеек. А без секторов мы бы вообще читали не весь сектор (512 байт, например), а только то, что нам нужно "Не побитовым же чтение делать... Посекторное оно только для того, чтобы оптимизировать количество и эффективность операций чтения-изменения-стирания-записи. Работает оно примерно как с оперативкой (там же интерфейсные схемы похожие) После команды чтения сектор копируется в внутренний буффер из SRAM. Если дальше приходит команда на запись — сначала изменения остаются в этом буффере. Потом сбрасываются Если бы чтение было побитовым, то при последовательном чтении (да и не только) интерфес был бы забит командами Т.е. там должна быть какая-то минимальная единица информации для обмена с устройством, это как раз и есть сектор Почему сначала решили сохранить 512 байт — вот тут может быть совместимость (хотя не жёстко)" - @mwwmt
19		Скорость, стоимость, ограниченность перезаписи, устойчивость к тряске	

2.1 Архитектура фон Неймана и её альтернативы

1	Рассказать основные принципы.	1. Следует использовать двоичную систему счисления 2. Программный принцип управления, т.е. обеспечивает универсальность разработки 3. Адресность памяти: у каждой ячейки памяти есть свой номер и к любой ячейке должен быть быстрый доступ 4. Последовательность выполнения команд 5. Однородность памяти, т.е. в одной памяти хранятся и данные и команды	
2	Рассказать про гарвардскую систему. Сказать что модифицированная есть.	Гарвардская архитектура - те же принципы, что и в Фон Неймане, кроме последнего. В Гарвардской архитектуре разные каналы и физические устройства хранения для данных и инструкций. Про модифицированную версию: есть несколько вариантов, все они используют принцип "а давайте немного ослабим разделение между данными и инструкциями". Скаков рассказывал про split-cache: у нас разные кэши для данных и инструкций, но память общая. Собственно программист может никогда и не узнать, что у него модифицированная гарвардская, но выигрыш в скорости он получит	
3	Привести альтернативы каждому принципу Фон Неймана	1. Десятичная(!!!) система счисления 2. Аппаратное управление 3. Машина Тьюринга 4. Суперскаляр OoO 5. Неоднородная память (например Гарвардская архитектура)	Чем хуже? 1) 2) программное управление универсальнее, можно прогать разные алгоритмы. а аппаратное тупо делать железо под определенный алгос 3) у машины тьюринга относительная память, можете обмазаться вики 4) суперскаляр может оптимизировать выполнение параллельных действий - по времени выиграем, но железка сложнее и дороже 5) однородность проще, можно динамически строить код. но медленне и есть баги в безопасности
4	*Принципы Фон Неймана. Почему двоичная запись (первые машины были 10-ные)*	Сначала пытались сделать 10-ные, но затем поняли что: - двоичную систему проще реализовать (достаточно заряжать/разряжать ячейку памяти) - довольно эффективная скорость вычисления (лучше только троичная система счисления, но она хуже исследована и сложнее)	Если Паша спросит, почему не юзается 3-чная, скажите, что использование 2-чной сложилось исторически, а на 3 -ную никто переходить не будет, тк 2-ная хорошо изучена и исследована, а так же не выгодно писать софт, придумывать стандарты и тд, под новые 3-чные компы
5	В чем разница между фон Неймановской архитектурой и Гарвардской с точки зрения программистов?	В Гарвардской архитектуре принципиально невозможно осуществить операцию записи в память программ, что исключает возможность случайного разрушения управляющей программы в случае ошибки программы при работе с данными или атаки третьих лиц.(http://digteh.ru/dsp/garvard/)	
6	Че лучше. Аппаратное или программное управление? (любимый вопрос Скакова)	**Аппаратное** - невозможно исправить ошибки в уже выпущенной железке. Отзывать их и делать новое дорого. Можно в документации конечно прописать что соре за ошибку, но это не очень хорошо. + сложнее хакнуть, потому что железка умеет делать, только то что вшито при производстве + жрет меньше энергии. Иногда вместе с программным вставляют аппаратные блоки, реализующие какую-нибудь штуку(например видео декодер), в целях оптимизации энергопотребления и тепловыделения. - с другой стороны заточено под определенный стандарт, например, если видео в каком-то не подходящем формате, то декодер может не справиться **Программное** + возможность исправления ошибки(выпустить апдейт и готово) +просто в разработке. Берем готовый универсальный вычислитель, который уже был сделан, и радуемся жизни. -С другом стороны железка будет дороже * для специализированного железа лучше сделать аппаратное, потому что стоить будет дешевле. Выгодней такие железки штамповать, чем сложное дорогое железо, в котором будет использоваться лишь одна функция(это все зависит от объема производства, в маркетинг упирается карочу)	Если есть что добавить или уточнить @willy_tiger
7	**Есть два устройства: одно на Фон Неймане, другое на Гарвардской. Как отличить? (включать нельзя)**	На кристалле у Гарвардской будут две шины: для данных и для команд. А у фон Неймана только одна.	
8	**Есть два устройства: одно на Фон Неймане, другое на Гарвардской. Как отличить? (открывать коробки нельзя)**	Программно: Однородная память ячейка может быть или данными или кодом, в зависимости от того как мы к ней обращаемся. Есть читать, записывать, то воспринимаем как данные. Если goto то как команду Неоднородная память. у нас отдельные указатели для памяти и команд адресное пространство нарезано на кусочки, какая-то область адресов отвечает за память, а другая за команды. К каждой области мы можем обращаться только по своему. Т.е. к памяти данных нельзя обращаться как к командам.	
9	**Есть два устройства: одно на программном управлении, другое на аппаратном. Как отличить? (нет, не посмотреть на кристалл - он может быть одинаковым, только у аппаратного отрублена программируемая часть)**	Ну казалось бы они будут отличны внешне, когда разберем. Этот вариант, ему конечно тоже можно вкинуть. Но. Внимание. Они могут быть внутри абсолютно одинаково устроены, но у аппаратного может быть аппаратно заблокирована расширяемость. Можно еще вкинуть про то, что у программного будет версия прошивки, и соответственно его можно обновить. Можно вкинуть: посмотреть в документации(но это чето тупо конечно, но на уровне вброса неплохо)	
10	**Есть ли отличие между блоком данных для команд и данных в Гарвардской архитектуре?**	В гарвардской архитектуре характеристики устройств памяти для инструкций и памяти для данных не обязательно должны быть одинаковыми. В частности, ширина слова, тактирование, технология реализации и структура адресов памяти могут различаться.	Длина слова, тайминги, технологии реализации двух блоков памяти могут быть различными. В некоторых системах программа исполнения записана в ROM'е, т.к. нет необходимости ее менять, а вот блок данных может быть перезаписываемым. Где-то, например, требуется больше памяти для команд => вместимость блоков и размер адресов для команд и данных могут быть различными.

