Конспект к экзамену по билетам (архаичные ЭВМ) 1-й семестр

Латыпов Владимир (конспектор)
t.me/donRumata03, github.com/donRumata03, donrumata03@gmail.com

Скаков Павел Сергеевич (лектор)
t.me/pavelxs

26 сентября 2022 г.

Содержание

1. Введение

Максимально сжатый материал: если читатель не знаком с курсом, возможно, стоит сначала изучить конспект Тимофея на Overleaf.

2. Названия билетов (ровно как в оригинале)

1. Устройство памяти
    1.1. Элементная база вычислительной системы: логические элементы, триггеры.
    1.2. Оперативная память: статическая/динамическая, организация.
    1.3. Оперативная память: характеристики, типы динамической памяти. NUMA.
    1.4. Кэш-память.
    1.5. Протоколы когерентности кэш-памяти.
    1.6. Носители информации: магнитные, оптические и на основе флеш-памяти. RAID.
2. Архитектура процессорных систем
    2.1. Архитектура фон Неймана и её альтернативы.
    2.2. Архитектура набора команд (ISA) и микроархитектура.
    2.3. Конвейерная архитектура. Конвейер MIPS.
    2.4. Проблемы конвейера (hazards) и пути их решения.
    2.5. Суперскалярная и VLIW архитектуры. Спекулятивное исполнение. Уязвимости классов Spectre и Meltdown.
    2.6. Многоядерные/многопроцессорные системы, одновременная многопоточность (SMT/HT).

Нужно ответить на два вопроса: по одному из каждой части. Пользоваться ничем нельзя, отвечать сразу.

3. Устройство памяти: О чём говорить при каждом из билетов?

3.1. Элементная база вычислительной системы: логические элементы, триггеры.

Европейские и американские обозначения логических элементов

Полусумматоры и сумматоры (медиана + XOR или два полусумматора + OR)

RS-триггер, каноническая и енаиболее эффективная версия — через два ИЛИ-НЕ.

Проблемы высоких частот и малого размера, «В Ethernet соотношение сигнал/шум — как разговаривать рядом с турбиной самолёта»

$\Longrightarrow$ Синхронная версия RS-триггера, D-триггер через «не» на входе.

3.1.1. MEM

Декодер 3to8 (для каждого из 8 выходов AND от трёх, возможно, инвертированных входов) $\Longrightarrow$ мультиплексор ($3 + 8 \to 1$) и демультиплексор ($3 + 1 \to 8$).

Из этого можно сделать модуль памяти «Mem» на 8 бит:

Входы: 3 адресных бита, R/W, C (clock), D (запись, если выбран режим W).

Выход: «Q» — если выбран режим R (не обязательно только в этому случае) — на нём значение, соответствующее биту по адресу $\overline{A_0 A_1 A_2}$.

3.1.2. JK-триггер

JK-триггер (Входы J, K, синхронный, $\Leftarrow$ умеет инвертировать состояние при двух единицах). Крафтится из двух RS-триггеров с тактированием в противофазе, на вход первого кроме оригинальных входов J, K через «AND» даётся то, что запомнил второй (то есть что было на первом до начала такта). В нормальной версии результат (Q и $\tilde{Q}$) берётся из выходов первого триггера.

3.1.3. Физические основы работы логических элементов

Нас интересуют в основном транзисторы.

Транзистор: затвор влияет на электрические характеристики перехода исток-сток. Биполярный: влияет **ток** через затвор (на самом деле, называется «база») Полярный: влияет **напряжение** на затворе

n-канальные полярные транзисторы. Исток — «внизу», то есть там напряжение ниже, чем на стоке. Состоит из

Напряжение считаем относительно истока. Подложку мы заземляем, чтобы не влияла на электрическое поле, которое создается остальными элементами. Если подать положительно напряжение на затвор (опять же, относительно истока), а в случае p-типа — отрицательное, притянутся электроны и образуется насыщенный электронами канал между двумя проводниками.

3.1.4. Логики

По многим причинам победила $CMOS$-логика, конструирующая только транзисторы (причём полевые) для конструкции элементов. Это позволяет минимизировать потребляемую энергию, так как ток течёт только при переключении. В отличие от полевых транзисторов (где он нужен через базу для поддержания открытого состояния), а также $NMOS$ логики, где он течёт по резистору в некоторых случаях (когда транзистор открыт).

В случае $CMOS$ логики расход энергии растёт при увеличении частоты почти линейно по понятным причинам.

