Занимательная история айти

Введение

На сайте расположены учебники по администрированию реляционных баз данных. Если начинать изучение с нуля, тема баз данных покажется сложной и неинтересной. Перед изучением администрирования, обычно, предлагалось начать изучение с языка SQL. Книга по SQL выложена на сайте, но даже эта книга может показаться занудной, при самостоятельном изучении.

В этой книге собраны занимательные истории развития индустрии айти, чтобы стали понятными её истоки.

Часть 1. История чисел

Унарная система счисления

Система счисления - это представление чисел с помощью знаков. Самая древняя система счисления - унарная (unary, единичная). В унарной системе есть единственная цифра, которую обозначают единицей, чёрточкой. Например, число три в унарной системе можно записать как три чёрточки: ||| . Унарная система счисления является "непозиционной", то есть позиция чёрточки (первая или вторая по порядку) не влияет на число (сумму чёрточек), которое обозначается чёрточками. Унарная система применяется:

1) при обучении счёту в начальных классах школы с использованием счётных палочек;

2) при использовании зарубок для ведения календаря там, где нет бумаги. Робинзон Крузо в романе Даниэля Дефо делал зарубки на деревянном столбе для подсчёта дней;

3) для отображения уровня на индикаторах:

4) для обозначения цифр внутри шестидесятеричных разрядов в вавилонской системе счисления.

Числа использовались для подсчёта предметов и вычислений. Например, сколько порций еды нужно запасти, чтобы кормить трёх домашних животных в течение недели, если одно животное ест одну порцию в день. Понадобится 7*3=21 порция. Вычисления проводили руками или подручными средствами - раскладывали палочки или камушки.

Двенадцатеричная система счисления

Двенадцатеричная система счисления возникла из счёта, при котором большим пальцем руки считают каждую фалангу четырёх пальцев той же самой руки. Двенадцатеричный пальцевой счёт сейчас распространён на территории Индии, Пакистана, Афганистана, Ирана, Турции, Ирака.

---

*Первые три степени числа 12 имеют собственные названия: дюжина = 12; гросс = 12 дюжин; масса = 12 гроссов. Двенадцать тарелок это "дюжина тарелок". Мелкие вещи, такие как скрепки, продавали гроссами, по 12*12=144 штук. Фраза "масса народу" обозначала не вес, а число людей: как минимум, 12*12*12=1728 человек.*

Удобство двенадцатеричной системы в том, что у числа 12 много делителей: 2, 3, 4, 6. Это значит, что двенадцатеричные числа легче делить без остатка и можно обходиться без

дробей. Двенадцатеричная система счисления совместима с шестидесятеричной, так как 12 является делителем числа 60.

В древнем Риме унция была 1/12 частью меры веса libra (в переводе "весы́"). Современный фунт обозначается как "lb", что является сокращением от libra. Английская система мер основана на римской. В XX веке английская система мер стала вытесняться метрической, которая основана на десятеричной системе счисления.

Шестидесятеричная система счисления

Шестидесятеричная система счисления использовалась в Вавилоне за две тысячи лет до нашей эры. Эта система счисления появилась как комбинация двенадцатеричной и пятеричной систем, так как шумерские названия чисел 6, 7, 9 имеют следы пятеричного счёта. В римской записи также используются буквы для обозначения пятёрок: I=1, V=5, X=10, L=50, C=100, D=500, M=1000. Для дробей в римской системе использовались значки S = 1/2, Ↄ = 1/12 (унция). Для записи чисел использовались значки:

"|" для обозначения единиц;

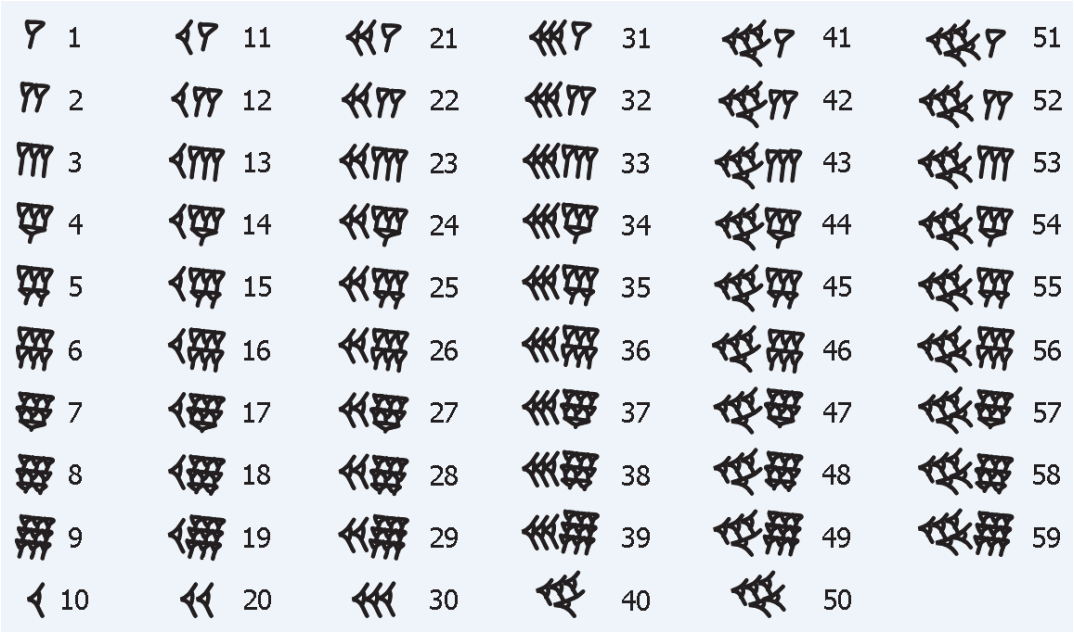
"<" для обозначения десятков внутри шестидесятеричного разряда;

"|–" для обозначения числа 100;

"<|–" для обозначения числа 1000.

Пример записи числа 23: "<<|||". Почему значки имеют такую своеобразную форму начертания? Значки наносились треугольной палочкой (клином) на сырую глину и являлись отпечатком клина. Поэтому вавилонская письменность называется клинописью (cuneiform). Пример изображения древней сумочки и клинописного текста:

В вавилонской системе для нуля не было значка, что приводило к неоднозначной записи чисел. О значении чисел приходилось догадываться по контексту или по отступам между значками. Позднее, между VI и III веком до нашей эры, обозначение нуля появилось в виде символов "\\", но использовалось только в средине шестидесятеричных чисел для обозначения пустых разрядов. В конце числа значки нуля использовались только в астрономических вычислениях. Пример записи чисел от 1 до 59 в вавилонской шестидесятеричной системе:



Шестидесятеричная система использовалась древнегреческими астрономами для измерения угловых координат звёзд. Элементы вавилонской шестидесятеричной системы счисления дошли до нашего времени и используются при измерении времени и углов: в часе 60 минут, в круге 360 градусов.

Астрономам всегда были нужны точные вычисления. После развития мореплавания, точные вычисления понадобились для навигации в море. После изобретения бездымного пороха в 19 веке, когда снаряды смогли летать на большие расстояния, точные вычисления понадобились для расчёта траекторий снарядов (баллистики). Там, где есть большие расстояния, в вычислениях накапливаются погрешности и результат искажается, поэтому ищутся способы повысить точность вычислений. Потребность в точных вычислениях способствовала развитию математики и привела к созданию вычислительных машин.

Десятичная система счисления

В позиционных системах счисления значение каждого значка зависит от его позиции ("разряда"). Вавилонская система - позиционная, но каждое число записывается в "аддитивной" (непозиционной) форме. Одной из проблем записи было то, что для понимания того, какое число изображено приходилось использовать отступы (пробелы). Пример записи двух чисел в вавилонской системе: "| |" = 61, "| | |" = 121.

Первое свидетельство о применении десятичной системы счисления обнаружено в Индии и относится к 595 году. Основным преимуществом системы было то, что в ней имелся знак для обозначения нуля, а самое главное, что этот знак использовался в конце числа (финальной позиции).

Число 10, как основание системы счисления, было выбрано по числу пальцев на двух руках, которые можно было использовать при счёте, загибая пальцы и показывая обеими руками число. Один палец обозначал единицу в унарной системе.

Из Индии десятичная система пришла в арабские страны. аль-Хорезми описал индийский счёт в своей книге, которая попала в Европу. Простые и удобные правила сложения и вычитания чисел десятичной позиционной системе, сделали её популярной. Книга аль-Хорезми написана на арабском языке, поэтому десятичная система счисления стала называться в Европе арабской, а индийские цифры стали называть арабскими. Арабские цифры постепенно вытеснили римскую запись цифр и другие непозиционные системы. Позиционная запись арабскими цифрами в десятичной системе имеет преимущества по сравнению с римской потому, что:

- она компактнее римской;
- она позволяет визуально сравнивать числа по величине;
- в ней есть простые способы умножения и деления.

В римской записи сложно записывать большие числа, а дроби вообще не записываются. Переход на арабские цифры и десятичную систему ускорил развитие математики.

Буквы и цифры

До появления арабских чисел, цифры обозначались буквами алфавита. Для цифр не существовало собственных символов.

Авраам (основоположник трёх религий), наблюдая за звёздами, сделал вывод о существовании единой силы.

В Коране 6:76 написано: когда стало темно, он увидел звезду и сказал: Это - мой Господь. А когда она закатилась, он сказал: Я не поклоняюсь тому, что закатывается.

В Библии, главе Бытие 26:5, написано: "Итак Авраам слышал Мой голос".

В Вавилонском Талмуде написано: "Ами сын Аба сказал: Аврааму было 3 года, когда написанное (в главе Бытие 26:5) произошло" и приводятся вычисления методом "гема́трии".

В гематрии у каждой буквы имеется своё числовое значение. "Гематрией слова" называется сумма числовых значений входящих в слово букв. Если у слов одинаковая гематрия, то считается, что между словами есть смысловая связь. Гематрия слова "итак" (עֵקֶב, состоит из трёх букв, произносится "экев") равна 70+100+2=172. Делается вывод, что Авраам слышал голос 172 года. Авраам жил 175 лет. Делается вывод, что Авраам услышал голос в 175-172=3 года.

Аналоги гематрии есть в других алфавитах: акшара-санкхья в индийском (деванагари), абджадия в арабском.

В пустыне (или в горах) в безоблачную погоду на ночном небе отчётливо видны звёзды от края и до края неба. Вид звёздного неба воодушевил маленького Авраама и он задался вопросом: "Кто создал это" (произносится как "Ми бара Эле"), вопрос стал утверждением, что кто-то создал.

До Авраама люди видели то же самое небо, но вряд ли обращали внимание на его красоту, возможно, потому, что не видели в нём пользы для своей нелёгкой жизни в древнем мире, и его вид не пробуждал в них мыслительную активность.

Вид звёздного неба (Starlight Headliner) доступен пассажирам автомобилей Роллс-Ройс с 2003 года, начиная с модели Фантом VII:

Ноль

В римских цифрах отсутствует значок для отображения нуля. В Европе, до появления арабских цифр, использовались римские цифры и отсутствие нуля затрудняло развитие математики. Вместо нуля использовали слова nulla или nullae (в значении "нет").

Использование нуля и операций с нулём впервые изучил и описал индийский астроном и математик Брахмагупта в 628 году. До нас дошло его сочинение "Брахма-спхута-сиддханта", которое переводится как "Доктрина (дайджест, канон) манифестации (проявления, учения) Брахмы". В сочинении описывается "математика процедур" (алгоритмы), "математика семян" (уравнения), арифметические операции, ряды, пропорции (дроби).

В "Книге рекордов Гиннеса" самая большая единица измерения времени - индийская "кальпа" ("день Брахмы", равный 4,32 миллиарда лет). Единица измерения описана в "пуранах" (древних сказках), которые были записаны в период с III по XII век.

Брахмагупта определил ноль как результат вычитания из числа самого числа. В современной алгебре ноль определяется как "нейтральный элемент" относительно операции сложения: `a+0=0+a=a`. Кроме сложения, ноль также нейтрален относительно вычитания. Единица является "нейтральным элементом" относительно операции умножения, возведения в степень и деления.

Брахмагупта описал правила арифметических операций над положительными, отрицательными числами и нулём, рассматривая при этом положительные числа как имущество, а отрицательные числа как "долг". Брахмагупта даже дал определение деления на ноль:

Деление нуля на ноль равно нулю;

Деление положительного или отрицательного числа на ноль равно дроби с нулём в знаменателе;

Деление нуля на положительное или отрицательное число равно нулю.

Единица

В древности математики также отдельно выделяли число 1, считая его даже не числом, а отдельным понятием. Древнегреческий математик Евклид, живший примерно в 300 году до нашей эры, сначала дал определение единице, а затем определил "число", как множество единиц. По его определению, единица не является числом и уникальных чисел не существует. К примеру, любые две единицы из набора единиц являются числом 2.

При использовании двоичной системы счисления, неоднозначности исчезают, но в то время двоичную систему счисления ещё не изобрели, она появилась только в 1703 году.

Академик, доктор физико-математических наук, профессор Журавлёв, в своей книге "Основы теоретической механики" писал:

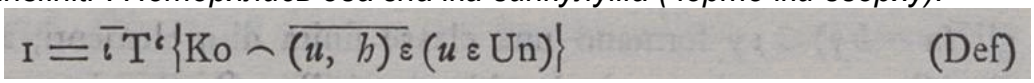
Точно так же аксиоматизация арифметики не является предметом самой арифметики. По этому поводу уместно процитировать А. Пуанкаре *, который, анализируя проблему аксиоматизации арифметики в главе "Математика и логика", пишет:

"...Бурали-Форти определяет число 1 следующим образом:

$$1 = \iota T'\{\mathrm{Ko} \frown (u, h)\varepsilon(u\varepsilon U n)\}.$$

Это определение в высшей степени подходит для того, чтобы дать представление о числе 1 тем лицам, которые никогда о нем ничего не слышали!".

Примечательно, что формула, приведённая в книге Poincaré "Science et méthode", отличается от той, что приведена в статье Бурали-Форти "Una questione sui numeri transfiniti". Потерялись два значка винкулума (чёрточки вверху):



$$1 = \overline{\iota}\,\mathrm{T}'\{\mathrm{Ko} \frown \overline{(u,\ h)}\,\varepsilon\,(u\,\varepsilon\,\mathrm{Un})\} \qquad \text{(Def)}$$

Можно предположить, что винкулум обозначает группировку элементов и значок можно было бы опустить, но черточка продолжается над эпсилоном и просто так её опускать нельзя. Первую чёрточку Бурали-Форти тоже поставил не для красоты. Сложные обозначения и запутанные определения, которые используют математики, помогают им найти что-то новое, взглянуть на математические понятия с другой стороны. Если вычурное определение единицы помогло Бурали-Форти открыть "парадокс Бурали-Форти", то такие обозначения и цепочка логических умозаключений были для него полезны. Для других математиков его обозначения бесполезны потому, что всё, что можно было открыть с их помощью, Бурали-Форти, вероятно, уже открыл. Поэтому, Пуанкаре и Журавлев не вникали в обозначения, которые использовал Бурали-Форти. При обучении математике стараются использовать наиболее простые обозначения и определения, но это не всегда удаётся.