2.2 Архитектура набора команд (ISA) и микроархитектура

1	Что такое Isa? для чего она? как она определяется? Что в себе содержит? Описание происходящего при ошибках (!)	ISA - Instruction Set Architeture (Архиктектура Набора Команд). Зная ISA и зная наш код, нашу программу, мы можем точно определить как она будет выглядеть в откомпилировнном виде. ISA определяет: 1) Архитектуру памяти. Стек, аккумулятор, Reg-Reg, Reg-Mem, Mem-Mem 2) Набор команд (доступных программисту, пишущему на машинном языке) 3) Колличество регистров на кристалле. (CISC (complex instruction set computer) - обычно до 32, RISC (reduced instruction set computer) - обычно больше 32) 4) Обработка ошибок. (Протокол обработок ошибок, типо че и как делает наш код в случае если случилось говно. Конкретно см. внизу ячейки) 5) Разрядность адресов. (Видимо, адресов памяти) 6) Доступные типы данных. (целые, с плавающей точкой, знаковые, беззнаковые, векторные) 7) Протокол взаимодействия с внешними устройствами ввода и вывода 8) Режимы адресации: а) непосредственная адресация - хранение в адресной части самого операнда, а не его адреса или какой-либо другой инфы. таким образом можно работать только с константами. б) прямая адресация - хранение полного адреса операнда. команда всегда имеет доступ только к одному и тому же адресу памяти. значения в нём могут меняться, а вот сам адрес - нет. таким образом прямая адресация может использоваться только для доступа к глобальным переменным. в) регистровая адресация - напоминает прямую, только в данном случае вместо ячейки памяти указывается регистр. благодаря быстрому доступу к регистрам и их коротким адресам это самый распространённый режим адресации. многие компиляторы определяют самые часто встречающиеся переменные (например, параметр цикла for) и помещают их в регистры (ее оптимайз). г) косвенная регистровая адресация - искомый операнд берётся из памяти или отправляется в память, но его адрес не фиксируется в команде, а содержится в регистре. если адрес используется таким образом, он называется указателем. преимущество - можно обращаться к памяти, не имея полного адреса. можно использовать разные слова памяти, меняя лишь значение в регистре. (гляньте на стр.407 Андрюхи Таненбаума, там показано, как это может упростить жизнь). д) индексная адресация - обращение к памяти по регистру и константе смещения (от себя : полагаю, что это напоминает обращение к ячейке массива : ук азатель на начало + индекс смещения). Пример использования данного режима адресации см. стр.409 Таненбаума) е) относительная индексная адресация - адрес вычисляется путём суммирования значений ячеек двух регистров и смещения.один из регистров - база, а другой - индекс. // MOV R4, (R2 + R5) *начинаем упарываться* ж) стековая адресация - максимально короткие команды. настолько сука короткие, что в них нет адресов. это возможно при наличии стека. при работе со стеком обычно используется постфиксная запись. ВАЖНО! Хорошая архитектура должна определять набор команд, которые можно эффективно реализовать не только в современной, но и в будущей технике. ОБРАБОТКА ОШИБОК Следующее предложение означает, что если программа выполняет код операции, который не определен, он должен вызвать системное прерывание, а не просто игнорироваться: Выполнение зарезервированного кода операции должно вызывать системное прерывание. Может быть и альтернативный подход: Результат выполнения зарезервированного кода операции определяется реализацией. Это значит, что разработчик компилятора не может рассчитывать на какое-то конкретное поведение; следовательно, у разработчиков появляется свобода выбора.			
2	Что стало лучше после появления ISA?	После появления ISA программы, написанные для одного компьютера будут работать на любом другом, с такой же ISA. Вне зависимости от аппаратной реализации этой ISA. Т.е. возникла некоторая "прослойка" (абстракция) между аппаратным и программным обеспечением			
3	Что делает ISA для того, чтобы проги юзались на всех железках?	Обратная совместимость - способность новой машины выполнять старые программы без изменений Новые компьютеры дополняют старый набор команд <=> один и тот же софт работает на новом, если на старом робил. (все компьютеры различаются лишь внутренним устройством) Кроме обратной совместимости: любые железки любого производителя имеющие одну и ту же ISA могут выполнять программы, написанные для этой ISA		про имеют одинаковый набор не очень грамотно..(Fixed)	
4	Что такое Микроархитектура и "с чем ее едят"?	В компьютерной инженерии микроархитектура также называемая организацией компьютера — это способ, которым данная архитектура набора команд (ISA) реализована в процессоре. Каждая ISA может быть реализована с помощью различных микроархитектур. Например: Pentium и Intel Core - разные микроархитектуры, которые поддерживают одну и ту же ISA (x86). Или System/360. Грубо говоря, сложение 32-х битного числа может быть реализовано 32-х битным сумматором, а может – с помощью двух 16-тибитных.			
5	Приведите примеры Микроархитектуры.	Haswell, Sandy Bridge, Nehalem, Skylake - x86 (еще Pentium и Intel Core) Со стороны ARM могу привести пример sun3i (sun4i -- sun 8i)			
6	Примеры разных ISA (хватит двух) и разных микроархитектур с одной ISA	ISA: MIPS, ARM, x86. Pentium и Intel Core - разные микроархитектуры, но одна ISA (x86) System/360 - чудесная IBMовская линейка времен динозавров. Все модели поддерживали одну и ту же ISA, но реализована она была по-разному. (у дешевых моделек железо попроще, работали они, соотвественно, дольше и памяти имели меньше).			
7	Различия CISC и RISC	CISC: + переменная длина команд(1-15 байт) (частые команды занимают меньше места) - сложные команды переменной длины - сложно декодировать - Reg-Mem или Mem-mem - Арифметические команды выполняются целиком за один раз - Сложно параллелиться - мало регистров. Регистры специализированные. - сложная система обращения к памяти Про что это? О том что можно писать в квадратных скобочках в команде. Возможно сложное обращение к памяти, типа умножить на что-то регистр, прибавить еще что-то	RISC: + быстрее, чем CISC. Т.к места под часто выполняемые команды на кристалле больше, длина инструкций фиксированная (проще декодировать) + reg-reg + много регистров и они общего назначения + простая адресация к памяти. Можно только смещение юзать	в риске простая адресация к памяти?? у меня в конспекте ровно наоборот <- соре, такой конспект говно **А можно услышать аргумент, почему она простая и почему она сложная?** **Чет я в этот пункт не вдупляю** Если вопрос в целом, то ответ - из-за декодера, ему надо декодировать не 1 команду, а 3-4 (у нас же папаллельные конвеиры), а определить где начало 3 или 4 команды когда у них не фиксированная длина это классное развлечение, при фиксированной можно просто скопипастить 4 декодера и парарельно декодить <- его дичь. в RISC простая адресация к памяти, потому что там нет чудесного режима адресации Scale как у, например, x86 с адресами вида "eax + 100 + 4*ebx". в RISC придется сначала ручками посчитать эту прелесть, а потом уже обращаться к памяти. в CISC так адресоваться к ячейке можно одной командой	
8	Что быстрее: стек, аккумулятор, reg-reg, reg-mem, mem-mem?	reg-reg Ну, смотри, самой оперативки сильно больше, чем регистров, поэтому и адреса в ней длиннее. Из-за этого команда для *-mem будет занимать больше места даже если оба аргумента — регистры, поэтому сам код больше и занимает больше места, в том числе в кэше. А если кэш забит бесполезными данными, то приходится обращаться к оперативке, которая сильно медленнее. Чем компактнее команды — тем они лучше кэшируются, тем быстрее всё и работает.			