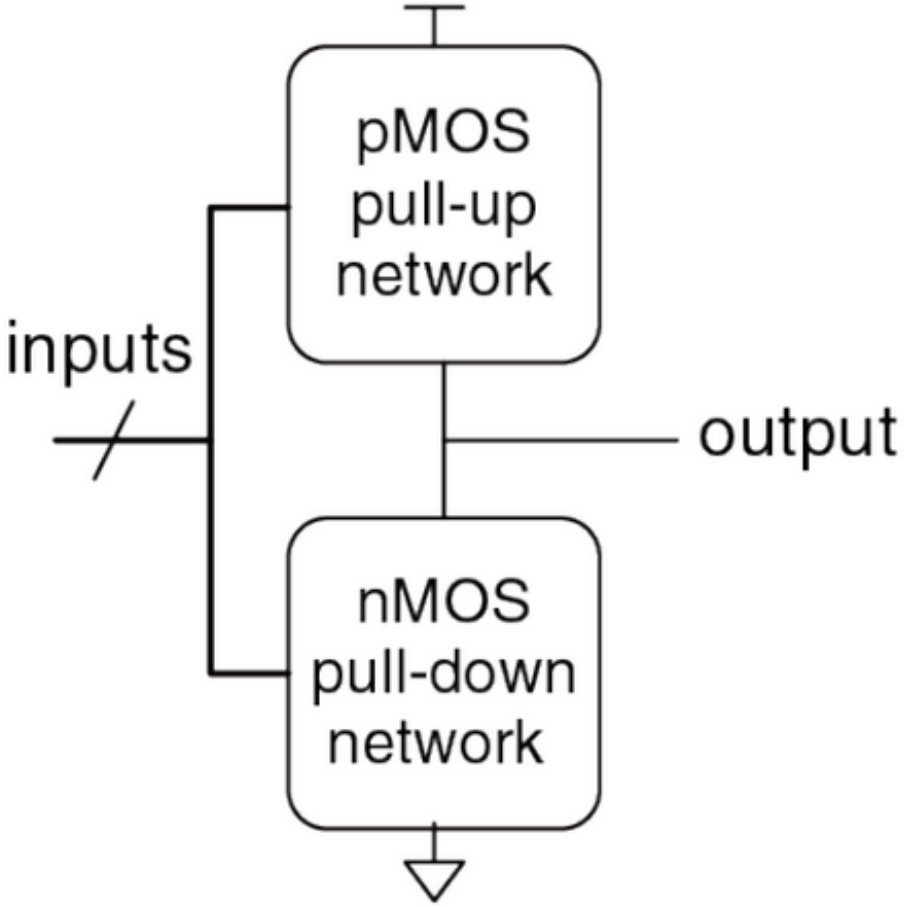

Рис. 1: Общий принцип построения схем в CMOS логике

В качестве элементарных частиц используем $NOT, NAND, NOR$ (нельзя сделать AND и OR эффективнее, чем соответствующий $NAND/NOR \circ NOT$)

Fun Fact: можно сделать NOR и $NAND$ на много входов эффективнее, чем просто внешне композировать (например, для трёх — 6 вместо 8-и).

3.1.5. Дребезг контактов

Кнопки из реальной жизни при нажатии не сразу устанавливается в новой состояние, а сначала колеблется. Это назвается Contact bounce (дребезг контактов). Причём чем старше и некачественнее контакты,

тем дольше будет происходить bouncing.

Есть несколько подходов к борьбе с ним. Бывает, программно, бывает аппаратно. Причём криме вопросов реализации возникают ещё и концептуальные. Проще всего реагировать на нажатие с запозданием: по истечении времени с начала или с последнего изменения. Другой вариант — реагировать как только произошло изменение после затишья, но после этого игнорировать дребезг, пока не установится. Однако, если у нас не просто бинарная кнопка с одним контактом «нажат/не нажат», а имеющая положения «не нажата, положение 1 и положение 2» и мы хотим на выход подавать бинарный сигнал в виде последнего «кас´анного» контакта, то всё гораздо очевиднее и у нас есть ультимативное решение через триггер и, возможно, транзистор. Не забыть про подтягивающие резисторы.

3.2. Оперативная память: статическая/динамическая, организация.

Папмять располагают в 2D решётчатой структуре.

Количество проводов $\propto cols + rows \Rightarrow O(\sqrt{memory\_size})$

Row отвечает за выбор ряда. Если он ноль, ячейка вообще не работает.

Если подаётся $true$, ячейка открывается. Можно считывать информацию с соответствующих Col и $\overline{Col}$. Также можно записывать на эти входы.