В 1960 году в ВМС США создали принцип проектирования KISS (Keep It Simple, Stupid): чем проще система, тем она лучше работает. Принцип запрещает использование более сложных средств, чем необходимо для решения задачи. Этот принцип применяется и при проектировании и при разработке программ.

Французский математик Андре Вейль рассказал математику Владимиру Арнольду, что он и его однокурсники, закончив учиться, пришли к выводу, что всё преподавание математики непонятно с самых основ и нужно всё переделать. Этой переделкой они и занялись, создав для этого группу под названием "Никола Бурбаки".

Для обозначения пустого множества Бурбаки ввели символ Ø (перечёркнутый ноль), который стал использоваться для обозначения нуля в компьютерных шрифтах для того, чтобы не возникало путаницы с буквой "О", похожей на ноль.

Обозначения и правила их использования влияют на лёгкость изучения. Это относится к любой области, и к математике и к информатике.

Натуральные числа

Целые положительные числа исторически называются "натуральными", так как они возникают естественным (натуральным) образом при подсчёте предметов. Натуральные числа используются для описания числа предметов и порядкового номера предмета.

Существует два подхода к определению натуральных чисел:

1) определить их как числа, возникающие при нумерации (подсчёте) предметов: первый, второй, третий, четвёртый, пятый. В этом подходе ряд натуральных чисел начинают с единицы. Ординальные (порядковые) числа придумал математик Кантор. Его подход назвали наивной теорией множеств, так как математик Бурали-Форти нашел в ней парадокс, который так и назвали "парадокс Бурали-Форти".

2) определить их как числа, возникающие при обозначении количества предметов: 0 предметов, 1 предмет, 2 предмета, 3 предмета, 4 предмета, 5 предметов. При таком определении, ряд натуральных чисел начинается с нуля. Этот подход использует Бурбаки. Наличие нуля облегчает формулировку и доказательство теорем арифметики.

Математики до 20 века не смогли определиться, стоит ли считать ноль натуральным числом. В этом им помогла Международная Организация по Стандартизации (International Organization for Standardization, ISO). В 1992 году ISO выпустила стандарт ISO 31-11 "Математические обозначения и символы для использования в естественных науках и технологии", где отдельно оговорили, что символ $\mathbb{N}$ обозначает "множество натуральных чисел, включая ноль". Последние два слова положили конец многовековой неоднозначности, и ноль занял своё место в ряду натуральных чисел.

Число элементов называют мощностью или кардинальным числом множества предметов. Мощность обозначается буквой алеф א. Мощность множества натуральных чисел $\mathbb{N}$ обозначают как "алеф-нуль" $\aleph_0$.

---

11 августа 1982 года Эдсгер Дейкстра, первый нидерландский программист, опубликовал статью EWD831 (он нумеровал статьи своими инициалами и порядковым номером) "Почему нумерация должна начинаться с нуля".

*Нумерация элементов в массивах, коллекциях и символов в строках в большинстве языков программирования начинается с нуля (языки C, C++, Java, Python), но есть языки, где используются **другие** правила. В языках Lua, MATLAB нумерация массивов начинается с единицы, а в языке программирования Algol при определении массива в форме `a[0:3]` нумерация начинается с нуля, а при определении в форме `a[3]` нумерация начинается с 1.*

*Дейкстра приводит аргумент с точки зрения практики: "Обширный опыт работы с языком программирования Mesa показал, что использование трёх **других** правил, было постоянным источником неловкости и ошибок, и на основании этого опыта программистам Mesa теперь настоятельно рекомендуется не использовать их".*

Написать статью Дейкстру "побудил инцидент, когда в эмоциональном порыве, один из преподавателей-математиков в университете (не являющийся специалистом по информатике) обвинил нескольких молодых преподавателей информатики в "педантизме", потому что (по привычке) они начали нумерацию с нуля. Математик воспринял сознательное принятие наиболее разумной условности как провокацию."

Разница в нумерации у массивов, вероятно, в том, что разработчики одних языков представляли себе указатель элемента массива как смещение от начала массива, а других языков представляли себе указатель как порядковый номер элемента массива.

Древние игры

В древности, для игр использовались доски с фигурами (фишками), так как не было компьютерных игр. В 1980х годах компьютерные игры способствовали широкому

распространению домашних компьютеров. Примером древней игры является древнеегипетская настольная игра с передвижением фишек по доске, называющаяся "сенет" (ⲥⲓⲛⲉ, "прохождение"), известная с 3500 года до нашей эры.

Произношение названия игры неизвестно, как и любых слов египетского языка. Самый древний полный комплект для сенет найден в гробнице врача Хеси-Ра, библиотекаря фараона Гёсера в Саккаре (III династия, 2686 год до нашей эры). Как и во многих играх, выигрыш в сенет зависел от соединения умения (стратегии игры) и случайности (удачи). Элемент случайности ведён в игру "генератором случайных чисел" - броском четырёх палочек. Использовались палочки плоской формы, одна сторона чёрная, другая белая.

После броска палочек считали: одна палочка упала белой стороной вверх - 1 очко и дополнительный бросок; две - 2 очка; три - 3 очка, четыре - 4 и дополнительный бросок. Если все палочки упали чёрной стороной вверх - 5 очков и дополнительный бросок (это максимальный результат). Очки определяли насколько ходов можно переместить фишки.

Дальше в игру вступало умение игрока: он мог использовать очки для передвижения пяти фишек, одной или нескольких. Результат игры зависел от умений игроков и случайности (удачи).

Сенет фараона Аменхотепа III

В древнем Риме играли в "табулу" (произошло от греческого слова τάβλη, доска), которая является предшественником современной игры "нарды".

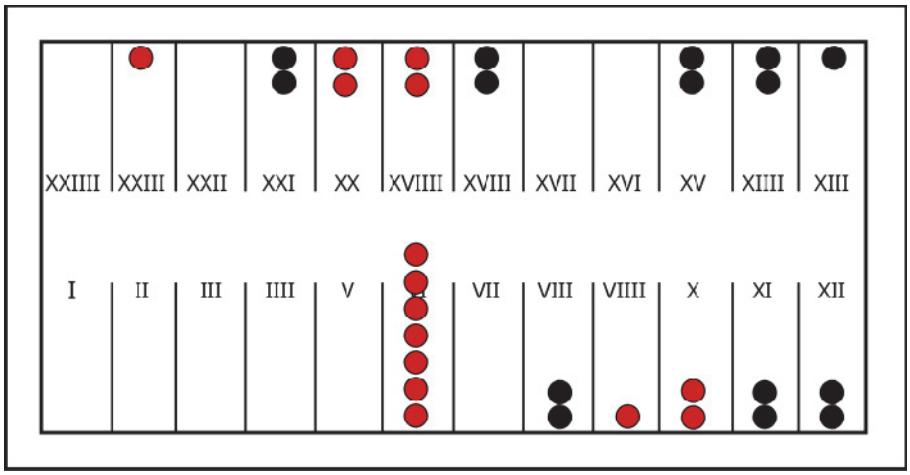


схема доски для игры в табулу

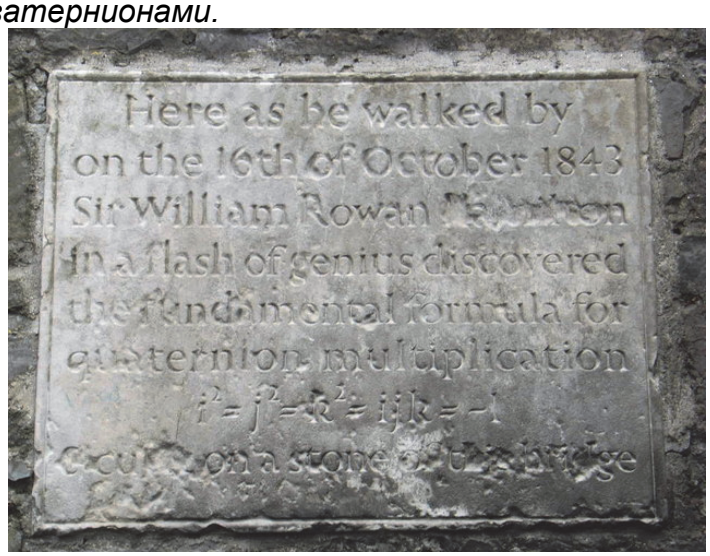
В табуле бросали не палочки, а кубики.

В трёхмерном мире есть 5 "правильных" многогранников ("Платоновы тела") - таких, которые имеют пространственную симметрию и одинаковые грани. Один из них - это куб с 6 квадратными гранями. У трёх других - 4, 8, 20 треугольных граней, у одного - 12 пятиугольных граней.

В 4-мерном пространстве есть шесть правильных многогранников.

В пространствах с бóльшим числом измерений по три правильных многогранника.

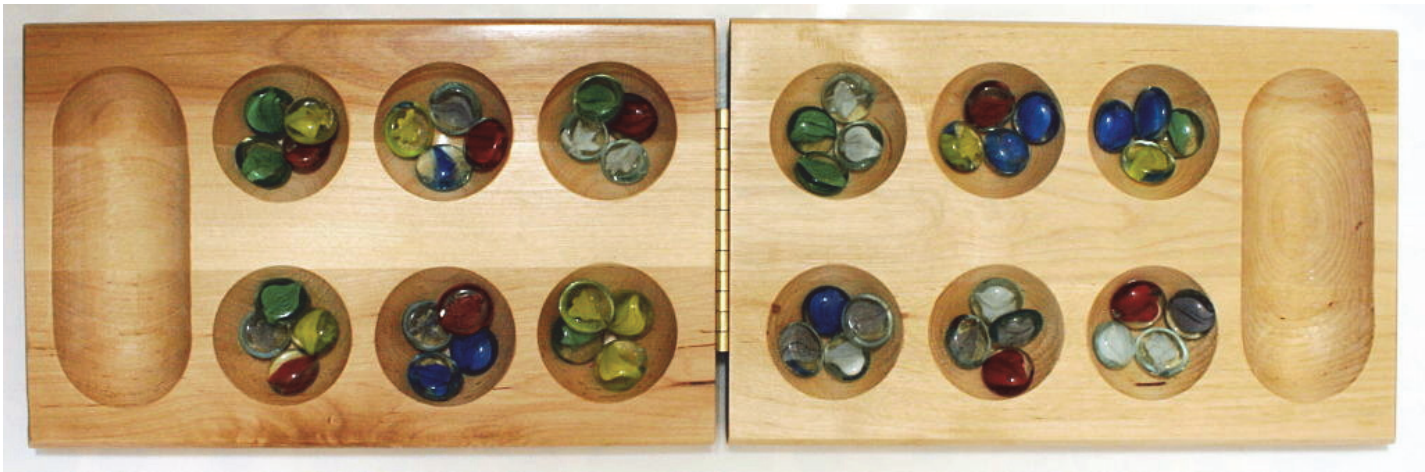
С пространствами связана теорема Фробениуса и теорема Гурвица. Векторное произведение однозначно определено только в трёхмерном пространстве и описывается кватернионами.



памятная табличка: Прогуливаясь здесь, 16 октября 1843 года, в гениальном озарении, сэр Гамильтон открыл формулу перемножения кватернионов.

Для дуализма есть алгебра октонионов, но их произведение лишено ассоциативности.

В Африке и Азии играли в игры семейства "манкала" (منقلــة , manqalah). Это игра для двух игроков, которые перемещают зёрна, семена, косточки, камушки по лункам. Игра qalah (калах) - современная игра из этого семейства. Доска для игры:



Игра калах была реализована на компьютере БЭСМ-6 (Большая Электронная Счётная Машина). В БЭСМ-6 была диалоговая программа Джин. Пользователь компьютера играл с программой. У игрока и программы было:

6 лунок - игровых полей,

Одна большая лунка, которая называется qalah, в неё надо переместить камни.

Первоначально камни распределены поровну по всем игровым лункам.

Постоянная тонкой структуры - безразмерное число, примерно равное 1/137,035999. Определяет силу взаимодействия между электронами и ядрами атомов. Если бы постоянная была больше на 4%, то звёзды не могли бы производить углерод и более тяжёлые элементы.

Алгебра

Книга Брахмагупты попала в Багдад, где её перевели на арабский язык. В 9 веке учёный (астроном, географ, историк, переводчик, философ, математик) аль-Хорезми написал книгу "Китаб **аль-джебр** валь-мукабала" (Книга о сложении и вычитании). От названия книги произошло слово **алгебра**. аль-Хорезми описал индийскую позиционную десятичную систему счисления, сформулировал правила вычислений, в том числе для нуля. Индийское название нуля он перевёл как as-sifr или просто sifr, откуда пошли слова цифра и шифр.

аль-Хорезми был первым человеком, который рассматривал алгебру как самостоятельную дисциплину, а также первым начал преподавать алгебру в простой форме (буквально "с нуля"), поэтому он признан основателем алгебры.

аль-Хорезми родился в городе Хива Хорезмской области в 783 году. Сейчас город Хива находится в Узбекистане. аль-Хорезми переехал в Багдад, где возглавил Байт аль-Хикма (Дом мудрости, аналог современных Академий наук), созданную халифом аль-Мамуном, сыном Гаруна аль-Рашида, который упоминался в книге "Тысяча и одна ночь" (Китаб альф лейла ва лейла).

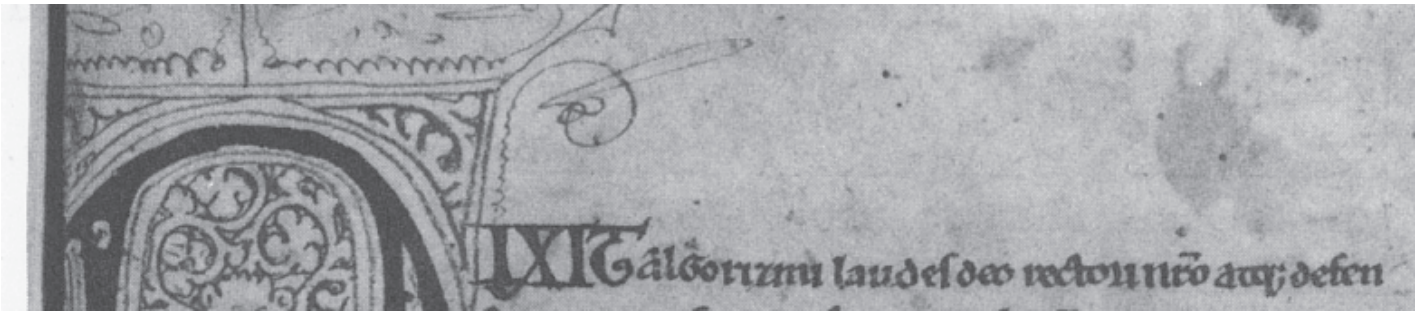
аль-Мамун примечателен тем, что в 831 году первым, со времен строительства пирамиды фараона Хуфу (произношение египетских слов неизвестно, греки писали имя фараона как Суфис->Саофис->Хеопс), вошел в большую галерею пирамиды Хеопса. аль-Мамун покровительствовал учёным, по его заказу был сделан перевод "Альмагеста" Птолемея на арабский язык.