2.3 Конвейерная архитектура, конвейер MIPS

1	На конвейере к примеру требуется 7 тактов, чтобы выполнить 1 команду. Но ведь можно взять убрать конвейер, и для этой команды к примеру будет достаточно 3 тактов (или вообще сделать супер железку и будет нужен 1 такт). Получается конвейер хуже?	Конвеер лучше, потому что: 1. длина такта у конвеера короче 2. больше полезного делаем за единицу времени (выше IPC Instruction Per Cycle) т.к по максимуму используем железо в целом у конвеера выше latency, но сильно меньше throughput. Мы как бы жертвуем временем исполнения одной команды от начала и до конца, чтобы получать команды каждый такт.		
2	Почему бы не сделать процессор с кучей этапов на конвейере, как например Pentium 4, у которого 31. Считаем, что возможно так оптимизировать, что каждая малая деталь будет работать достаточно быстро, что это получится так же по скорости, как и нормальный конвейер.	Вариант 1: Пентиум грелся шо пиздa))0) а ещё энергии жрал как i7 и вообще выигрыш там был исключительно за счёт высокой тактовой частоты (если 3 и 4 пентиумы запустить на одинаковой частоте, то 4 проиграывает. Собственно поэтому после него Интел обратно откатилась на архитектуру третьего) Вариант 2: при любых JMP будет очень неэффективно использоваться конвеер, т.к. придется откатываться на очень много стадий назад. Вариант 3 (дословный Сказов): хазардов станет так много, что станет больно и дорого. упадет IPC. Будет много NOPов. Будет жрать энергию как тварь.		
3	Что выполняется на каждой стадии MIPS конвейера на примерах команд JMP, LD, SUB	**JMP:** IF - прочитали команду из памяти ID - поняли, что это JMP, поменяли значение в Program Counter (Сказ называет эту штуку Instruction Pointer как в x86) EX - NOP MEM - NOP WB - NOP **LD:** IF - прочитали команду из памяти ID - декодировали EX - посчитали адрес в памяти (сложили регистр и оффсет) MEM - прочитали WB - положили в регистр **SUB:** IF - прочитали команду из памяти ID - декодировали EX - посчитали MEM - NOP WB - положили результат в регистр		
4	*Mips конвейер. Все стадии + пример. Все инструкции (придирается ко всему абсолютно) (x2 раза было!)*	1. IF - Instruction Fetch - считываем команду из памяти, передаем дальше 2. ID - Instruction Decode - декодируем инструкцию, т.е определяем что делаем, откуда берем данные и куда пишем. Считываем данные из регистров 3. EX - Execute - выполняем команду либо считаем адрес в памяти 4. MEM - пишем в память/ читаем с памяти 5. WB - Write-back - пишем в регистр Примеры смотри выше		
5	Что будет, если каждую стадию конвейера разбить еще на подстадии и до какого момента мы сможем так получать выигрыш?	Будем выигрывать по тактовой частоте и сильно проигрывать по энергопотреблению. В теории можно уменьшать время одного такта до тех пор, пока мы успеваем писать в регистры (которые у нас между стадиями) без кеков, и пока у нас ток от тактового генератора успевает поступать везде). На практике получим много хазардов и большое энергопотребление (см. вопрос 2)		
6	За сколько минимально можем выполнять операции без конвейера и на нем?	Ну, допустим у нас на конвеере 5 стадий. Latency (время выполнения самой первой команды) у нас 5 тактов, зато следующую команду мы получим уже на следующий (шестой) такт. Без конвеера мы мб можем выполнять одну команду за условные 3 такта, но и следующую мы получим еще через три ¯_(ツ)_/¯		
7	Что делает сложение на этапе MEM MIPS-конвейера?	NOP - точно		
8	Откуда IF берет команду?	Обращается к памяти по адресу лежащему в IP. IP инициализируется числом при старте системы, с этого числа начинается BIOS		
9	Что такое IP физически?	Специальный регистр		

