1) Физические основы ЭВМ.

a) Если с точки зрения компьютера у нас два сигнала: 0 и 1, то в реальной жизни всё не так радужно. Для каждого логического элемента/схемы есть диапазон значений, который этот элемент трактует как 0, и соответствующий диапазон для 1. 0 - это от Vgnd (напряжение земли) до Vil (input low - нижнее входное значение). 1 - от Vih (input high - верхнее входное) до Vdd (напряжение источника). На выходе мы тоже получаем не абсолютные значения, а Vol(output low - нижнее выходное) в качестве 0 и Voh(output high - верхнее выходное) в качестве 1.

b) Внутри компьютера всё завязано на полупроводниках. Они бывают двух типов: p и n.

c) <неинтересная физика, могу обмануть>В n-типе у нас есть 4-валентный кремний, в который добавили 5-валентный мышьяк. “Лишний” электрон мышьяка не образует связей с кремнием и остаётся свободным. При комнатной температуре часть атомов мышьяка лишается электрона и становится положительными ионами, который, однако, не может захватить электрон у соседнего кремния, так как последний тупо сильнее держится за своё богатство. Поэтому дырки и течения электронов не возникает, в то время как оторвавшиеся электроны являются электронами проводимости.

d) P-тип, наоборот, образуется, когда мы к 4-валентному кремнию добавляем что-то 3-валентное (у Ромы на слайде алюминий). Тогда у нас образуются “дырки”, в которые природа хочет запихнуть электрончик.

e) Когда мы соединяем полупроводники n и p типов, на стыке образуется запирающий слой. Если мы теперь к этом подключим источник тока так, чтобы плюс источника подсоединился туда, где у нас много электронов, то электроны проводимости и “дырки” начнут удаляться друг от друга, а значит и от слоя, тем самым увеличивая толщину слоя и увеличивая сопротивление проводника. Если же теперь поменять полюсы источника местами, то “дырки” и электроны устремятся навстречу друг другу, уменьшая запирающий слой и сопротивление проводника.</неинтересная физика, могу обмануть>

f) Помимо транзисторов ещё есть конденсаторы, которые накапливают заряд тока. Выглядят, наверное, как две пластины, между которыми слой диэлектрика (обычно воздух). Но это неинтересно, вернёмся к нашим транзисторам.

g) Транзисторы раньше использовались биполярные, но они уже устарели. На смену пришли металл-оксид полупроводники (сокращённо на русском МОП, на английском - MOS). Транзисторы n-типа aka nMOS - являются подложкой p-типа, в которой расположены области n-типа. Транзисторы p-типа aka pMOS - наоборот. Между двумя областями: истоком (source) и стоком (drain) находится затвор (gate). Прикладывание напряжения к

затвору создаёт или убирает электрическое поле, которое обеспечивает переход электронов от истока к стоку через канал (channel).

h) Отличие nMOS от pMOS. nMOS не пропускает ток при отсутствии напряжения на затворе и начинает пропускать, если это напряжение появится. pMOS, наоборот, пропускает ток в обычном состоянии и говорит “ТЫ НЕ ПРОЙДЁШЬ”, если подвести ток.

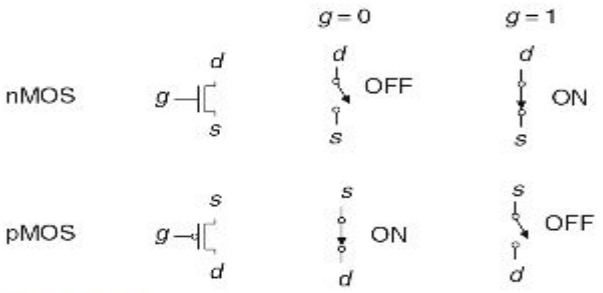


Рис. 1.31 Модели переключения полевых МОП-транзисторов

i) Из транзисторов собирают основные логические вентили: НЕ (NOT); И-НЕ, которое вообще-то НЕ-И (NAND); ИЛИ-НЕ, которое НЕ-ИЛИ (NOR). Схема НЕ: Схема И-НЕ: Схема ИЛИ-НЕ:

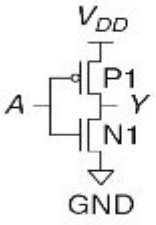


Рис. 1.32 Схема вентиля НЕ

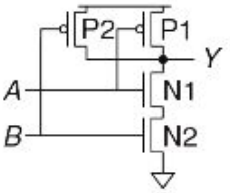


Рис. 1.33 Схема вентиля И-НЕ с двумя входами

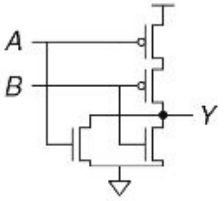


Рис. 1.36 Схема вентиля ИЛИ-НЕ с двумя входами

j)

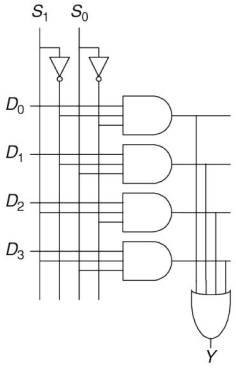
2) Функциональные схемы

a) Функциональных схем два вида: комбинационные и последовательные.

b) Комбинационные (комбинаторные) - схемы, которые выдают значение в зависимости только от входных значений. Вот они слева направо: мультиплексор, дешифратор, демультиплексор, сумматор.

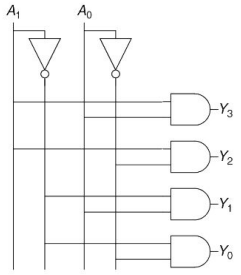
   i) Мультиплексор - n управляющих входов, 2^n входов данных, 1 выход. Значения управляющих входов являются двоичным

представлением входа данных, значение с которого передаётся

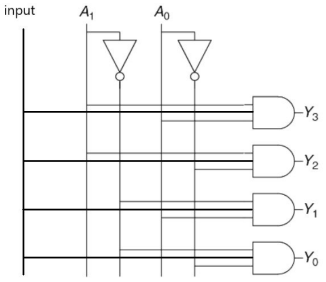


на выход.

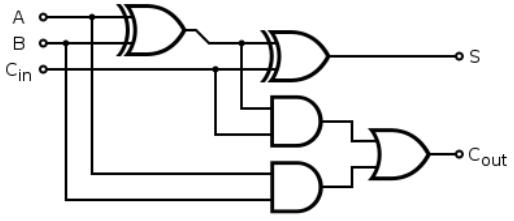
ii) Дешифратор - n входов, 2^n выходов. На входы подаётся в двоичном виде номер выхода, на который подастся 1.



iii) Демультиплексор - 1 вход данных, n управляющих входов, 2^n выходов. Управляющими входами определяется выход, на который передаётся значение входа данных.

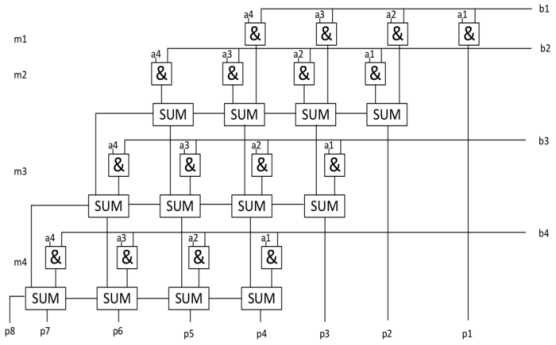


iv) Полный сумматор - говорит сам за себя. Принцип: два XOR'а, два AND'а и OR. Младший бит у нас 1 в случаях, если входные биты у нас одинаковы и бит переноса 1 или если бит переноса 0 и входные биты различаются. Старший бит у нас 1, если входные биты различаются И бит переноса 1 ИЛИ если оба входных бита .