Альмаге́ст (от арабского аль-Маджисти) "Великое математическое построение по астрономии в 13 книгах" - произведение Клавдия Птолемея, созданное около 140 года и включающее все известные астрономические знания Греции и Ближнего Востока того времени. На протяжении 13 веков Альмагест оставался основой астрономии.

В Европу Альмагест попал в период Возрождения в арабском переводе.

Спустя столетие после аль-Хорезми в Байт аль-Хикма работали аль-Бируни и его коллега ибн-Сина (Авиценна). В 1000 году аль-Бируни в своём сочинении "Хронология, или памятники минувших поколений" описал все известные ему календари и составил хронологическую таблицу всех эпох, начиная от библейских патриархов.

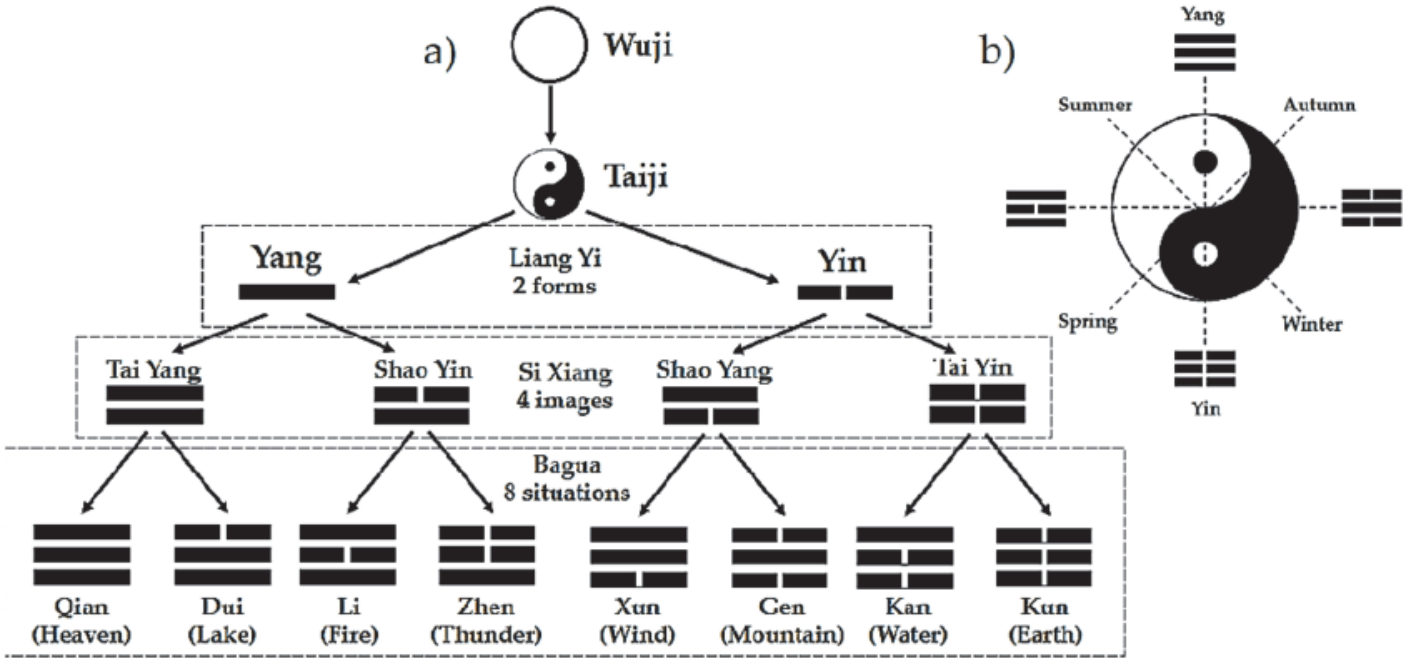
В первой половине 12 века книга аль-Хорезми попала в Европу в латинском переводе. Вместе с книгой аль-Хорезми в Европу попали индийские цифры, которые стали называть арабскими. Книга начиналась со слов: "Dixit algorizmi" (Аль-Хорезми говорил), откуда и пошло слово алгоритм.



Книга перемен

Книга перемен ("И цзин") - китайский философский текст, датируемый примерно 700 годом до нашей эры. Текст книги примечателен тем, что в нем используются символы двоичного счисления: горизонтальная черточка и прерывистая черточка. В тексте книги приведено 64 значка, обозначаемые 6 разрядами. В современном двоичном счислении значки могли бы обозначаться двоичными числами: 000000 = ䷀, 000001 = ䷪, 000010 = ䷆, 000011 = ䷩. Символы называют "гексаграммами" от слова гекса (шесть). Гексаграммы состоят из двух триграмм. Каждая триграмма и гексаграмма имеет своё название, значение и толкование.

Также гексаграмму (6 делится на 3 и 2) представляют как 3 пары из двух черточек. В древнекитайской философии есть три элемента: небо, человек, земля. Первая пара черточек, начиная снизу, символизирует землю, вторая пара - человека, третья небо.



Из бесконечности (Wuji - ничто, обозначается кружком - значок нуля) создаётся разделение на две противоположности (Taiji). Одна сторона (Ян) обозначается линией, вторая сторона (Инь) прерывистой линией. Дальше они комбинируются, получается четыре комбинации, каждая из двух черточек. Потом ещё раз комбинируются, получается восемь комбинаций по три черточки (триграммы).

Знаки триграмм и гексаграмм присутствуют в современной универсальной схеме кодирования символов Unicode, это символы с 2630 по 2637 и с 19904 по 19967.

В двоичной системе счисления шесть разрядов позволяет хранить 2^6 (два в степени шесть) = 64 значения. Один байт состоит из 8 битов и хранит 2^8=256 значений.

Двоичная система счисления

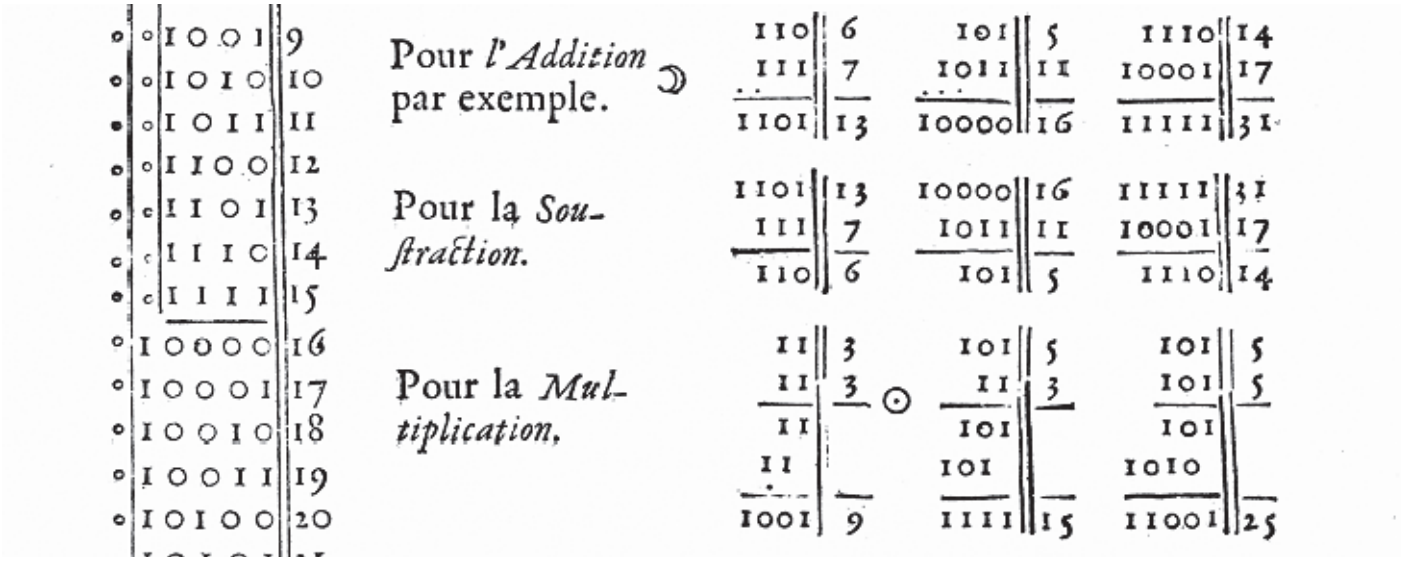
Двоичная система счисления была впервые описана в 1703 году немецким учёным Лейбницем в статье "Объяснение двоичной арифметики". Лейбниц изобрёл термин "функция", стал основоположником математического анализа и первым президентом Берлинской

академии наук. Лейбниц первый начал использовать точку "·" для обозначения умножения, поскольку символ крестик "×" можно перепутать с переменной, обозначаемой буквой "x" (икс). В языках программирования для операции умножения используется символ "*". С символом деления получилось наоборот. Лейбниц стал использовать символ ":" для деления, чтобы не спутать операцию деления с обозначением дробей, которые в то время были популярны в математических статьях. Двоеточие оказалось неудачной заменой, так как двоеточие используется в обычных текстах. Чтобы не было неоднозначности, стали использовать символ "÷", но этот символ не получил распространения и отсутствует на клавиатуре компьютеров.

В языках программирования дроби, как тип данных не используются, поэтому в языках программирования оператор деления обозначается символом "/".

На создание двоичной арифметики Лейбница натолкнуло знакомство с китайской Книгой перемен, с которой он ознакомился, переписываясь с миссионером и картографом Бувэ. Лейбниц заметил, что гексаграммы Книги Перемен соответствуют двоичным числам от 000000 до 111111.

Двоичная система удобна для выполнения арифметических операций. Например, сложение имеет три правила: 0+0 = 0; 0+1 = 1; 1+1 = 0 и перенос 1 в старший разряд.



таблицы сложения, вычитания, умножения из статьи Лейбница "Объяснение двоичной арифметики"

Готфрид Лейбниц родился 1 июля 1646 года. В 6 лет он потерял отца, профессора этики Лейпцигского университета. Отец оставил после себя большую библиотеку.

Учитель Лейбница заметил, что его ученик читает книги и пошел жаловаться родственникам Готфрида, чтобы они обратили внимание на "неуместное и преждевременное" чтение книг, которые, по мнению учителя, были не для его возраста. Он бы убедил в этом родственников Лейбница, если бы случайным свидетелем этого разговора не оказался, живший по соседству, учёный и много путешествовавший дворянин.

Поражённый недоброжелательностью учителя, который мерил всех одной мерой, он сказал, что было бы глупо ограничить интерес мальчика к науке и надо приветствовать стремление к знаниям. На всякий случай, пригласили Готфрида, он правильно ответил на вопросы дворянина и тот заставил родственников Лейбница дать слово, что Готфриду дадут доступ в библиотеку отца и разрешат читать всё, что захочет.

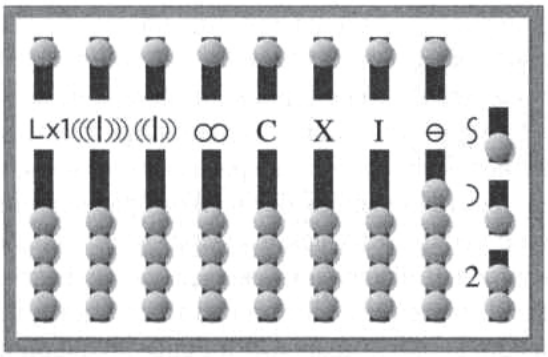
Как писал Лейбниц: "Я торжествовал, как если бы нашёл клад, потому что сгорал от нетерпения увидеть древних, которых знал только по имени - Цицерона и Квинтилиана, Сенеку и Плиния, Геродота, Ксенофонта и Платона, писателей эпохи императора Августа и многих латинских и греческих отцов церкви. Всё это я стал читать, смотря по влечению, и наслаждался необычайным разнообразием предметов. Таким образом, не имея ещё двенадцати лет, я свободно понимал латынь и начал понимать по-гречески."

Библиотека отца позволила Лейбницу изучить книги, к которым он мог бы получить доступ только в студенческие годы. В возрасте 12 лет Лейбниц был знатоком латыни, в возрасте 13 лет стал писать стихи. В 14 лет он окончил гимназию и в возрасте 15 лет поступил в Лейпцигский университет, где преподавал его отец.

Сложно сказать, стал бы Лейбниц учёным, если бы не трудности в получении доступа к книгам, которые подогревали стремление получить желаемое. В наше время доступ к большому объему информации прост, благодаря наличию интернет и поисковых сайтов. Читая книги, можно встретить непонятный термин и можно легко найти в интернет его описание. С другой стороны, при обилии информации, важность книг нивелируется и уменьшается желание разбираться в том, что в них пишут.

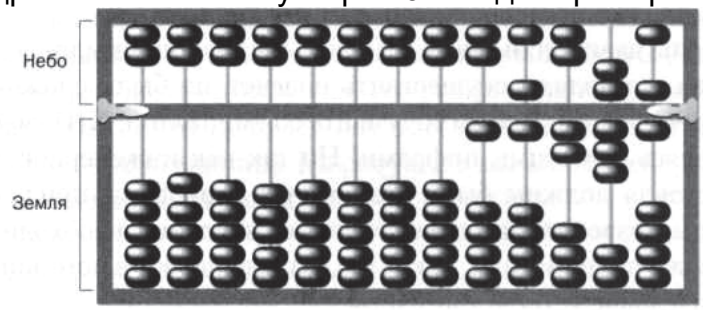
Счёты

Для подсчёта чисел в римской записи использовались римские счёты, которые назвались abacus ("абак"). Значение слова можно трактовать, как "доска с песчинками" (семенами, зёрнами, косточками, камушками). Прототипом римских счётов можно считать доску с линиями (углублениями), в которые клали камушки (косточки), которая использовалась в Вавилоне более 3000 лет до нашей эры.



римские счёты (abacus)

В Китае, тоже были счёты, они назывались "суаньпань". Первое упоминание о китайских счётах датируется 190 годом. Самое раннее изображение китайских счётов приведено в древнекитайском букваре 1371 года. Пример китайских счётов:



Из книги нобелевского лауреата по физике Ричарда Фейнмана "Вы, конечно, шутите, мистер Фейнман!":

И тут до меня доходит: он не знает чисел. Когда у тебя есть счеты, не нужно запоминать множество арифметических комбинаций; нужно просто научиться щёлкать костяшками вверх-вниз. Нет необходимости запоминать, что 9+7=16; ты просто знаешь, что когда прибавляешь 9, то нужно передвинуть десятичную костяшку вверх, а единичную - вниз. Поэтому основные арифметические действия мы выполняем медленнее, зато мы знаем числа.