Статическая память строится как

	Статическая память	**Динамическая память**
Скорость	быстрая	медленная
Количество транзисторов	Много (=6)	Мало (обычно один + кондерсатор)
Надёжность	Наличествует	Постоянно дегенерирует $\Rightarrow$ надо регенерировать
Примеры	Регистры, кэш	Оперативная память, видеопамять

Параметры, по которым оценивается память:

Объём,

скорость доступа («latency»),

скорость передачи («throughput», пропускная способность)

Уязвимость Raw-Hammer: если очень долго обращаться к ячейкам, соседним с данной, можно установить её в ноль.

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

Адресуемся в порядке: старшие биты адреса — номер строки, младшие — номер столбца.

Типичные интерфейсные выводы оперативки:

·

Порядок работы с памятью:

1. Процессор сообщает о таймингах
2. Всё время идет синхронизация
3. Выбор строки $\Rightarrow$ модуль динамической памяти заносит строку в статическую
4. Работа со строкой: запрос адреса
5. После передачи данных и выбора следующей строки происходит «закрытие» старой — в динамические ячейки вновь записываются данные из статической строки.
6.

Что какие тайминги означают?

Аббривеатуры:

- RAS — Row Access Strobe — сигнал выборки строки
- CAS — Column Access Strobe — —"— столбца
- Precharge — «закрытие строчки»
- CL — CAS Latency — с получения адреса столбца до ответа
- t_{RCD} — RAS to CAS Delay — открытие строки
- t_{RP} — Row Precharge — закрытие строки

- t_{RAS} — Минимальное время активности строки

Производители обычно указывают либо 4 тайминга:

$$CL - t_{RCD} - t_{RP} - t_{RAS}$$

Либо только CL.

Причём измеряется это в количестве тактов при какой-то заданной частоте.

Если обращаемся к памяти редко, задержка будет $t_{RCD} + t_{CL}$. Если часто, причём в случайные строки, то $t_{RAS} + t_{RP}$

Улучшенные стандарты памяти:

- FPM DRAM: можно работать с одной строкой сколько угодно, не закрывая её
- EDO RAM: можно держать столбец не до самого конца чтения, а отпускать, пока ждём CL
- BEDO DRAM: выдаёт вместе с ячейкой памяти выдаётся 3 следующих (если кончается на $...00$)
- SDRAM: модуль и контроллер памяти работают синхронно + за один запрос отправляем не 32 бита, а 64
- DDR SDRAM: передаём данные по двум тактовым сигналам в противофазе (и по восходящему фронту, и по нисходящему)
- DDR2 SDRAM: увеличили тактовую частоту в 2 раза, а внутри стала больше только передающая шина
- DDR3 SDRAM: Ещё в 2 раза. Итого — 512 бит
- DDR4 SDRAM: мелкие технические улучшения

GDDR: максимизируем скорость передачи, на доступ забиваем

Существуют двух- и более портовые ячейки памяти. Есть тот же бит, но много интерфейсных транзисторов. На каждый — новые «Col_k», «Row_k»

На практике используется редко, так как сложно изготавливать.

Многобанковая vs. Многоранговая vs. Многоканальная памяти.

Многобанковая: одному модулю памяти соответствует несколько банок — матриц Многоранговая: в одном физическом блоке (слоте) есть несколько модулей Многоканальная: параллельные слоты под память, для $fine - grained$ параллелизма нужны похожие модули.

3.4. Кэш-память.

Это такой большой костыль, возникший из-за того, что процессоры стали быстрее памяти. Находится близко к вычислительным ядрам, на кристале процессора. Делает то, о чём говорит его название: кэширует запросы к памяти. Имеет меньший размер, чем оперативка, но работает быстрее.

Работа основана на идеи адресно-временной локальности.

«look aside» vs. «look through»

«write through» vs. «write back»

3.5. Протоколы когерентности кэш-памяти.

Зачем нужно поддержание когерентности? (пример с DeadLock-ом из-за отсутствия когерентности). Можно решать на уровне пргограммиста, можно — на уровне железа. Второе лучше, так как может учитывать контекст исполнения. То есть сбрасывать кэш может быть не нужно, если, например, он сейчас только у этого ядра.

Обозначения сигналов:

- PR — это ядро изволиои хотеть читать
- PW — это ядро изволиои хотеть писать
- BR — этот кэш отправляет по шине запрос на чтение
- BRfO — этот кэш отправляет по шине запрос на чтение + получение Ownership
- BU — этот кэш уже имеет данные, но хочет стать единственным владельцем, говорит другим «кыш»
- BR — этот кэш получает по шине от другого кэша запрос на чтение
- BRfO — этот кэш получает по шине от другого кэша запрос + получение Ownership

- BU — **другой** кэш уже имеет данные, но хочет стать единственным владельцем, говорит другим «кыш»
- Data — ответ данными
- DataW — ответ данными + запись в оперативку

MSI:

- M — Modified — уникальное изменённое состояние кэш-линии
- S — Shared — точная закэшированная копия оперативки
- I — Invalid — данная кэш-линия отсутствует в кэше.