2.4 Проблемы конвейера (hazards) и пути их решения.

1	Привести пример RaW хазарда + решение, Control + решение	**RaW:** ADD R1 R2 R3 SUB R4 R1 R5 Как фиксить: Forwarding, "это не баг, а фича", "а давайте конвеер постоит, пока досчитается, а потом начнем следующую инструкцию" **Control:** ADD ... JMP A SUB ... ADD ... <- вот это тоже начнет считаться, азаз A: XOR Как фиксить: исполнять JMP еще на стадии ID (ADD отвалится) + "это фича)))00"	тут есть подробно со слов Сказова	
2	Что такое структурные хазарды (нет, не предсказание переходов)?	Это когда у нас железо не может выполнять некоторые команды одновременно. Например если у нас одна шина и на инструкции, и на данные, то когда у нас выполняется LD/SD на стадии MEM, то мы не можем считать новую команду на стадии IF, потому что один канал до памяти и мы не можем одновременно обращаться к памяти по одному каналу. Как пофиксить: 1. Добавить железа. А именно, если у нас два канала до памяти, то такой баги не произойдет(гарвардская архитектура поэтому и быстрее). Опа, а кэш первого уровня обычно как раз и делится на команды и данные 2. сказать в документации "ну соре, команда чтения/записи занимает много времени" 3. Опять же, ID поймет, что на IF ничего разумного не пришло в этот момент и кинет NOP	(под редакцией @wily_tiger для обсуждения пишите тут в комментах или в тг)	
3	Могут ли возникать WAR и WAW конфликты на MIPS?	НЕТ. Почему? Для них вроде нужно изменение порядка выполнения команд, т.е. когда команды выполняются не последовательно, поэтому они возможны на Superscalar с OoO, но невозможны на MIPS и на суперскаляре с InO/InO		
5	**Как именно устроен forwarding? (видимо на уровне реализации, нужен гугл)**	Рискну предположить, что есть регистры, число которых равно мощность множества {EX, MEM, WB}, и после вычисления результат кладется в регистр, причем эти регистры устроены по принципу очереди, и, возможно, когда ID видит, что досчитать результат и вернуть его в нормальные регистры не получится, меняет адрес аргумента на адрес в этой типо очереди.	Преположу что ID видит что нам нужна информация, которая будет доступна через forwarding. Например есть у нас регистры промежуточный после EX. Соединяем их обратно с EX. Тогда, когда нам в EX нужны те значения, которые еще не лежат в основных регистрах, мы идем в промежуточный регистр после EX и берем значения оттуда. За это все отвечает ID, он умный (@wily_tiger)	
6	Кто кидает NOP?	это может делать стадия ID. Или, как в MIPS - не делать. И тогда либо программист сам ручками добавляет NOP, либо сам дурак		
7	Код WaR и WaW	в мипсе таких хазардов нет (см вопрос 3)		

2.5 Суперскалярная и VLIW архитектуры.