Более того, сама идея о приближенном методе вычисления была за пределами его понимания, несмотря на то, что обычно невозможно точно вычислить кубический корень.

Счётные машины (арифмометры)

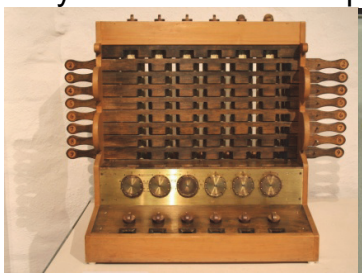
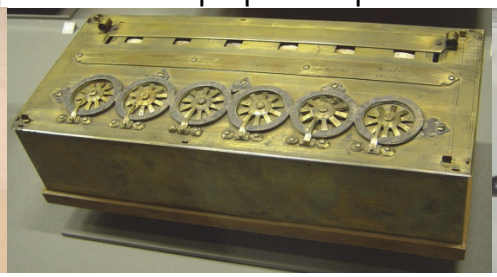
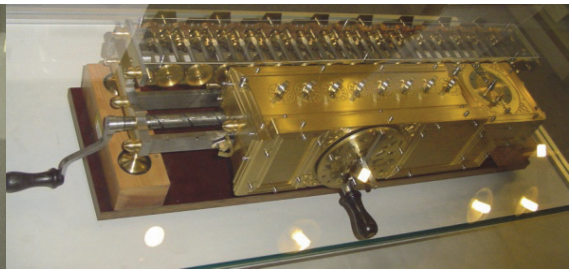
Арифмометры - это механические устройства, выполняющие арифметические операции над числами. Чтобы устройство считалось арифмометром, достаточно, чтобы оно умело складывать числа.

Первый арифмометр создал Вильгельм Шиккард в 1623 году. Его арифмометр выполнял четыре арифметических действия над шестизначными десятичными числами. Шиккард сделал два экземпляра арифмометра. Один экземпляр предназначался астроному и математику Кеплеру, который примечателен открытием, что планеты движутся по эллиптическим кривым, а не по кругу. Оба арифмометра сгорели при пожаре, но чертежи арифмометра сохранились и доказали приоритет Шиккарда, как изобретателя арифмометра.

Вторым арифмометр создал Блез Паскаль в 1962 году, когда ему было 19 лет. Его отец был сборщиком налогов и часто выполнял долгие расчеты. Блез захотел автоматизировать этот утомительный труд, чтобы отец больше времени уделял семье. В то время финансовые расчёты велись в денежных единицах, которые назывались ливры, су, денье. Сложность была не в названиях, а в том, что в ливре было 20 су, а в су было 12 денье. Арифмометр же использовал десятичную систему и был не очень удобен для подсчётов, поэтому не получил широкого распространения. Блез Паскаль собрал всего 50 арифмометров.

Третий арифмометр придумал Лейбниц в 1673 году. Он решил создать арифмометр после того, как математик, астроном и первый президент Французской академии наук Гюйгенс посетовал, что ему приходится выполнять много утомительных вычислений.

Шиккард, Паскаль, Лейбниц создали свои арифмометры независимо друг от друга, поэтому могут называться изобретателями арифмометра.

Более сложные счётные машины стали создавать гораздо позже. Технологии того времени не позволяли создавать сложные механизмы за разумную стоимость. Первый коммерчески успешный арифмометр был изобретен в 1820 году, стал продаваться в 1840х годах, а массово продаваться стал только в конце 1870х годов. Этот арифмометр сделал Шарлем Ксавье Тома, который за своё изобретение получил французский орден Почётного легиона.

В 1822 году Бэббидж построил модель счётной машины, которая вычисляла значения математических функций по интерполяционной формуле Ньютона методом конечных разностей (finit difference), поэтому назвал счётную машину difference engine (разностной машиной).

Первый проект компьютера

В 1833 году Бэббидж решил создать универсальную вычислительную машину, которую он назвал аналитической. В ее состав входили арифметическое устройство, которое Бэббидж назвал "мельница"; запоминающее устройство для хранения 50 чисел ("склад"); устройства ввода-вывода с использованием перфокарт. Разностная машина могла выполнять только одну задачу, а аналитическая смогла бы выполнять произвольные задачи (программы), поэтому она является проектом первого в истории компьютера. Эта машина должна была состоять из 25000 деталей - шестерёнок и вращающихся цилиндров, а приводиться в движение паровым двигателем. Технологии того времени не позволяли построить машину такой сложности.

Компьютеры стало возможным создать только с появлением электричества на электрических реле, лампах и полупроводниковых элементах.

Ада Лавлейс

Бэббиджу помогала Августа (Ада) Лавлейс, родившаяся 10 декабря 1815 года. Ада описала алгоритм вычисления чисел Бернулли на аналитической машине, введя понятие цикла, переменных, инициализации переменных. Это была первая программа, написанная для выполнения на вычислительной машине, поэтому Ада Лавлейс - первый программист в истории человечества.

Ада с детства увлекалась механикой и в 12 лет решила сконструировать летательный аппарат. Начала с проектирования крыльев. Она изучила материалы, среди которых были бумага, промасленный шёлк, проволока и перья. Для определения размера крыльев и формы она изучала анатомию птиц. В 12 лет Ада написала книгу, чтобы не забыть свои наработки.

Первая компьютерная программа

В 28 лет Ада опубликовала под псевдонимом A.A.L. статью об аналитической машине. В статье была представлена программа вычислений из 25 шагов, включая циклы:

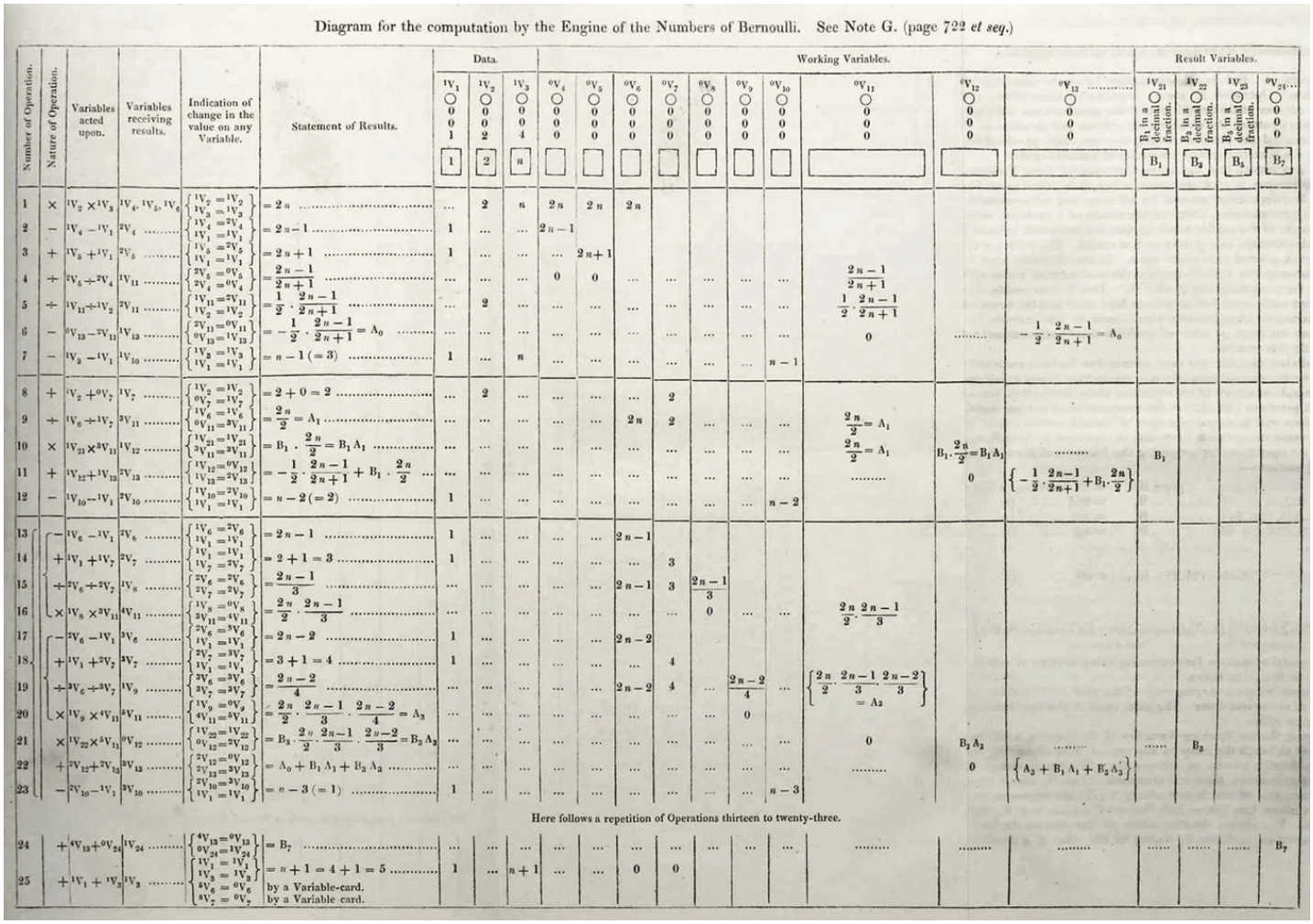


Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 *et seq.*)

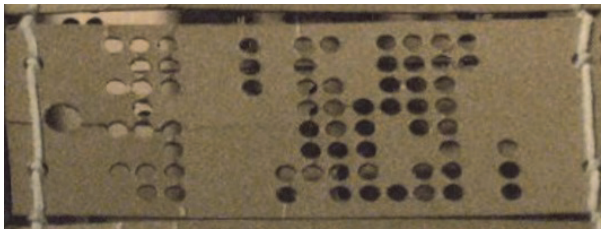
Number of Operation.	Nature of Operation.	Variables acted upon.	Variables receiving results.	Indication of change in the value on any Variable.	Statement of Results.	Data.			Working Variables.								Result Variables.			
						$^1V_1$	$^1V_2$	$^1V_3$	$^0V_4$	$^0V_5$	$^0V_6$	$^0V_7$	$^0V_8$	$^0V_9$	$^0V_{10}$	$^0V_{11}$	$^0V_{12}$	$^0V_{13}$	$^1V_{21}$	$^1V_{22}$	$^1V_{23}$	$^0V_{24}$ ...
						1	2	n											B_1	B_2	B_5	B_7
1	×	$^1V_2 \times ^1V_3$	$^1V_4, ^1V_5, ^1V_6$	$\left\{\begin{matrix}^1V_2 = {}^1V_2 \\ ^1V_3 = {}^1V_3\end{matrix}\right\}$	$= 2n$	...	2	n	$2n$	$2n$	$2n$										
2	−	$^1V_4 - ^1V_1$	$^2V_4$	$\left\{\begin{matrix}^1V_4 = {}^2V_4 \\ ^1V_1 = {}^1V_1\end{matrix}\right\}$	$= 2n-1$	1	...	...	$2n-1$												
3	+	$^1V_5 + ^1V_1$	$^2V_5$	$\left\{\begin{matrix}^1V_5 = {}^2V_5 \\ ^1V_1 = {}^1V_1\end{matrix}\right\}$	$= 2n+1$	1	...	...	...	$2n+1$											
4	+	$^2V_5 \div ^2V_4$	$^1V_{11}$	$\left\{\begin{matrix}^2V_5 = {}^0V_5 \\ ^2V_4 = {}^0V_4\end{matrix}\right\}$	$= \dfrac{2n-1}{2n+1}$	...	...	...	0	0	...	...	...	...	...	$\dfrac{2n-1}{2n+1}$					
5	+	$^1V_{11} \div ^1V_2$	$^2V_{11}$	$\left\{\begin{matrix}^1V_{11} = {}^2V_{11} \\ ^1V_2 = {}^1V_2\end{matrix}\right\}$	$= \dfrac{1}{2} \cdot \dfrac{2n-1}{2n+1}$	...	2	...	...	...	...	...	...	...	...	$\dfrac{1}{2} \cdot \dfrac{2n-1}{2n+1}$					
6	−	$^0V_{13} - ^2V_{11}$	$^1V_{13}$	$\left\{\begin{matrix}^2V_{11} = {}^0V_{11} \\ ^0V_{13} = {}^1V_{13}\end{matrix}\right\}$	$= -\dfrac{1}{2} \cdot \dfrac{2n-1}{2n+1} = A_0$	...	...	...	...	...	...	...	...	...	...	0	.....	$-\dfrac{1}{2} \cdot \dfrac{2n-1}{2n+1} = A_0$				
7	−	$^1V_3 - ^1V_1$	$^1V_{10}$	$\left\{\begin{matrix}^1V_3 = {}^1V_3 \\ ^1V_1 = {}^1V_1\end{matrix}\right\}$	$= n-1\ (=3)$	1	...	n	...	...	...	...	...	...	$n-1$						
8	+	$^1V_2 + ^0V_7$	$^1V_7$	$\left\{\begin{matrix}^1V_2 = {}^1V_2 \\ ^0V_7 = {}^1V_7\end{matrix}\right\}$	$= 2+0=2$	...	2	...	...	...	...	2									
9	÷	$^1V_6 \div ^1V_7$	$^3V_{11}$	$\left\{\begin{matrix}^1V_6 = {}^1V_6 \\ ^0V_{11} = {}^3V_{11}\end{matrix}\right\}$	$= \dfrac{2n}{2} = A_1$	...	...	...	...	...	$2n$	2	...	...	...	$\dfrac{2n}{2} = A_1$					
10	×	$^1V_{21} \times ^3V_{11}$	$^1V_{12}$	$\left\{\begin{matrix}^1V_{21} = {}^1V_{21} \\ ^3V_{11} = {}^3V_{11}\end{matrix}\right\}$	$= B_1 \cdot \dfrac{2n}{2} = B_1 A_1$	...	...	...	...	...	...	...	...	...	...	$\dfrac{2n}{2} = A_1$	$B_1 \cdot \dfrac{2n}{2} = B_1 A_1$	.....	B_1			
11	+	$^1V_{12} + ^1V_{13}$	$^2V_{13}$	$\left\{\begin{matrix}^1V_{12} = {}^0V_{12} \\ ^1V_{13} = {}^2V_{13}\end{matrix}\right\}$	$= -\dfrac{1}{2} \cdot \dfrac{2n-1}{2n+1} + B_1 \cdot \dfrac{2n}{2}$ ...	...	...	...	...	...	...	...	...	...	...	.....	0	$\left\{-\dfrac{1}{2} \cdot \dfrac{2n-1}{2n+1} + B_1 \cdot \dfrac{2n}{2}\right\}$				
12	−	$^1V_{10} - ^1V_1$	$^2V_{10}$	$\left\{\begin{matrix}^1V_{10} = {}^2V_{10} \\ ^1V_1 = {}^1V_1\end{matrix}\right\}$	$= n-2\ (=2)$	1	...	...	...	...	...	...	...	...	$n-2$						
13	−	$^1V_6 - ^1V_1$	$^2V_6$	$\left\{\begin{matrix}^1V_6 = {}^2V_6 \\ ^1V_1 = {}^1V_1\end{matrix}\right\}$	$= 2n-1$	1	...	...	...	...	$2n-1$										
14	+	$^1V_1 + ^1V_7$	$^2V_7$	$\left\{\begin{matrix}^1V_1 = {}^1V_1 \\ ^1V_7 = {}^2V_7\end{matrix}\right\}$	$= 2+1=3$	1	...	...	...	...	...	3									
15	÷	$^2V_6 \div ^2V_7$	$^1V_8$	$\left\{\begin{matrix}^2V_6 = {}^2V_6 \\ ^2V_7 = {}^2V_7\end{matrix}\right\}$	$= \dfrac{2n-1}{3}$	...	...	...	...	...	$2n-1$	3	$\dfrac{2n-1}{3}$								
16	×	$^1V_8 \times ^3V_{11}$	$^4V_{11}$	$\left\{\begin{matrix}^1V_8 = {}^0V_8 \\ ^3V_{11} = {}^4V_{11}\end{matrix}\right\}$	$= \dfrac{2n}{2} \cdot \dfrac{2n-1}{3}$	...	...	...	...	...	...	...	0	...	...	$\dfrac{2n}{2} \cdot \dfrac{2n-1}{3}$					
17	−	$^2V_6 - ^1V_1$	$^3V_6$	$\left\{\begin{matrix}^2V_6 = {}^3V_6 \\ ^1V_1 = {}^1V_1\end{matrix}\right\}$	$= 2n-2$	1	...	...	...	...	$2n-2$										
18	+	$^1V_1 + ^2V_7$	$^3V_7$	$\left\{\begin{matrix}^2V_7 = {}^3V_7 \\ ^1V_1 = {}^1V_1\end{matrix}\right\}$	$= 3+1=4$	1	...	...	...	...	...	4									
19	÷	$^3V_6 \div ^3V_7$	$^1V_9$	$\left\{\begin{matrix}^3V_6 = {}^3V_6 \\ ^3V_7 = {}^3V_7\end{matrix}\right\}$	$= \dfrac{2n-2}{4}$	...	...	...	...	...	$2n-2$	4	...	$\dfrac{2n-2}{4}$	...	$\left\{\dfrac{2n}{2} \cdot \dfrac{2n-1}{3} \cdot \dfrac{2n-2}{3}\right\} = A_2$					
20	×	$^1V_9 \times ^4V_{11}$	$^5V_{11}$	$\left\{\begin{matrix}^1V_9 = {}^0V_9 \\ ^4V_{11} = {}^5V_{11}\end{matrix}\right\}$	$= \dfrac{2n}{2} \cdot \dfrac{2n-1}{3} \cdot \dfrac{2n-2}{4} = A_3$	...	...	...	...	...	...	...	...	0	...						
21	×	$^1V_{22} \times ^5V_{13}$	$^0V_{12}$	$\left\{\begin{matrix}^1V_{22} = {}^1V_{22} \\ ^0V_{12} = {}^2V_{12}\end{matrix}\right\}$	$= B_3 \cdot \dfrac{2n}{2} \cdot \dfrac{2n-1}{3} \cdot \dfrac{2n-2}{3} = B_3 A_3$	...	...	...	...	...	...	...	...	...	...	0	$B_3 A_3$	.....	.....	B_3	
22	+	$^2V_{12} + ^2V_{13}$	$^3V_{13}$	$\left\{\begin{matrix}^2V_{12} = {}^0V_{12} \\ ^2V_{13} = {}^3V_{13}\end{matrix}\right\}$	$= A_0 + B_1 A_1 + B_3 A_3$	...	...	...	...	...	...	...	...	...	...	.....	0	$\{A_2 + B_1 A_1 + B_2 A_2\}$			
23	−	$^2V_{10} - ^1V_1$	$^3V_{10}$	$\left\{\begin{matrix}^2V_{10} = {}^3V_{10} \\ ^1V_1 = {}^1V_1\end{matrix}\right\}$	$= n-3\ (=1)$	1	...	...	...	...	...	...	...	...	$n-3$						
					Here follows a repetition of Operations thirteen to twenty-three.																	
24	+	$^4V_{13} + ^0V_{24}$	$^1V_{24}$	$\left\{\begin{matrix}^4V_{13} = {}^0V_{13} \\ ^0V_{24} = {}^1V_{24}\end{matrix}\right\}$	$= B_7$	...	...	...	...	...	...	...	...	...	...	.....	.....	.....	.....	.....	.....	B_7
25	+	$^1V_1 + ^1V_3$	$^1V_3$	$\left\{\begin{matrix}^1V_1 = {}^1V_1 \\ ^1V_3 = {}^1V_3\end{matrix}\right\}$	$= n+1 = 4+1=5$	1	...	$n+1$	...	...	0	0										
				$\left\{\begin{matrix}^5V_6 = {}^0V_6 \\ ^5V_7 = {}^0V_7\end{matrix}\right\}$	by a Variable-card. by a Variable card.																	