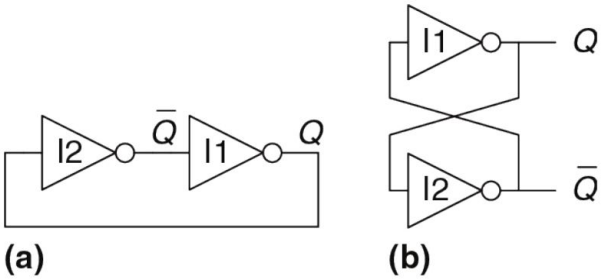
v) Так же ещё существует каскадный сумматор - это просто много полных сумматоров подряд. N сумматоров - можем складывать n-битные числа.

vi) Матричный умножитель - умножает столбиком, кек. Вот вам схемка, на всякий.

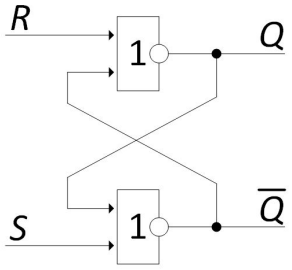


c) Последовательные схемы - зависят не только от входных данных, но и от предыдущего состояния схемы.

i) Самая простая штука - два инвертора. Хранят один бит информации без возможности изменения, но обычно можно ручками туда что-то ткнуть.

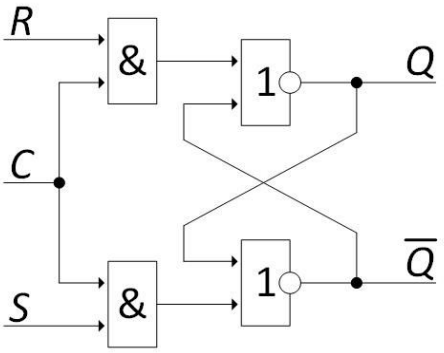


ii) RS-триггер aka защёлка(?). Имеет два входа: R(eset) и S(et) - и два выхода: Q и инвертированный Q. При подаче обоих нулей триггер выдаёт то значение, что было до этого. При подаче 1 на S триггер выдаёт 1 на Q, при подаче 1 на R триггер выдаёт 0 на Q. При подаче двух 1 у нас шляпа.



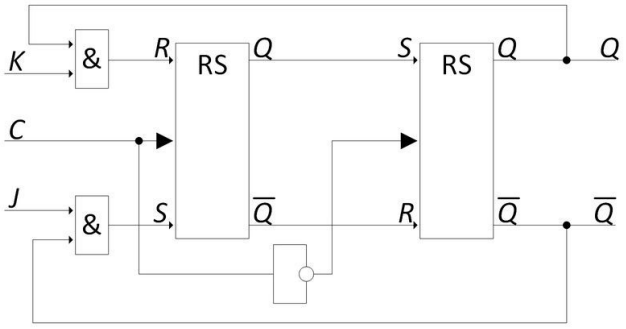
iii) Есть проблема с тем, что мы живём не в идеальном мире (плак-плак) и сигнал по проводам приходит не одновременно. Из-за этого мы можем получить не то, что нам хочется. Решение: добавим провод синхронизации. Если на синхронизации 0, то мы выдаём предыдущее значение, как только синхронизация 1, начинаем работать по прежней схеме. Частоту смены синхронизации стоит делать такой, чтобы оба сигнала однозначно успевали.При двух единицах у нас всё ещё

гроб-гроб-кладбище-кхм.