1	Плюсы минусы VLIW и суперскалярной архитектур	VLIW: - Программист должен сам заботится о оптимальном заполнении конвееров - Поддержка старых архитектур становится невозможной (например, добавим еще один конвеер, и все, непонятно что куда распихивать) - хороший компилятор написать ну очень сложно + Относительно просто(ну по сравнению с суперскаляром) + Меньше места занимает + Проще и дешевле + Так как работает с кодом компилятор он может скомпилировать лучше, чем Scheduler, поскольку у компилятора в распоряжении сильно больше времени и памяти	Суперскалярная: Если планировщик умный, то: + Runtime оптимизация программы на основании данных кеша и чего нибудь еще + Оптимальное использование ресурсов(если как в шахматах считать на пару ходов вперед) + NOP-ы генерирует сам планировщик -> больше команд влезает в кэш. - ХААЗАААААААРДЫ! Много Хазардов. (если ООО) - Планировщик дорогой по энергии(сам ничего полезного не считает) - Планировщик ограничен по памяти, поэтому может работать хуже чем компилятор (которому доступна вся оперативка)	- последний плюс VLIWa потенциально некорректен. он сказал что у влива есть больше времени и памяти. - У суперскаляра и VLIWa одинаковый набор хараздов: RaW WaW WaR. Это их общий минус, и его надо убрать	
2	Пример работы со VLIW				
3	Привести программу для VLIW и суперскаляра, которая считает сумму чисел в восьми регистрах	код для суперскаляра: ADD R1 R1 R2 ADD R3 R3 R4 ADD R5 R5 R6 ADD R7 R7 R8 ADD R1 R1 R3 ADD R5 R5 R7 ADD R1 R1 R5 ну и можно выгрузить результать из R1 куда-то в память командой LD, для влива тот же прикол с параллельностью, только NOP надо самому писать (именно поэтому я написал суперскаляр, хех(0))			
4	Каким образом происходит планировка при компиляции, почему это плюс?	Это плюс, потому что у компилятора сильно больше времени и ресурсов на планирование.		-Что значит "каким образом"? Просто компилятор берёт и переводит программу во VLIW архитектуру? \\ну берет и просчитывает как раскидать по доступным конвеерам	
5	Плюсы динамического планирования? Или пример, когда динамическое лучше (он же может такое спросить?)	Планировщик знает про железку сильно больше, чем компилятор. А еще он может подстраиваться под ситуацию (компилятор один раз скомпиллил, и все). Например, при кеш-промахе планировщик знает, что ничего связанного с текущей командой запроса к памяти ему ближайшие 100 тактов не светит, и поэтому перестраивается и считает что-то другое.			
6	Может попросить нарисовать обе архитектуры.	тык			
7	Попросил написать код для VLIW и Superscalar, который выполнится по-разному, но выдаст один результат.				
8	Почему код команд более компактный в суперскалярной архитектуре	Потому что не храним NOP в коде.			
9	Что быстрее?	Superscalar т.к. код получается компактнее и лучше кэшируется			
11	Код WaR WaW	WAR 1) MUL R1, R2, R3 2) XOR R6, R1, R5 3) ADD R4, R2, R3 4) SUB R5, R4, R5 Поскольку операция MUL очень тяжёлая, а XOR зависит от MUL, то вначале выполнятся 3 и 4 операции, и в XOR мы будем уже юзать новое значение аргумента WAW 1) MUL R1, R2, R3 2) ADD R4, R2, R3 3) SUB R1, R4, R5 Вначале выполнятся операции ADD и SUB, а потом в R1 запишется результат умножения		А тут точно нужны 3 и 2 команды в war и waw соответственно? Без них разве не будет хазарда? Я просто скопипастил код Зайки WAW точно не будет	
2.6 Процессоры: общего назначения/потоковые, ядра/многопроцессорные системы, одновременная многопоточность (SMT, HT)					
1	Чем smt лучше суперскаляра?	Быстрее [0.5; 2] раза а в среднем на 20-30%. Позволяет эффективнее юзать железо. Утверждается что железо в любом случае используется НЕ хуже, чем суперскаляром			
2	Зачем вообще делать потоковый процессор?	Потому что это выгоднее, когда у нас много независимых задач, которые можно легко разбить на треды, и нам всё равно, когда посчитается один тред, важно только насколько быстро посчитаются все треды.			
3	Чем отличается многоядерность от многопроцессорности?	У каждого процессора свой контроллер памяти. У ядер общая память(ну, исключая низкий кеш и регистры)			
5	Почему задачи, решаемые на потоковом процессоре, надо решать именно на нем, а не на ЦПУ?	Задачи, решаемые на потоковом процессоре очень хорошо параллепятся. И это их свойство потоковые процессоры используют лучше, чем CPU. (т.к. у потоковых процессоров много простых ядер, на каждое из которых можно ещё и кучу тредов закинуть)		Вроде бы все эти вопросы имеют почти один и тот же ответ	
6	Когда SMT бывает вреден?	если у нас один thread еле влезал в кэш, а мы решили запустить не 1, а 2, то они будут плохо кэшироваться и всё будет работать медленнее			
7	Что такое Hyper-threading? Чем отличается ядро с HT от ядра без него?	См. SMT(пункт 13). HT это патент Intel на технологию SMT в HT ядра в 2 раза больше регистров + в планировщик добавили логики, чтобы он брал инструкции с разных тредов		в HT ядра в 2 раза больше регистров, умный планировщик. хз верно нет "в планировщик надо добавить логики" слова Скхва, и про регистры он тоже говорил, так что ты прав	
8	Что такое потоковый процессор? Почему мы используем их в видеокарте, а в обычном процессоре почти нет?	В обычном процессоре это невыгодно. Т.к. большую часть программ нельзя разбить на очень-очень-очень много тредов. Получается что запуская что-то с двумя-тремя тредами на хилых ядрах потокового процессора очень сильно проигрываем по скорости. (вспомним, что один тред нельзя выполнять на разных ядрах)		Вроде бы все эти вопросы имеют почти один и тот же ответ	
9	Когда мы получаем преимущество при использовании SMT?	Когда 1) Число виртуальных ядер больше числа физических(иначе зачем оно надо) 2) Наша задача либо хорошо параллепится, либо ее надо считать много раз и это можно делать на разных ядрах 3) Каждый thread должен быть написан "не очень хорошо", чтобы оставались свободные конвейеры, которые планировщик будет забивать задачками от других thread'ов + треды должны не подраться за ресурсы (кэш)			
10	Необходимые условия для работы HT. Какой планировщик больше всего выигрывает от HT?	Не получится без многопоточности, нужно писать правильные программы. Выиграет тупой, т.к. почти за бесплатно получит независимые инструкции		вопрос 15 связан	
11	Какие технологии GPGPU есть? Что было до CUDA/OpenCL	Есть CUDA - платформа и апи от NVIDIA. Работает только с их картами. Код выглядит красивенько, потому что вся магия с видеокарточкой прячется в библиотеку. Есть OpenCL - открытая платформа. Поддерживается зелененькими карточками, но плохо (потому что а кто тогда будет использовать CUDA). Код выглядит как ну такое, потому что магию приходится писать ручками. До этого брали и рисовали два полигона, создавали текстуры из массивов данных и шейдерами считали из этих текстур новую. Потом эту новую текстуру считали массивом выходных данных.			
12	Почему нельзя написать ось под видюхи?	Потому что у GPU нет "обвязки" из вспомогательной логики, которая есть в CPU, например системы разделения памяти, т.е. каждое з ядер не может разделять память между thread'ами и не защищает память одного thread'а от модификации другими.		а смысл??? Или тут спрашивает про то, почему нельзя выкинуть проц??	
13	Что такое SMT?	SMT это когда у чипа есть реальные ядра, и виртуальные. Так как планировщик ядра не может выжать 100% забивки конвееров, то делают такой финт: давайте удвоим число регистров, и запустим два потока(треда) вычислений. А что, они же по умолчанию независимы, так что планировщику проще, и железка пашет			