Важно, что из кэша могут выселять, тогда при M мы записываем в память.

$MESI$ — решает проблему, что при любом PW из состояния S отправляем всем сигнал BU. Это забивает шину, хотя большая часть данных используется только в одном ядре.

Добавляется состояние E — «exclusive» — уникальная, но всё ещё копия оперативной памяти.

Тут становится важно, кто отвечает на запрос о BR. Если память, переходим в E, иначе — в S. И отвечать на запрос, конечно, становится обязательным.

$MESIF$. Добавляется состояние F — Forwarded. Передаётся как эстафета, живёт у последнего прочитавшего Shared. Если кто-то выкинул, то новым прижётся идти в оперативную память.

Другой вариант — $MOESI$. Состояние O — Owner — появляется, если кто-то хочет прочитать данные, а они Modified.

Раньше в $Intel$ был $MESIF$, в AMD — $MOESI$, теперь — неизвестно.

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

Дискреты

Cylinder-Head-Sector

В каждом секторе ≈512 байт. В каждом хранится адрес, CRC, а потом ещё ECC поверх всего.

HDD

LDA — Logical block adressing (теперь количество секторов уменьшают ближе к центру => адресацией занимается кто-то другой)

Эффективнее делать секторы побольше (сейчас — 4кБ), чтобы качественнее исправлять, но так как все захардкодили 512, диски притворяются, что у них столько и есть. Но если нужно весь сектор, плохо, так как читаем его 8 раз.

SMART — Self Monitoring And Reporting System

Shingled-винчестеры

Оптичекие диски

Секторы расположены в форме спирали, информация — есть дырка или нет, делать и проверять их наличие может лазер.

Бывает несколько слоёв: фокусировка лазера.

Audio CD: не нужны CRC и ECC.

и магнитные ленты.

Флеш-память: используются тарнзисторы с плавающим затвором, в которые можно заселять или выселять электроны. Для записи подаётся напряжение сток-плавающий затвор, что заставляет жлектроны переходить через тонкий слой диэлектрика.

RAID — комбинация дисков для увеличения скорости и/или отказоустойчивости.

4. Архитектура процессорных систем: О чём говорить при каждом из билетов?

4.1. Архитектура фон Неймана и её альтернативы.

1. Двоичная система счисления (главное — не 10-ная)
2. Программное управление
3. Наличие чётких (абсолбтных и постоянных) адресов (vs. Машина Тьюринга)
4. Последовательное выполнение инстукций (vs. Конвейер)

5. Общая память для инструкций и данных (vs. Гарвардская)

4.2. Архитектура набора команд (ISA) и микроархитектура.

Стековая, кумулятивная, Reg-Reg, Reg-Mem.

5. Appendix

5.1. Получение каждого хазарда

5.2. Data хазарды

Вознкают, если новые блихко расположенные инструкции зависимы от результатов предыдущих.

5.2.1. RaW Data

Сначала какая-то инструкция пишет, потому другая должна прочитать, что она записала, но читает старые данные $\Rightarrow$ hazard.

Пример: из $MIPS$.

5.2.2. WaR Data

Сначала какая-то инструкция читает, потому другая должна записать, но записывает раньше и первая читает новые данные вместо старых $\Rightarrow$ hazard.

Пример: из SuperScalar (OoO/OoO).

DIV R1, R2, R3

ADD R4, R1, R5

SUB R5, R6, R7

Если SS исполнит SUB раньше DIV (оно ведь от него не зависит?!), а это сделать хочется, так как умный конвейер занят, то ADD считает новые данные вместо старых, так как выполнится после DIV, то есть и после SUB.

Одно из решений — аппаратно изменять соответствие между пользовательскими регистрами (R_i) и аппаратными.

5.2.3. WaW Data

Запись данных происходит не в том порядке, в результате оказывается не то значение.

Пример: из SuperScalar. Берём наивный InO/OoO.

Div R1, R2, R3

Add R1, R4, R5

5.2.4. Control

Возникает из-за инструкций передачи контроля. Если выполнять JMP на EX, то на конвейер зайдёт ещё две команды, что плохо. На $MIPS$ решают переносом его в ID + докуменацией, что есть Delay Slot.

5.2.5. Struct

Не хватает скорости памяти. Одновременно к ней обращаются IF и MEM. Побеждает MEM.

Решения: либо успользовать гарвардскую архитектуру, либо ускорять память, либо по конвейеру отправляются NOP-ы.