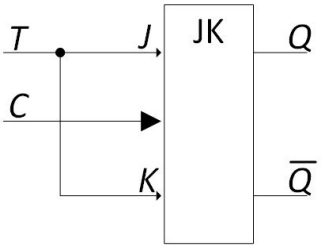


R	S	Q
0	0	Q
0	1	1
1	0	0
1	1	Forbidden

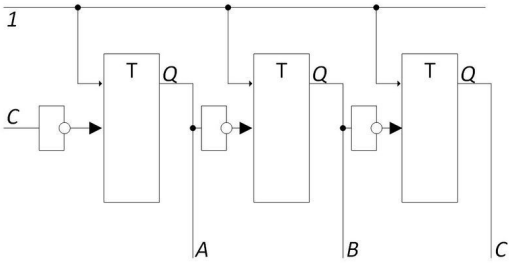
iv) J(ump)K(ill)-триггер aka прыгай-убивай - улучшенная версия RS-триггера. Jump = Set, Kill = Reset, а при подаче двух единиц значение Q (и соответственно не-Q) инвертируется. Если убрать из него синхронизацию, то подача двух единиц запускает кошмар эпилептика в виде постоянного переключения. Если синхронизацию (это тот инвертор посередине) перенести в начало, то существует секретная комбинация, которая ломает бедный триггер: 01 (вывели Qпред - поставили внутри 0), 11 (поставили на выход 0, на спаде поставится 1 внутри), 00(мы просили его вывести 0, а он нам выведет 1).



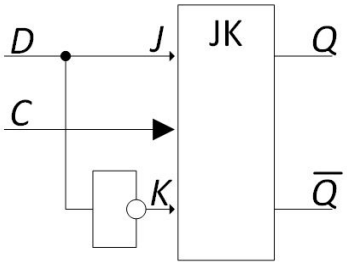
v) T-триггер: при подаче 0 выводит предыдущее значение, при подаче 1 инвертируется. Если поддерживать 1 на входе, то меняет значение каждый такт синхронизации.



Можно из таких собрать счётчик.



vi) D-триггер - хранит бит, который мы дали ему на вход (хороший мальчик). Если взять много таких, то можно сделать простую ячейку памяти, но об этом чуть позже.



3) Представление чисел в компьютере.

a) В компьютерах используется двоичная система счисления. Это удобно: 0 - нет тока/его мало, 1 - ток есть/гораздо выше, чем при 0. Оптимально хранить числа в системе счисления e (экспонента), но хер ты это реализуешь. В то же время, троичная система счисления обладает большей плотностью записи, но к тому времени как появились нормальные образцы на троичной системе счисления, уже всё устоялось на двоичной и менять на троичную было слишком дорогим и неоправданным риском (Эх, Сетунь, моя Сетунь).

b) Хранить положительные числа легко и приятно, но что делать с отрицательными?

   i) Прямой код aka первый бит под знак. Легко получить, имеем одинаковое кол-во положительных и отрицательных чисел. Минусы: у нас появилось два нуля (000 и 100), а также как теперь складывать?

   ii) Код со сдвигом: ко всем числам прибавляем сдвиг. Т.е. если у нас 2^n чисел всего, то будем сдвигать на 2^{n-1}. Т.е. -5 -> -5 + 128 -> 123, 10 -> 10 + 128 -> 138. Код легко получить, нет двух нулей. Минусы: у нас разное кол-во положительных и отрицательных чисел, а также при реализации арифметических операций нам нужно учитывать сдвиг, что сложна.

   iii) Дополнение до 1 (это точно так называется?): Отрицательное число - инвертированное положительное. Легко преобразовывать, представление натуральных не поменялось, а ещё равенство

положительных и отрицательных. Минусы: два нуля, а ещё сложно производить арифметические операции.