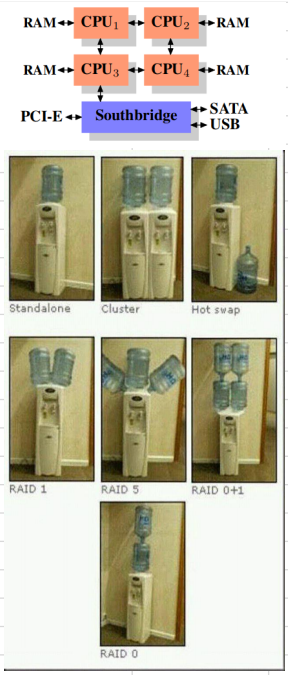
Раскраска по авторитетности источника ->>>>	смело можно использовать (официальная документация и/или советовал СККВ)	Литература (лежит в папке литература на диске)				
	почти можно доверять (не оф документация, содержащая догадки, но нормальная, а еще нормальные конспекты)	Цифровая схемотехника	<- простое			
	большей частью адеватно (более-менее адекватные таблички прошлых лет)	Computer Architecture: a Quantitative Approach	<- крайне рекомендую приложения A, B и C			
	полуправда	Таненбаум	<- можно найти процентов 40 от билетов			
	не проверялось					
	другое					

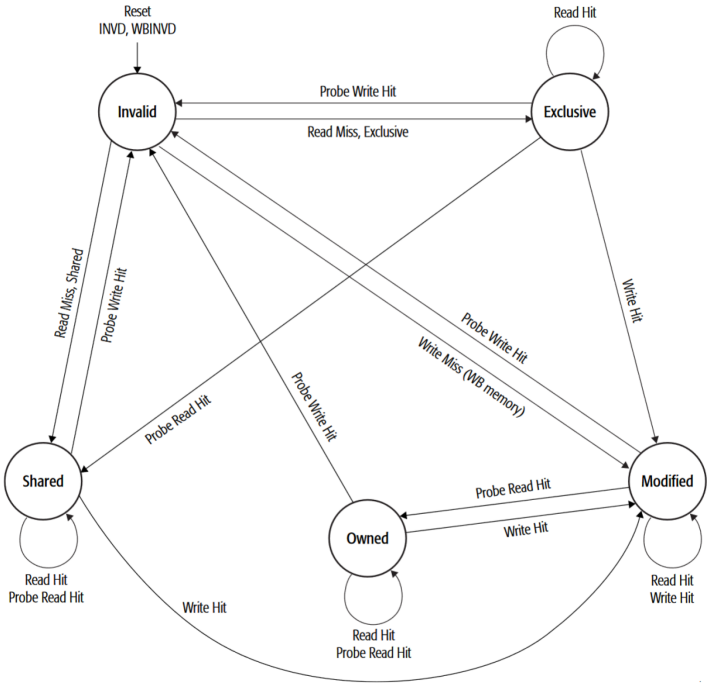
Доки	Другое:	Статейки (английский - это судьба (ц))	Презентации	Технические документации	Видосики	СТАТИСТИКА
Самая важная дока	Симулятор схем	***ОЧЕНЬ ИНТЕРЕСНАЯ СТАТЬЯ***	Про MIPS конвейер (на английском, но с картиночками)	Описание ISA ARMv8-A	HDD	**y2015**
безбажные конспектики	Рисовалка схем	***БЕСКОНЕЧНАЯ МЫШКА***		Описание процессора ARM Cortex-A76 (например, на		**y2016**
дока препода 34-35 (есть картинки, но много багов	DJ Zayka	про тайминги оперативки		DDR		**y2017**
конспекты Жени	представление целых чисел в компудактере	оч интересные 238 страниц про ассемблер и конвейеры различных цпу		Память		**y2018**
билеты by y2017	представление вещественных чисел в компудактере	Референс по AMD64 (на страницах 221-223 в pdfке есть подробная инфа о MOESI)				
еще билеты by y2017	курс на coursera, если проебали пару про стек, reg-reg и kek-kek, тут почти 1 в 1	глава из книжки про триггеры и все такое				
		про счетчики				