На каждом шаге программы расписано, какая выполняется операция над какими переменными, в какую переменную записать результат. Циклы обозначены фигурными скобками. Окончательный результат вычислений B7 числа Бернулли записывается на 24 шаге в переменную V24.

Приоритет первой ошибки (bug, произносится как "баг") в компьютерной программе тоже принадлежит Аде: в 4 строке, 3 столбце вместо V5/V4 должно быть V4/V5.

Перфокарты

В аналитической машине Бэббиджа предполагалось использовать перфорированные карточки ("перфокарты") для ввода данных и печати результатов. Поэтому в программе Ады используется термин "Variable card". Перфокарты были изобретены Жаккардом в 1804 году для ткацких станков. Ткань, изготовленную на станках Жаккарда, стали называть жаккардовой.
Фамилия Жаккард созвучна с фамилией Шиккарда, который жил на столетие раньше.


перфокарта Жаккарда

Отверстия на перфокарте кодировали информацию об узоре на ткани. Можно сказать, что перфокарты определяли программу нанесения рисунка на ткань. Компьютером ткацкий станок назвать нельзя, так как не было даже минимальных вычислений. Для большого полотна ткани использовалось много перфокарт, которые соединялись вместе в ленту. Почему использовались карты, а не перфолента? Из перфокарт можно было составлять новые узоры, меняя набор перфокарт. Вторая причина - если испортится перфолента, то придется ее полностью менять, а при использовании перфокарт достаточно заменить одну карту.

ткацкий станок с перфокартами

Перфокарты удобны для записи чисел и других знаков в двоичной форме. Отверстие означает единичку, отсутствие отверстия нолик. Наличие отверстия можно считывать механическими щупами.

В 20 веке перфокарты и перфоленты использовались с компьютерами для хранения программ и данных. Они перестали использоваться после изобретения магнитных носителей информации - лент и дисков.

Для вывода результатов вычислений более удобным были другие средства отображения: колесики с нарисованными числами, лампочки, печатные машинки.

Причины, почему перфокарты не использовались для вывода результатов вычислений:

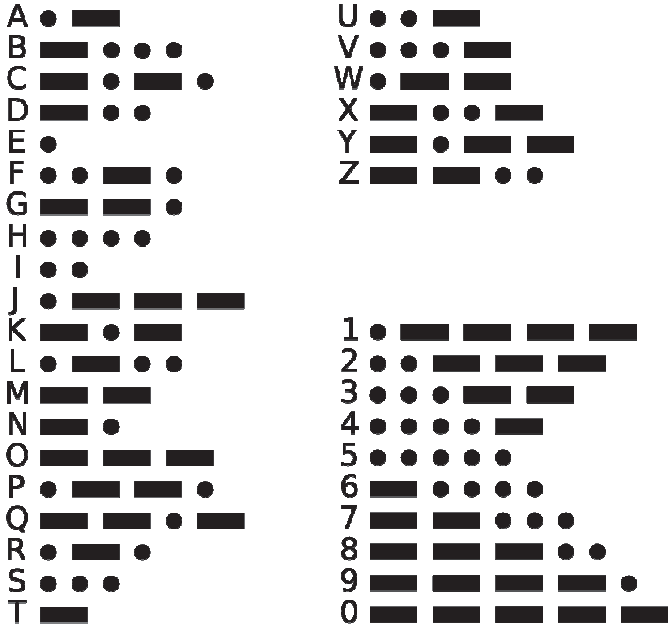
1) результаты вычислений просматривает человек, а для человека неудобно считывать символы с перфокарт;

2) результаты вычислений каждый раз разные и приходилось бы использовать каждый раз новую карту или ленту.

Азбука Морзе

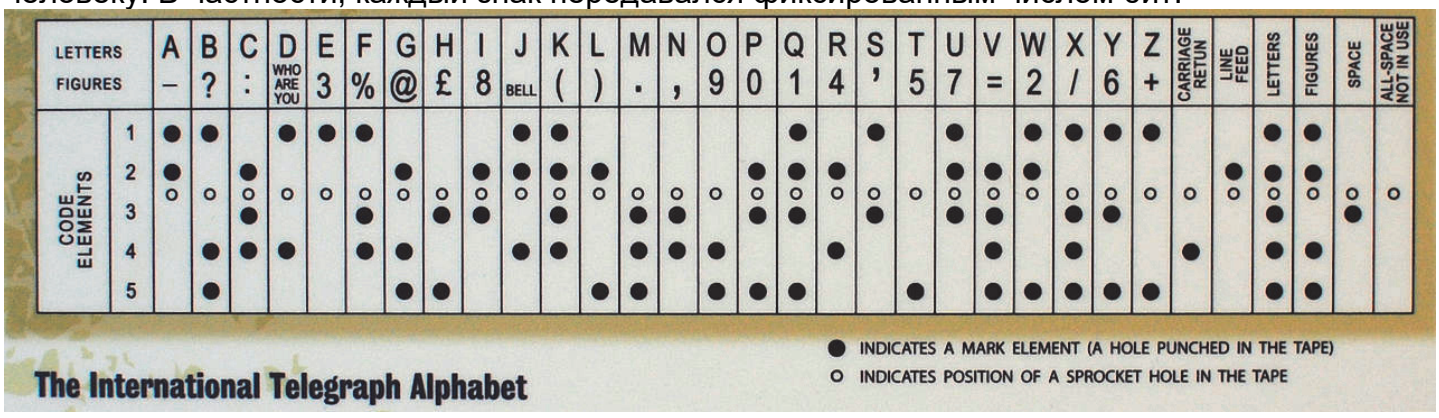
В 1838 году появился телеграфный аппарат со схемой кодирования символов Морзе. К компьютерной технике азбуку Морзе не относят, хотя для передачи используются короткие ("точка"), длинные ("тире") сигналы и промежутки между ними. Каждая буква кодировалась от 1 до 4 сигналами. Цифры передавались 5 сигналами. Чем чаще встречалась буква в английской речи, тем меньше сигналов использовалось. Для ввода-вывода в компьютерных устройствах азбука Морзе никогда не применялась.

Телеграфный аппарат и азбука Морзе - пример коммерчески успешных изобретений, но не имеющих будущего или теоретической (научной) ценности. Из языков программирования тупиковой ветвью является язык COBOL, хотя на нем в 1960-70х годах было написано много программ, которые работают до сих пор. Компьютеров Commodore 64 с 1982 по 1994 год было продано более 12,5 миллионов. Эта модель компьютеров попала в книгу рекордов Гиннеса, как самая продаваемая в мире. Линейка компьютеров Commodore перестала существовать, а компания-производитель Commodore International объявила о банкротстве в 1994 году, была куплена немецкой компанией Escom, которая через год после покупки Commodore International, также обанкротилась, хотя до этого была ведущим производителем IBM PC совместимых компьютеров в Европе с оборотом 2 миллиарда долларов в год.



код Морзе

В 1870е годы Бодо создал код, в котором каждый знак передавался 5 битами, для телеграфного аппарата, который он же и изобрёл. Код Бодо был ближе к машинам, чем к человеку. В частности, каждый знак передавался фиксированным числом бит.



один из вариантов кода Бодо

В 1963 году появился семибитный код ASCII и пятибитные коды стали вытесняться. В 1927 году в честь Бодо была названа историческая единица измерения скорости передачи знаков -

бод. Для двоичных данных бод равен числу бит, передаваемых в секунду. Например, если знак кодируется 8 битами, то 1 бод равен 8 битам в секунду.

перфолента с 5-битным кодом

IBM

В 1890 году Холлерит сконструировал суммирующую машину (табулятор), которая победила в конкурсе на обработку данных переписи населения США. В табуляторе использовались перфокарты. В 1896 году Холлерит создал фирму Tabulating Machine Company для производства счётных машин и перфокарт. Машины оказались полезными не только для переписи населения, но и в других отраслях экономики. В 1911 году компания объединилась с другими компаниями и поменяла название на Computing-Tabulating-Recording Company. В 1914 году компанию возглавил Том Ватсон, проработавший в ней до своей смерти в 1956 года. В 1924 году компанию переименовали в International Business Machines Corporation (IBM).

В 1944 году при участии IBM был создан релейный компьютер "Марк I". В 1952 году появился ламповый компьютер IBM 701, давший начало серии компьютеров IBM 700/7000, которые стали называть "мэйнфреймы". Компьютеры этой серии выпускались до 1964 года. В 1964 году была выпущена серия IBM System/360, которая намного превосходила продукцию конкурентов и сделала IBM монополистом на рынке мэйнфреймов. Мэйнфрейм - универсальный, отказоустойчивый компьютер с большим объёмом оперативной и внешней памяти.

В мэйнфреймах для программирования использовались языки FORTRAN, PL/1, Algol, COBOL, LISP. В 1960-х годах IBM занимала 70% рынка компьютеров в мире.

В 1971 году компания IBM создала гибкий магнитный диск, который заменил перфоленты и на десятки лет стал стандартом для хранения данных.

В 1981 году компания создала IBM PC - персональный компьютер, архитектура которого стала стандартом для компьютерной отрасли и персональные компьютеры стали использоваться повсеместно: в рабочих и домашних целях.

Теорема Найквиста-Шеннона

По теореме аналоговый сигнал с ограниченной максимальной частотой может быть передан и восстановлен числами, передаваемыми с частотой в два раза чаще максимальной частоты аналогового сигнала. Числа хранят измерение величины функции (амплитуду). Как следствие, для передачи звука с частотой до 22КГц (максимальная частота звука, которую слышит большинство людей) можно использовать частоту дискретизации 44.1КГц (использовалась в первых стандартах оцифровки звука для записи на компакт-диск, compact-disk digital audio, CD-DA). Амплитуду звука можно измерять 16-битными числами, которые дают `2^16=65536` уровней звука.

Теорема Найквиста-Шеннона лежит в основе цифровой передачи аналоговых сигналов по цифровым каналам. Для вычислительной техники более актуальна передача цифровых сигналов по аналоговым каналам передачи и Шеннон в 1948 опубликовал статью "Математическая теория связи", где развил идеи Найквиста и ввел понятие информационной энтропии, которая является мерой неопределённости информации.