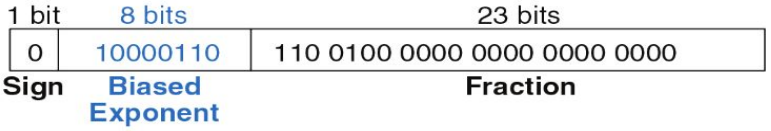
iv) Всякая херня, типа основание -2, основание 3 и прыжки вокруг нуля.

v) Дополнения до 2: вес старшего разряда -2^{n-1}. Отрицательное число выглядит как инвертированное положительное плюс один. Со сложением и вычитанием всё ок, нет двух нулей. Минусы: неудобно сравнивать, разное количество положительных и отрицательных. Самый часто используемый вариант.

c) Хранение нецелых - отдельная головная боль. Постоянно помним про неточность вычислений и хранения, сравнивать дробные числа на равенство - всё равно что подвесить над собой гильотину и пилить верёвку и прочие прелести жизни.
В дробных числах используется бит под знак.

i) Числа с фиксированной точкой. Под целую и дробную часть у нас выделено константное кол-во бит. Просто реализовать и понять, удобные арифметические операции. Минусы: небольшой диапазон, также говорят, что непонятно, где заканчивается целая и начинается дробная часть.

ii) Числа с плавающей точкой. Один бит под знак, экспонента - целое со сдвигом, обычно на 128, остальное - мантисса.



Число получаем, как $(-1)^{sign} * x, fraction * 2^{exp}$, где x - 0 или 1 (см. дальше). x обычно в записи самого числа не появляется, а держится “в уме”.
Есть два нуля и специальные числа: +inf, -inf (все биты экспоненты 1, все биты мантиссы 0); NaN (все биты экспоненты 1, мантисса ненулевая).
Если нам нужно округлять числа, то округляем в сторону ближайшего чётного.
Обычно, у нас есть возможность одно число представить бесконечностью способов, а теперь про x:

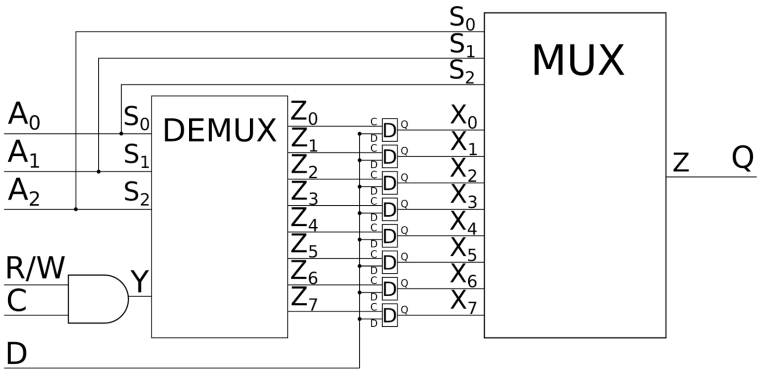
(1) Нормализованные числа: x = 1, мантисса принимает значение в диапазоне [1, 2). Не возникает проблемы с тем, что одно число можно представить разными способами. Разичают: half precision (1 + 5 + 10), single precision aka float (1 + 8 + 23), double precision aka double (1 + 11 + 52) и quad precision (1 + 15 + 112).

(2) Денормализованные числа: x = 0. Используют, чтобы повысить точность вычислений вблизи нуля. Если все биты экспоненты 0, то мы имеем дело с такими числами. Они работают медленно, но выигрываем в точности.

d) Умножение (деление): складываем (вычитаем) экспоненты, умножаем (делим) мантиссы. Сложение и вычитание: приводим к меньшему из порядков, складываем/вычитаем мантиссы, нормализуем, при необходимости округляем. Кек в том, что если складывать очень большое число и очень маленькое, то из-за ограничений в точности мы получим неизменённое большое число.

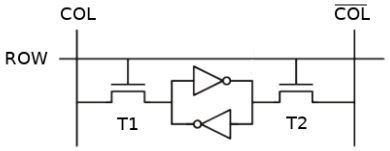
4) Устройство оперативной памяти

a) *Если взять много таких, то можно сделать простую ячейку памяти*

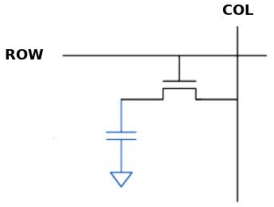


На восьми битах у нас всё хорошо, но в промышленных масштабах это занимает over дофига места, а ещё куча проводов, а ещё D-триггеры-Хрен-Синхронизируешь (а почему они Хрен-Синхронизируешь? Да потому что их Хрен, мать его, Синхронизируешь!), а ещё это энергии жрёт, словно корейская майнинговая ферма.