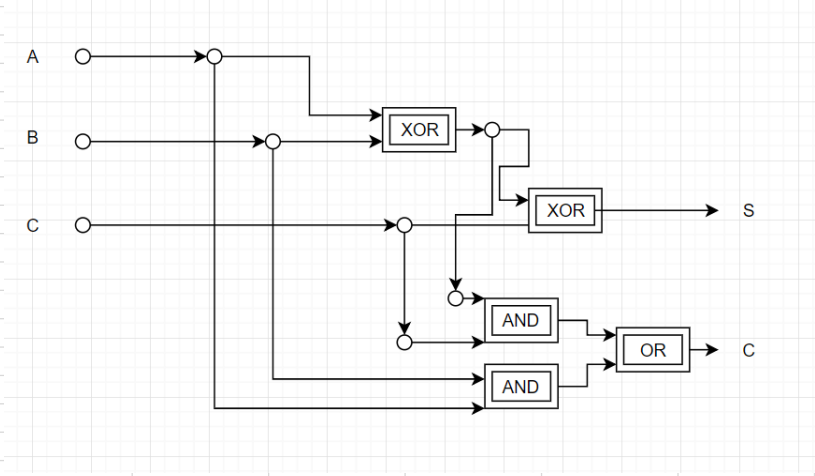
1.1 Элементная база вычислительной системы: логические элементы, триггеры			
1			Поток из гуглоформ
2			Если в sdram повысим тактовую частоту в два раза, то будет ли это эквивалентно ddr?
3			1. Зачем нужен кэш? 2. Почему именно с кэш-линиями работаем? 3. Почему раньше не очень использовали многоядерность? 4. Что такое HT и почему это круто?
1.2 Оперативная память: статическая/динамическая, организация.			1.5) Чем плох msi, как это решает mesi? Как решить без протоколов? Напиши код на асм, который без протоколов приведет к неверному результату. 2.6) Что такое HT и для чего нужен? Когда HT ускоряет? В чем разница потокового процессора и обычного? Напиши код на асм который хорошо разобьется на треды. Требовал именно асм, с++ не разрешил. Press F.
1.3 Оперативная память: характеристики, типы динамической памяти, NUMA.			Назвать все принципы Фон-Неймана (докапывался ДО ВСЕГО, учите слово в слово), плюсы и минусы программного и аппаратного управления Про протоколы когерентности толком ничего не спросил (на 2.1 норм поразвлекался потому что), разве что как мы можем поддерживать когерентность и зачем она нужна, ну и чуть послушал про MSI.
1	Если в sdram повысим тактовую частоту в два раза, то будет ли это эквивалентно ddr?	Нет. Потому что в ddr мы выкидываем в два раза больше данных, а команд и прочего такое же ч	Что такое сектор Как происходит обработка ошибок Как хуже поцарапать оптический диск Что такое ISA Что такое микроархитектура Приведи пример микроархитектур, которые мы проходили на уроке
2	основные для программиста характеристики RAM	Скорость доступа, скорость передачи, объём памяти (Скаков принял это)	Как получается F Код не работающий без когерентности
3	Четыре фактора, от которых зависит скорость передачи?	Может ширина шины, частота шины, тайминги памяти и DDR/не DDR? Скаков принял и досказал: ширина шины, частота памяти, сколько каналов, также сказал, что переход от SDRAM к DDR также увеличил скорость передачи засчёт увеличения внутренней шины	1.6 Что такое сектор? Почему это минимальный объём данных? Какая служебная информация хранится в каждом секторе в каждом типе носителей? 2.4 Привести код хазардов. На какой именно стадии происходят ошибки в каждом? Как их решать?
1.4 Кэш-память.			Все ответы верны, верифицированы непосредственно Скаковым. 2.1 В: В чем отличие программного от аппаратного управления? О: В аппаратной архитектуре отсутствуют команды как таковые, её поведение определено конструкцией. В: Пример программного нарушения принципа последовательности. О: VLIW. По самому коду видно, что команды будут выполняться параллельно. 1.3 В: Какие есть характеристики памяти? О: Объём, время доступа, время передачи. В: Что влияет на скорость передачи? О: Частота, стандарт памяти, ширина внешней шины.
	Пример машины, без кеша, на которой нет проблем по времени с доступом к памяти	Может видеокарты, у них его нет в обычном понимании Кого нет? кэша? но проблема с временем доступа есть. У нас куча тредов, после запроса треда к памяти мы отключаем текущий и кидаем следующий на выполнение, поэтому вообще не тратим время из-за памяти	1) основные для программиста характеристики RAM 2) от чего зависит скорость передачи в RAM 3) что сделали в FPM 4) все типы хазардов на MIPS с примерами кодов и рисунки
	Почему именно с кэш-линиями работаем?		Как выглядит инвертор построенный на транзисторах Зачем нужна многопортовая память Почему память в виде таблицы, а не линейная Код для суперскаляра и vliw В чем преимущества vliw перед суперскаляром
	Зачем нужен кэш?		написал в самом начале
1.5 Протоколы когерентности кэш-памяти			В адресности памяти нельзя говорить про аккумуляторную систему(это совсем не то), что происходит при write back? Пример машины, без кеша, на которой нет проблем по времени с доступом к памяти
	Напиши код на асм, который без протоколов приведет к неверному результату.		В чем минусы WB Код для vliw, суперскаляр
1.6 Носители информации: магнитные, оптические и на основе флеш-памяти, RAID			Инвертор на транзисторах. Плюсы и минусы статической и динамической памяти. Почему мало юзаем многопортовость. Дата хазарды помимо raw.
	Что такое сектор? Почему это минимальный объём данных? Какая служебная информация хранится в каждом секторе в каждом типе носителей?		1.3) 3 основные характеристики DRAM, от каких более глубоких хар-ик зависит скорость доступа. 2.6) Из-за чего началось массовое производство многоядерных процессоров?
2.1 Архитектура фон Неймана и её альтернативы			Не помню особо. 1. Зачем нужна синхронизация? 2. Расскажи подробно про каждую стадию в мипс 3. Нарисуй парочку триггеров И т.д. и т.п.
2.2 Архитектура набора команд (ISA) и микроархитектура			1. От чего зависит скорость доступа (от таймингов, от частоты и от стандарта памяти - он сам ответил) 2. Напиши код для VLIW и суперскаляра, который делает одно и то же, но разный в плане выполнения
	Приведи пример микроархитектур, которые мы проходили на уроке		Слишком подробно написал в прошлой форме.
2.3 Конвейерная архитектура, конвейер MIPS			Что такое ISA? Что она определяет? Что такое микроархитектура? Пример с пары (кроме MIPS)? Кэш на запись, кэш на чтение, их работа, их + и -, ассоциантивность (полная, 1) сравнить их.
2.4 Проблемы конвейера (hazards) и пути их решения.			Сравни магнитные и флеш накопители (в черной коробке), что такое сектор, что такое ISA и микроархитектура (+ примеры с лекции), cisc и risc
	Рассказать про всевозможные хазарды, как они решаются, привести код для всех хазардов (даже для структурного). Скакову не понравилось, что если не перекладывать за control-hazard на плечи программиста и самому фиксить баг с JMP, ID не игнорирует следующую за ним команду, а на EX посылает NOP, а декодирует JMP как NOP (хз, в чём разница, но Скакову не понравился мой ответ)		
2.5 Суперскалярная и VLIW архитектуры.			Написать код, который ломается без когерентности Как различить архитектуру Фон Неймана и Гарвардскую архитектуру визуально
			Разобрать команду на конвейре
	Почему раньше не очень использовали многоядерность?	(подтверждено отвечающими)не было нужного софта + (догадки) мб раньше еще могли улучшать одноядерный проц, поднимая частоту, а потом уперлись в энергию. Вроде он говорил, что многоядерность "с горя".	Четыре фактора, от которых зависит скорость передачи? Как написать разный код на VLIW и суперскаляре, чтобы он делал тоже самое?
	Что такое HT и почему это круто?	более полно загружаем конвееры	0) он не просил, но до выхода к нему нарисовал полевой транзистор, PMOS, NMOS, CMOS (NAND) и все триггеры 1) Зачем нужна синхронизация 2) Что такое ISA? Примеры ISA. 3) Что такое микроархитектура? примеры микроархитектуры.