Идея Шеннона состояла в том, что количество информации, которое возможно передать зависит от энтропии (случайности сообщений в источнике сигнала). Основываясь на статистической характеристике источника сообщений, можно закодировать информацию так, чтобы достичь максимальной скорости передачи данных, которая определяется теоремой. В то время это было воспринято как важное достижение, так как ранее полагали, что максимум информации исходного сигнала, которое можно передать через среду, зависит от свойств канала (частоты), но не от свойств сигнала. Сжатие данных перед передачей не было известно, так как ещё не существовало вычислительной техники и алгоритмов, которые могли выполнять сжатие.

Информатика и кибернетика

Норберт Винер в книге 1948 года "Кибернетика, или управление и связь в животном и машине" написал, что послал научному руководителю Шеннона принципы построения компьютеров:

1) Суммирующие и множительные устройства должны быть цифровыми (Буш создал аналоговую вычислительную машину);

2) Суммирующие и множительные устройства, являющиеся по существу переключателями, должны состоять из электронных ламп, а не зубчатых передач или электромеханических реле. Это нужно, чтобы обеспечить приемлемое быстродействие;

3) Должна использоваться более экономичная двоичная, а не десятичная система счисления;

4) Последовательность действий должна планироваться компьютером так, чтобы человек не вмешивался в процесс решения задачи с момента введения исходных данных до получения окончательных результатов;

5) Компьютер должен иметь устройство для накопления данных. Это устройство должно быстро их записывать, надежно хранить до стирания, быстро считывать, быстро стирать их и немедленно подготавливаться к получению новых данных.

Принципы оказались точными и предсказали, как будут развиваться компьютеры. По пятому принципу, довольно быстро, в 1956 году, появился первый накопитель на магнитных дисках IBM RAMAC объёмом 5 Мегабайт.

Винер и его коллеги решили назвать теорию управления и связи в машинах и живых организмах "кибернетикой", от греческого слова рулевой (cubernetes). Они считали, что рули кораблей были одними из первых устройств с обратной связью. Коллегой у Винера был доктор медицины Артуро Розенблют (Arturo Rosenblueth) с фамилией, созвучной фамилии Фрэнка Розенблатта (Frank Rosenblatt), создателя перцептрона.

Винер считал, что "информация это информация, а не материя и не энергия".

В 1948 году ещё не было термина "информатика" и использовался термин "кибернетика". Термин informatik впервые появился в 1957 году в статье на немецком языке. Во Франции термин informatique появился в 1962 году. Термин computer science появился в 1959 году. Computer science используется в англоязычных странах, информатика в других. Оба слова обозначают одно и то же.

Дейкстра также считал, что "информатика имеет к компьютерам не больше отношения, чем астрономия к телескопам".

Часть 2. Появление компьютеров

Булева алгебра

Любые арифметические вычисления с числами можно выразить с помощью трех логических операторов **AND** (`&`, и, ^, конъюнкция), **OR** (`|`, или, v, дизъюнкция) и **NOT** (`!`, не, отрицание, инверсия). Слова конъюнкция, дизъюнкция, инверсия произошли от латинских слов. Использование латыни было распространено в средние века. В настоящее время использование латинских слов затрудняет понимание.

Вычислительную часть компьютера можно построить из комбинации трёх элементов, реализующих логические операторы: `AND OR` и `NOT`. Сначала числа приводятся к двоичной форме, а потом производятся вычисления путём применения к битам логических операторов.

Кроме **этих трёх** операторов существуют логические операторы, которые можно создать путем их комбинирования:

1) `NAND`: образуется комбинацией элементов `NOT(x AND y)`. Сначала к двум операндам применяется AND, а потом к результату NOT. Что интересно, NAND обладает свойством "функциональной полноты", что означает, что любая логическая функция может быть реализована с помощью набора только элементов NAND.

2) `NOR: NOT(x OR y)`. Возвращает 1 только, если оба операнда равны 0. Оператор NOR тоже обладает "функциональной полнотой". Например:

```
(NOT x) = (x NOR x)
(x AND y) = (x NOR x) NOR (y NOR y)
(x OR y) = (x NOR y) NOR (x NOR y)
XNOR: NOT(x XOR y).
```

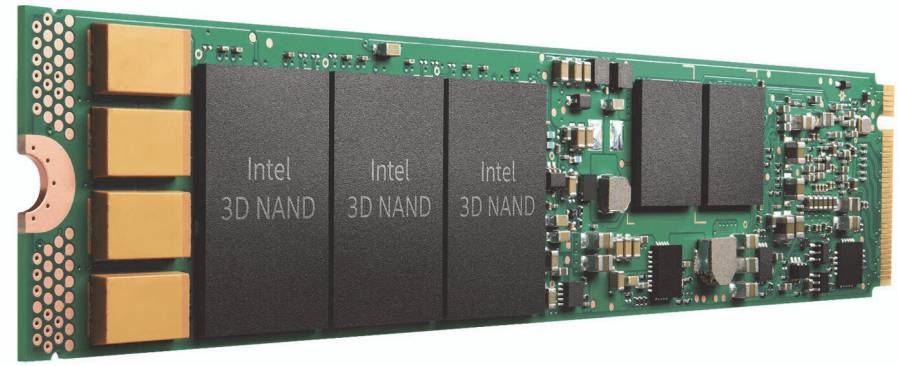
3) ≡, тождественность: `(x == y)`. Если оба операнда равны 0 или оба операнда равны 1, то оператор возвращает 1.

4) `XOR`: исключающее OR. `(x XOR y)` возвращает 0, если значения операндов одинаковы, а если значения операндов различны, то возвращает 1. `XOR` примечателен тем, что `(x XOR (x XOR y) = y`. Используя XOR можно поменять местами значения двух переменных одинакового типа данных, не используя временную переменную.

Как заменить XOR **тремя** операторами?

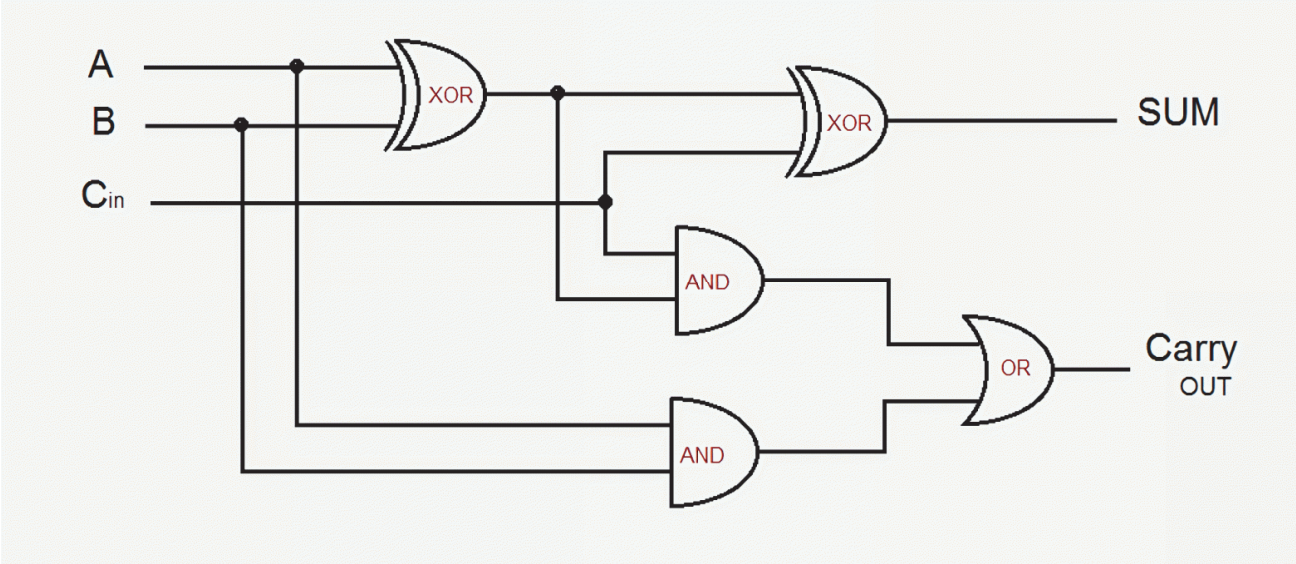
```
(x XOR y) = (x AND (NOT y) OR ((NOT x) AND y) = ((NOT x) OR (NOT y)) AND (x OR y).
```

Операторы `NAND`, `NOR`, ≡, `XOR` полезны тем, что их использование на уровне железа более эффективно, чем построение схем, реализующих **три** основных оператора. Технологии изготовления флэш-памяти называются NAND-флэш и NOR-флэш потому, что для хранения битов используются наборы транзисторов, которые реализуют логический элемент NAND или NOR.



Создание схем из логических элементов

Для сложения двоичных чисел используются схемы "сумматоров". Например, нужно сложить два двоичных числа: `01 + 01 = 10`. Правый разряд - наименьший (младший). В десятичной записи сложение этих двух чисел выглядит так: `1+1=2`. Для сложения нужно выполнить операцию с правыми битами и если в них обоих стоят единички, то результат будет ноль, а единичка должна перейти в разряд левее (старший разряд). Эту задачу и выполняют "сумматоры".



A и B - разряды, значения которых надо сложить (суммировать). Cin - сигнал от меньшего разряда. CarryOUT - выходной сигнал для старшего разряда, который соединится с Cin такого же сумматора, только для более старшего разряда. SUM - результат суммирования для разряда текущей схемы сумматора.

В примере сложения `01 + 01 = 10` для младшего разряда: Cin=0 (так как разряд самый младший, то контакт отсутствует и значит 0), A=1, B=1, SUM=0, CarryOUT=1.

Для старшего разряда сложения `01 + 01 = 10`: Cin=1, A=0, B=0, SUM=1, CarryOUT=0.

Для реализации сложения двухразрядных двоичных чисел нужно два сумматора.

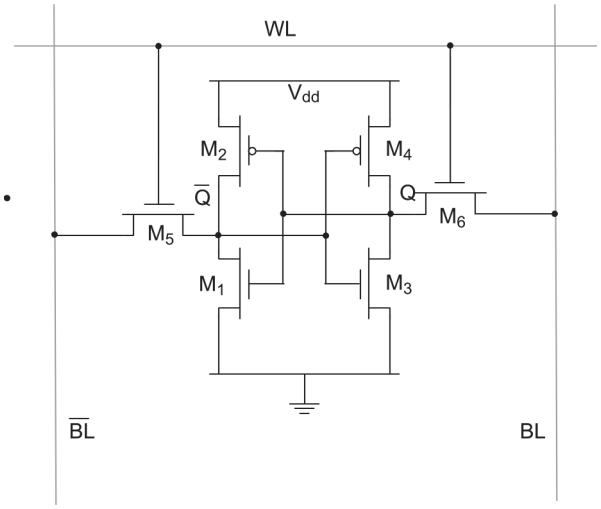
Современные компьютеры работают с 64-разрядными двоичными числами и число сумматоров довольно большое - тоже 64. В каждом сумматоре по 5 логических элементов `XOR, AND, OR`. Каждый логический элемент можно реализовать 6 транзисторами или другими элементами. Интересно, что ячейка памяти SRAM также использует 6 транзисторов. Итого на один сумматор 5*6=30 транзисторов, а для суммирования 64-разрядных чисел 30*64=1920 транзисторов. Это довольно много. Ещё больше число соединений между выводами элементов. До появления микросхем размеры компьютеров были внушительные, а длина проводов измерялась сотнями километров. Для точности можно сказать, что при проектировании без микросхем, схемы оптимизировались и число транзисторов (ламп, реле) для реализации логического элемента могло быть не 6, а 2-3, но приходилось добавлять резисторы. Резисторы выделяют тепло, поэтому их стараются не использовать в микросхемах.

SRAM и DRAM

Доступ к оперативной памяти в компьютерах произвольный (random access). Альтернатива произвольному доступу - последовательный доступ, как при чтении с ленты. Чтобы прочесть содержимое, записанное на ленте, нужно промотать ленту, а на это тратится время.

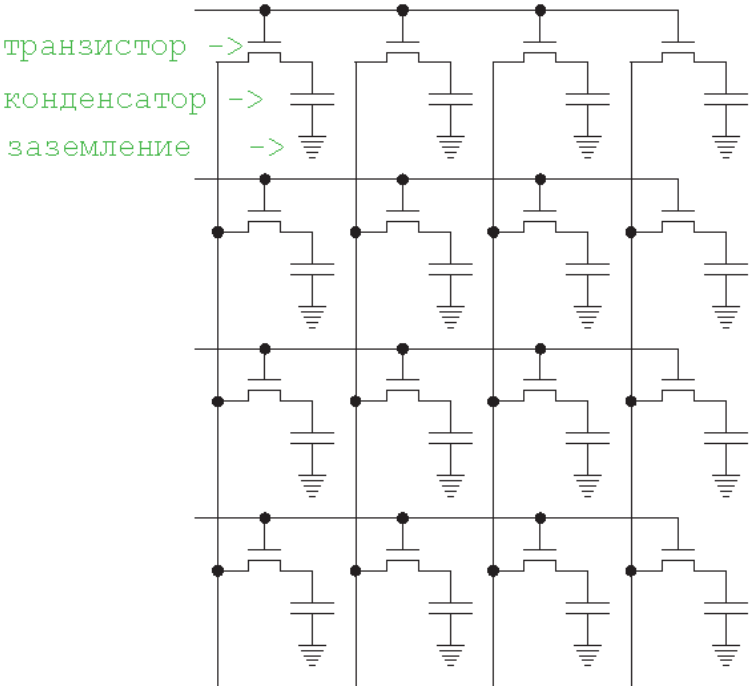
Для оперативной памяти используют:

1) SRAM (static random access memory) - статическая оперативная память. Называется так потому, что содержимое памяти не нужно периодически обновлять. Пример ячейки (хранит один бит) памяти SRAM:



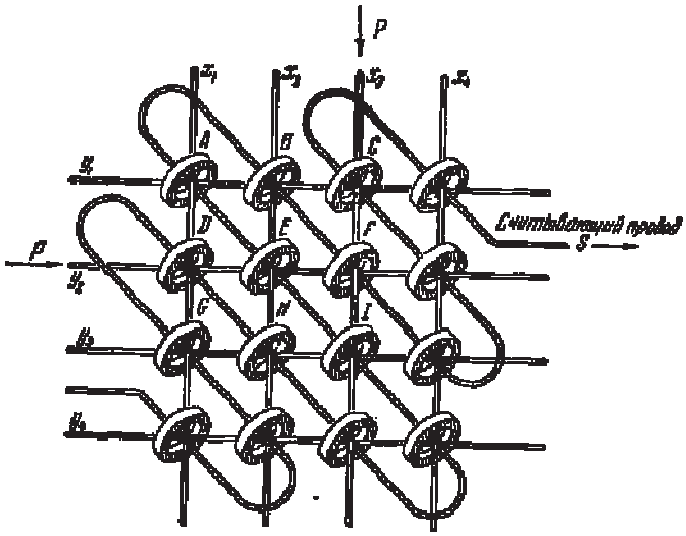
Для одного бита памяти SRAM нужно шесть транзисторов. Чтение и запись без задержек, так как не нужно выполнять подготовительных действий для доступа к ячейкам памяти.