b) Немного отвлечёмся от игрушек и посмотрим на нормальные ячейки памяти. У нас есть статическая память



И динамическая

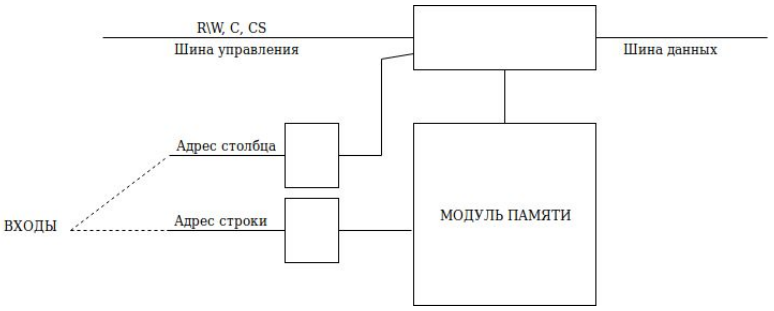


Статическая память как BMW: дорогая (целых 6 транзисторов) и быстрая, а ещё громоздкая. Ей необходимо постоянное

энергообеспечение, но взамен она даёт нам стабильное сохранение полученного значения. Для записи подаём то, что хотим записать на COLumn.

Динамическая память дешёвая (транзистор + конденсатор) и маленькая. За это мы платим тем, что заряд с конденсатора имеет свойство утекать, как следствие ячейку нужно регулярно перезаряжать (примерно раз в 64 мс). Чтобы почитать, подаём на COL 0,5 и если напряжение увеличивается, то в ячейке 1, иначе 0. После чтения нужно перезарядить. Чтобы записать, подаём на COL 0/1 -> заряд из ячейки утекает/втекает.

c) Как в общем выглядит оперативная память:



Писать и читать мы можем через один провод, потому что мы никогда не пишем и не читаем одновременно.

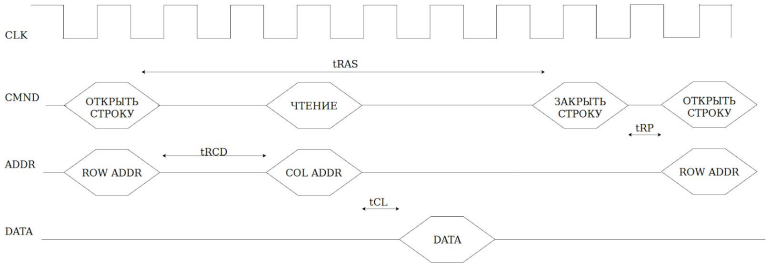
d) Как можно организовать модуль памяти?

Сделать по отдельной линии на каждую ячейку - жесть полная, мы в этих линиях утонем.

Расположить все ячейки в один ряд - нам понадобится ооогромный демультиплексор, а ещё мы хотим поэкономить место, а ещё возникают проблемы с синхронизацией и потерей сигнала.

Поэтому оптимально запихать всё в матричку.

e) Чтение и (наверное)всё, что с этим связано.



tRAS - время между открытием строки и её закрытием, то есть время, требуемое, чтобы полностью пройти цикл чтения из ячейки.

tRCD - задержка между открытием строки и открытием столбца.

tCL - задержка между открытием ячейки и началом получения из неё данных.

tRP - время на перезарядку ряда, т.е. то время, которое пройдёт между тем, как мы закончим читать, и тем, когда сможем снова открывать ячейку.

Здесь имеют значение два параметра: latency aka скорость доступа (время между началом подачи адреса и началом получения данных) и

throughput aka пропускная способность aka скорость передачи данных (время между началом получения одной порции данных и началом получения следующей порции).