	Напиши код на асм который хорошо разобьется на треды.	for (int i = 0; i < N; i++) {
		arr[i] += 2;
		}
		
		// R1 = N, R5 = arr, R2 = i, R3 = x[i]
		LD R1, N // R1 = N
		LD R5, ARR // R5 = arr
		MOV R2, 0 // R2 = 0
		L1:
		CMP R2, R1
		JGE END // if (i >= N) goto END
		LD R3, [R5 + R2]
		ADD R3, 2 // x[i] += 2
		ST [R5 + R2], R3
		ADD R2, 1
		JMP L1
		END:
		
		Вроде это аналог кода с лекции, только там было f(arr[i]) вместо arr[i] += 2, но это тоже должно биться
			Что такое сектора и для чего они нужны? Нужны ли они, если данные не портятся(нет необходимости в кодах коррекции)? Что такое микроархитектура? Какие мы проходили на лекциях? Что такое ISA? Какие ты знаешь?

	Из-за чего началось массовое производство многоядерных процессоров?		1.5 Спросил, в какие состояния может перейти S во всех протоколах, зачем нам O. спросил, как исправить это, не используя когерентность. Вроде правильный ответ сбрасывать кэш в первом цикле (надо заметить, что кэш сбрасывает не весь, а только у того ядра, на котором тред) 2.1 Докапывался до всего На адресности надо быть аккуратным. Вроде хорошо принял ответ: память состоит из одинаковых ячеек, адреса ячеек фиксированные, ко всем имеем быстрый доступ. На однородности памяти
			1.1 вообще ничего не спросил, посмотрел на картинки и сказал норм(я там нарисовал и, или-не и и-не на транзисторах и все триггеры)
			1.2: Используется ли двухпортовая DRAM? (Нет, тк DRAM используется только в RAM, двухпортовая сильно больше, дороже и больше проводков, вместо двухпортовости используют банки памяти => лови вопросы про анки памяти) Для чего используют двухпортовость? (ДП позволяет обращаться к двум! разным! (оба слова очень важны) строкам одновременно => меньше очередь запросов => выше скорость передачи) Почему используют банки памяти? (...=> меньше очередь запросов => выше скор
			1.3) 1)Какие характеристики динамической памяти (оперативки) влияют на её скорость доступа. Назвал частоту логики, tCL и tRCD. Ошибся, назвав ширину шины данных. За минуту так и не понял, чего не хватало. 2.1) 1) В каком случае выгоднее строить устройство на программном управлении, а в каком случае -- на аппаратном? Ответил, как это происходит в зависимости от тиража. 2) Стоимость чипа с аппаратным управлением не сильно ниже, чем стоимость чипа с программным управлением. Ес
			1.6: вопрос: чем схожи оптический диск с флеш памятью? (То что надо сначала удалить после записать уже не прокатывает, говорит, все так отвечают, а есть ещё одна схожесть, но я поплыл))) Потом про то, что такое RAID но я сам дурак читал русскую википедию, он не принял (не читайте русскую википедию, английский это судьба) 2.4 класека: просто рассказать все хазарды и как их решать







2. MOESI State Transitions