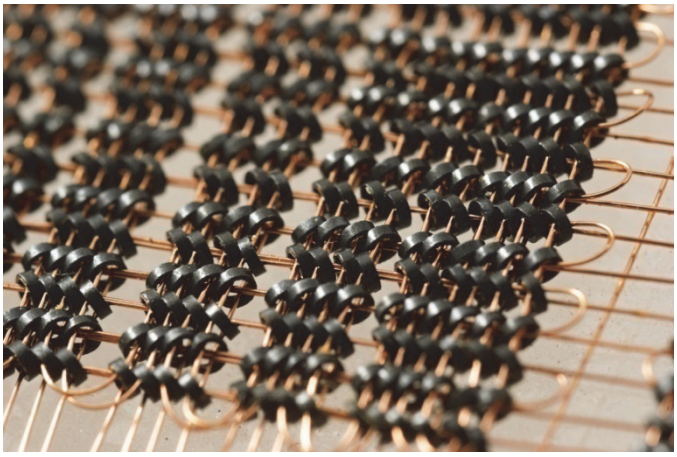
2) DRAM (dynamic random access memory) - динамическая память. Для хранения бита использует один транзистор и один конденсатор (ёмкость для накопления заряда из электронов), которые располагаются на кремниевом чипе. Заряженный конденсатор хранит ноль, заряженный единицу. Конденсатор из-за токов утечки разряжается со временем, поэтому конденсаторы надо периодически подзаряжать, отсюда и произошло название "динамическая", так как постоянно протекают процессы, изменяющие состояние памяти, даже если к памяти нет обращений. Транзистор используется для заряда-разряда конденсатора и как усилитель тока, чтобы можно было стабильно определить заряжен или разряжен конденсатор. Пример схемы 16 ячеек DRAM:



Для одной ячейки DRAM нужен транзистор и конденсатор. Периодически нужно освежать содержимое всех ячеек. Пока считывается или записывается одна ячейка в наборе ячеек (на рисунке по горизонтали или по вертикали), доступа к другим ячейкам нет, что снижает производительность работы с такой памятью. На зарядку-разрядку конденсаторов тоже нужно время.

До появления микросхем, на которые можно нанести слои конденсаторов и транзисторов, использовалась память на ферритовых кольцах:





Скорость намагничивания-размагничивания сравнима с зарядом-разрядом конденсаторов. Недостаток ферритовых колец в том, что размер каждого кольца был не меньше четверти миллиметра, что довольно много.

Идея использовать магнитный момент вместо электрических зарядов получила развитие в магнитно-резистивной памяти MRAM. Быстродействие такой памяти сравнимо с SRAM. Преимущества магнитной памяти в том, что радиация не влияет на магнитное поле, что позволяет использовать и ферритовые кольца и MRAM в космосе.

Релейные компьютеры

В 1937 году Шеннону было 22 года, когда он в магистерской диссертации "Символический анализ релейных и коммутационных схем" описал способ реализации операторов двоичной логики с помощью электронных реле и переключателей. Этим он заложил основу проектирования цифровых схем для будущих компьютеров.

Если последовательно соединить два нормально разомкнутых (то есть без напряжения) реле, получится логический элемент AND - сигнал на выходе будет тогда, когда напряжение будет подано на оба реле.

Изображение “И”

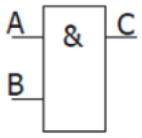
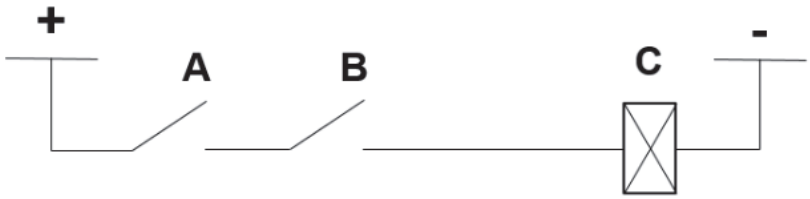


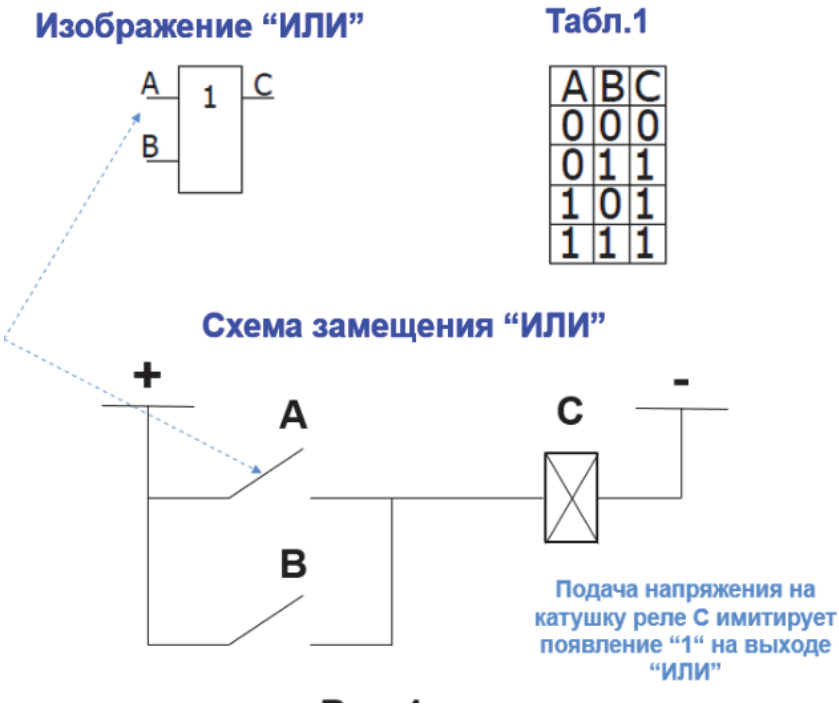
Табл.2

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

Схема замещения “И”



Если соединить реле параллельно, то получится логический элемент OR - сигнал на выходе будет, если хотя бы на одно реле подано напряжение.



Два последовательно соединенных нормально замкнутых реле дадут логический элемент NOR. Если нормально замкнутые реле соединить параллельно получится логический элемент NAND.

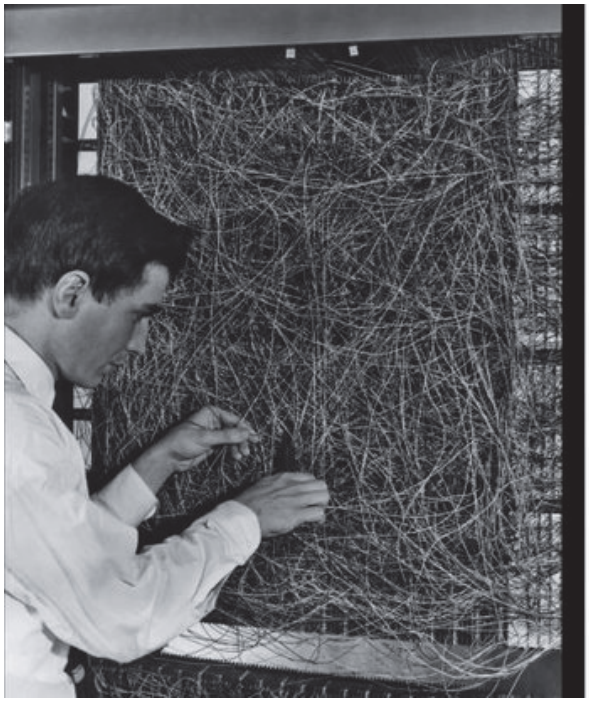
В 1944 году был построен компьютер (вычислительная машина с возможностью программирования) Mark I на основе электромеханических реле. Он весил 35 тонн, содержал примерно 755 тысяч деталей, 800 километров проводов и 3 миллиона соединений. Mark I складывал за 0.3 секунды, умножал за 3 секунды, делил за 15 секунд 23-разрядные десятичные числа. Тригонометрические функции и логарифмы вычислялись больше минуты. Марк I считывал и выполнял инструкции с широкой перфорированной бумажной ленты. Условных переходов среди команд не было и программа представляла собой длинный ленточный рулон.

Недостаток реле - низкая скорость срабатывания. Это ограничивало производительность релейных компьютеров. Норберт Винер указывал на это в своих принципах построения компьютеров.

Слово Mark - традиционное обозначение порядковой модели, версии. Это название присутствует и в названии британского компьютера "Colossus Mark 2", не имевшего отношения к Mark I.

Перцептрон

Название "MARK 1" также имело небольшое устройство, созданное Розенблаттом в 1958 году, которое называют "первым нейрокомпьютером". Устройство не было компьютером, оно реализовывало понятие "перцептрона" и было способно с какой-то вероятностью распознавать буквы английского алфавита. Перцептрон - модель восприятия информации мозгом - то, как представляли себе работу нейронов мозга. Изучая нейронные сети типа перцептрона, Розенблатт надеялся понять фундаментальные законы организации, общие для всех систем обработки информации, включая машины и человеческий разум. Эти исследования в то время не получили развития по причине отсутствия практических результатов. Перцептрон стал первой моделью нейросети.



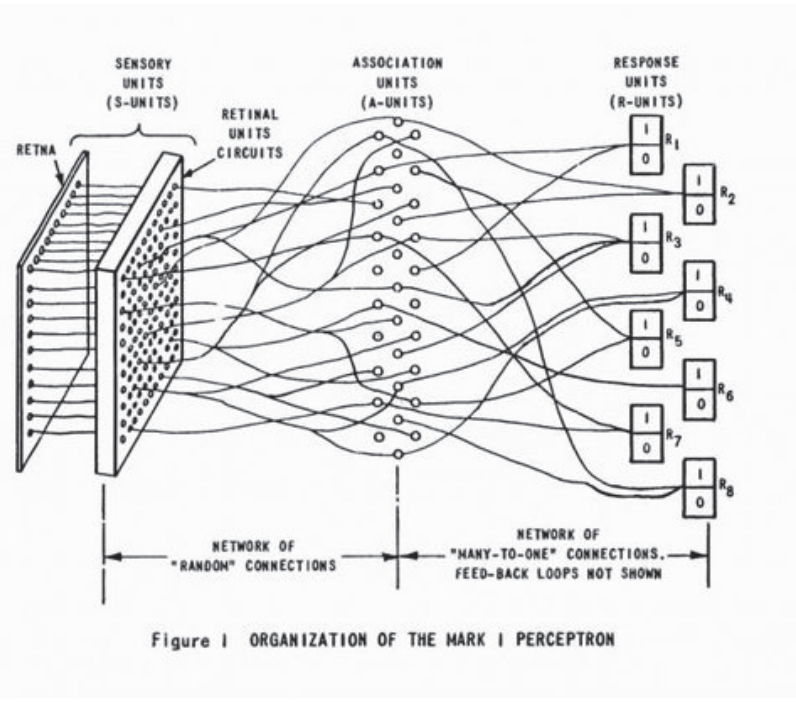


Figure 1 ORGANIZATION OF THE MARK I PERCEPTRON

Мозг обрабатывает данные, которые получает от органов чувств, в основном, от зрения и слуха. В процессе обработки, человек извлекает для себя что-то полезное - информацию. Данные - это очертания букв, по которым распознаются буквы. Перцептрон считывал очертания букв и пытался распознать, какая буква изображена.

Речь, являющаяся сложной системой обмена сигналами, позволила человеку развить мышление, обмениваться информацией и сохранять её в памяти других людей или письменно. Речь позволила совершить людям качественный скачок в развитии, по сравнению с животными. В процессе развития речи появились языки и письменность.

Большие языковые модели

Большая языковая модель (LLM, Large Language Model) - языковая модель, состоящая из нейронной сети с миллиардами параметров, самообучавшаяся (то есть без указаний: это утверждение верное, а это ложь и фантазия) на огромных объемах текстов. Большие объемы данных и параметров позволили совершить качественный скачок: выводы и заключения, которые генерируются LLM, более-менее логичны.

До появления LLM использовались языковые модели, в которых обучение было "контролируемым" (supervised, под руководством). Контролируемое означает, что языковой модели даётся ответ: правильно или не правильно она выдала заключение, что позволяет языковой модели подстраивать свои внутренние параметры и делать выводы более точно. Языковые модели успешно используются в узкоспециализированных целях: поиске аналогий и прогнозировании. Например, в распознавании лиц, распознавании песен по фрагменту, как в приложении для смартфонов Shazam.

LLM применяются для извлечения информации из текста, создания дайджестов (кратких описаний), ответов на вопросы, перевода другие языки.

Генеративный искусственный интеллект использует LLM для генерации текста, изображений в ответ на подсказки того, кто ставит ему задачи.

Генеративный искусственный интеллект может использовать текст, состоящий из слов; программный код; изображения для генерации картинок и видео; схемы молекул, последовательности аминокислот для генерации молекул и описанием свойств, которыми они могут обладать.

Ламповые компьютеры

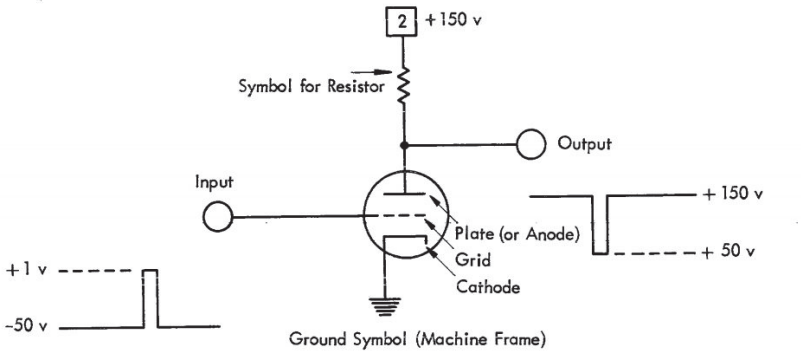
Быстродействие ламп гораздо выше, чем реле и лампы стали использоваться вместо реле.

В 1945 году было завершено создание Electronic Numerical Integrator And Computer (ENIAC) - компьютера на основе ламп, который успешно проработал до 1965 года. В нём использовались 17.5 тысяч ламп, 7 тысяч диодов, 1.5 тысячи реле.

С 1951 по 1958 год выпускались ламповые компьютеры UNIVAC I. Они состояли из 5200 ламп, весили 13 тонн и занимали площадь 4*2.5 метра. В качестве устройства вывода использовалась электрическая печатная машинка. В качестве внешнего накопителя данных - магнитная лента.

Ламповые компьютеры считаются первым поколением компьютеров.

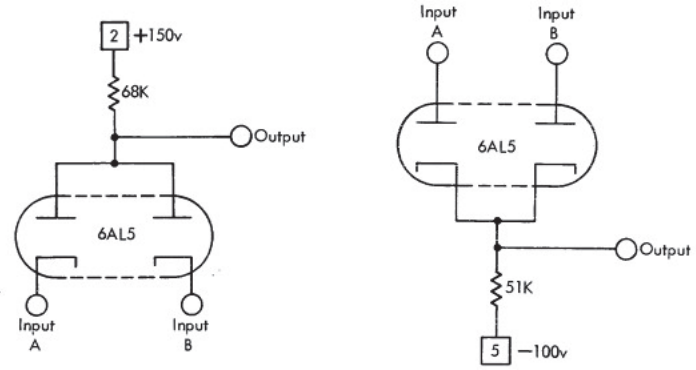
Пример реализации логики NOT на ламповом триоде:



При появлении напряжения +1v на контакте Input, триод "открывается" - электроны начинают перемещаться от катода к аноду и напряжение на выходе падает со 150 вольт до 50 вольт.

Недостаток ламп в высоком напряжении, температуре работы, необходимости настройки (наладки) и частых поломках. Настройка заключалась в проверке того, чтобы лампы переключались в заданных диапазонах напряжений. Поломки возникали часто и даже чаще, чем в релейных машинах.

Пример реализации элементов на вакуумных диодах:



слева реализация логики AND, справа реализация логики OR

В одной вакуумной колбе объединено два диода. Разница между логическими элементами в номиналах резисторов и напряжении. Для логики AND нужно, чтобы оба диода начали пропускать ток только при подаче напряжения на входы A и B. Ламповые диоды начинают пропускать ток скачкообразно, когда разница напряжений превысит порог, за которым электроны смогут перемещаться в вакууме от катода к аноду.

Для логики AND номинал резистора и напряжение подобраны так, что для открытия любого из диодов нужно чтобы напряжение появилось на обоих входах, появления напряжения на одном из входов не достаточно.

Для логики OR номинал сопротивления и разница напряжений меньше. Для открытия любого из диодов достаточно подачи напряжения на один из них.

Хотя лампы могут работать на высоких частотах, они требуют больших токов и напряжений. Использование ламп для коммутации вызывает скачки тока в цепях питания. Увеличение частоты переключений увеличивает число ошибок и ограничивает производительность ламповых компьютеров.

Транзистор

В 1947 году был создан транзистор (**tran**sfer re**sistor**). Логика работы транзистора похожа на ламповый триод, но размеры, напряжение, температура гораздо меньше. Также полупроводниковые элементы не изнашиваются и не подвержены механическим воздействиям. После начала промышленного производства транзисторов, они стали использоваться в компьютерах вместо ламп. Как реле и лампы, транзисторы используются в вычислительной технике как переключатели:

1) транзистор "открыт" ("включён", пропускает ток) - соответствует единице;

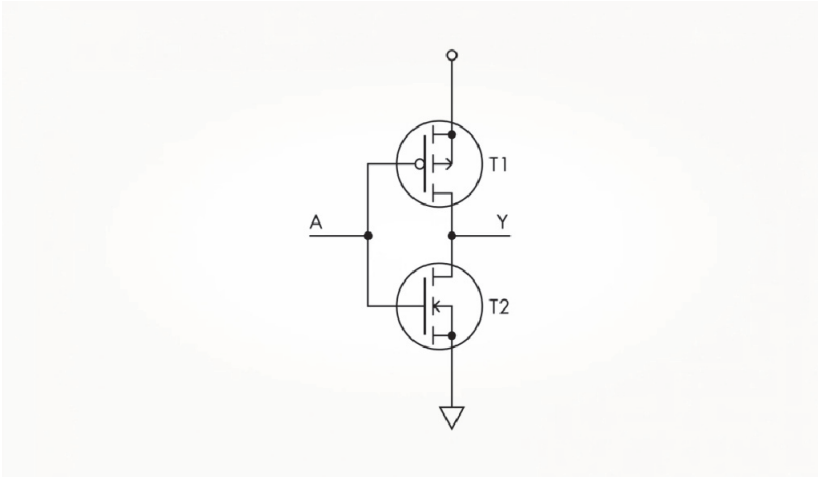
2) транзистор "закрыт" (не пропускает ток) - соответствует нулю.

Большие токи в режиме переключателей через транзисторы не проходят, напряжения тоже небольшие (несколько вольт), в отличие от ламп, у которых напряжение около сотни вольт. Уровень единицы у транзисторов 3.3-5 вольт, уровень нуля 0-1,6 вольт. Напряжение питания микросхем различается, но лежит в диапазоне 3-15 вольт.

Транзисторы бывают двух видов: с `n` и `p` каналами. Одно и то же напряжение открывает транзистор с `n`-каналом и закрывает транзистор с `p`-каналом, то есть они дополняют друг друга (комплиментарны). На схеме транзисторы с `n` и `p` каналами отличаются направлением стрелочек и/или наличием маленького кружочка у затвора.

Большой круг в схемах обозначает наличие корпуса у транзистора. В схемах применяются разнообразные обозначения. Например, для схем с полевыми транзисторами напряжение питания обозначают Vdd (drain, сток, куда стекают электроны, плюс), Vss (source, исток, источник электронов, минус).

Считается, что ток течёт от плюса к минусу и это запутывает. Это произошло потому, что электроны открыли позже электричества, не знали, что они являются носителями заряда и приняли направление тока: от плюса к минусу, так как при гальванизации атомы металла перетекают с плюса и осаждаются на минусе.

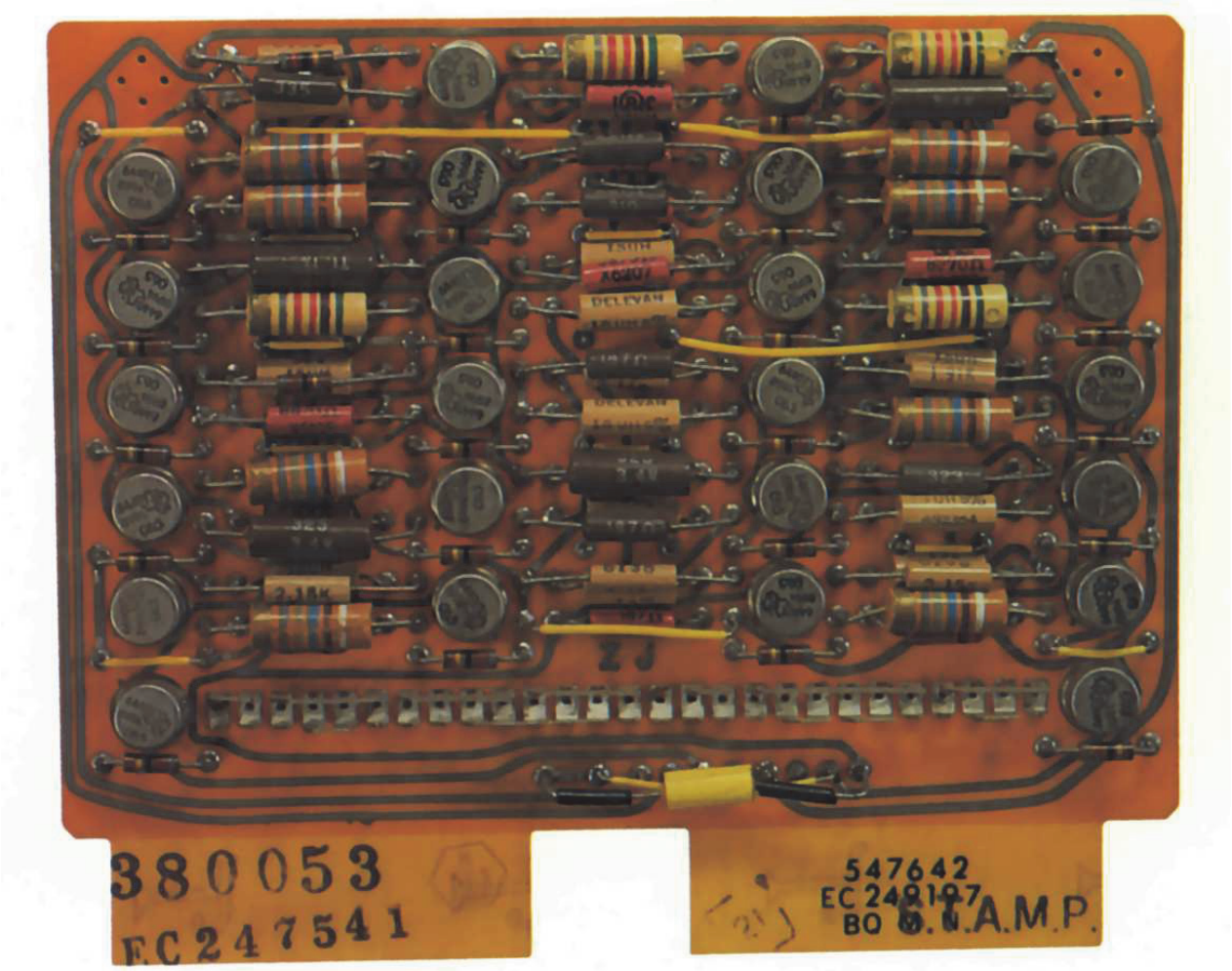


реализация логического элемента NOR на полярных транзисторах

Компьютеры, использующие транзисторы, относят ко второму поколению компьютеров, а использующие микросхемы - к третьему поколению.

Ко второму поколению относится компьютер БЭСМ-6. С 1968 по 1987 года было выпущено 355 машин.

К третьему поколению относится серия IBM System/360. Серия начала выпускаться с 1965 года. На смену 360 пришла серия 370 (анонсированная в 1970 году), затем серия 390 и System z. Пример платы транзисторного компьютера IBM 7090:



Круглые детали это транзисторы. Полосатые детали - резисторы (сопротивления). Цветными полосками закодировано значение сопротивления резисторов.

Компьютер IBM 7090 в Стэндфордском университете:

Микросхемы

При увеличении числа транзисторов, резисторов, конденсаторов надежность компьютеров ухудшается, а время поиска неисправностей увеличивается. Причина в том, что становится много контактов, а они могут окислиться и перестать проводить ток.

Так как рассеиваемая мощность при использовании транзисторов как переключателей ("вентилей") для реализации логических элементов невелика, то транзисторы можно сделать миниатюрными. Транзисторы можно объединить в один корпус (микросхему). Эта идея появилась в 1952 году и в 1959 году было налажено серийное производство микросхем.

В цифровой технике используются микросхемы, выполненные по технологии КМОП (комплиментарная структура металл-оксид-полупроводник; CMOS, complementary metal-oxide-semiconductor). В КМОП используются "полевые транзисторы с изолированным затвором". Причина использования полевых транзисторов в том, что у этого типа транзисторов маленький ток потребления и энергия тратится преимущественно в моменты переключения. Микросхемы КМОП появились в 1968 году.

Альтернатива КМОП - микросхемы ТТЛ (Транзистроно-Транзисторная Логика) на биполярных транзисторах и резисторах. Скорость работы и плотность монтажа КМОП намного выше, чем у ТТЛ.

Кремниевая долина

Полупроводниками являются химические элементы и химические соединения. Элементы: германий, кремний, углерод, бор, олово, теллур, селен. Германий и кремний имеют кристаллическую решетку типа алмаза.

Первые транзисторы использовали кристаллы германия, даже была изобретена микросхема (интегральная схема) на основе германия в компании Texas Instruments, но кремний оказался наиболее удобным для создания микросхем, так как его диоксид является отличным

диэлектриком и кремний механически прочен в более широком диапазоне температур, чем германий.

Для создания полупроводниковых элементов также подходят соединения химических элементов: арсенид галлия, карбид кремния, нитрид галлия (GaN). Нитрид галлия получил популярность относительно недавно, он нашел себя в цепях питания электронной техники. Схема блоков питания упростилась, они стали мощнее при небольшом размере.

В 1959 году компания Fairchaild Semiconductor создала "планарную" (плоскую, поверхностную) технологию создания микросхем. На плоскую пластину кремниевого кристалла наносят проводящие и непроводящие (диэлектрические) слои. До 1965 года, компания являлась лидером в индустрии производства полупроводников, но из-за неадекватного руководства из компании стали увольняться инженеры. Инженеры создали большое число технологических компаний в пригородах Сан-Франциско в штате Калифорния. Пригороды, где располагались новые компании, стали называться "Кремниевая долина". Два инженера - Роберт Нойс и Гордон Мур уволились из Fairchaild semiconductor в 1968 году и основали компанию Intel.

Гарвардская архитектура компьютеров

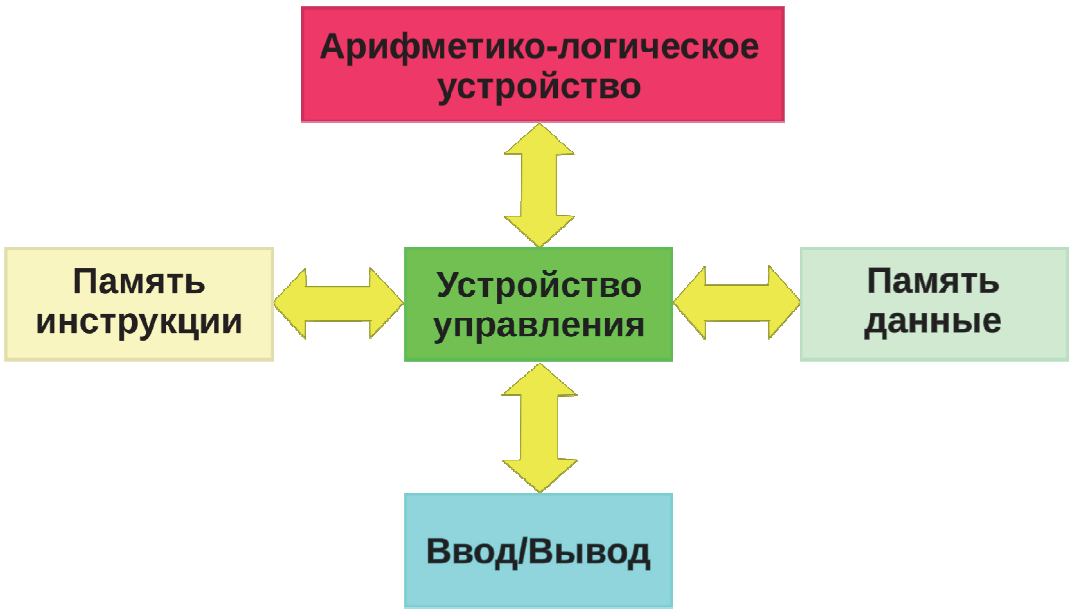
Архитектура компьютера - это модель компьютера, состоящая из частей, описание взаимодействия частей между собой и внешним миром, принципы проектирования и развития модели. В архитектуру компьютера входит описание форматов, типов команд, параметров и усовершенствований. Термин "архитектура компьютера" появился в 1959 году вместо термина "машинная организация".

Раз уж появились инь-янь, ноль и единица, транзистры с `n` и `p` каналами, то и в истории компьютеров было создано две архитектуры: гарвардская и принстонская.

Компьютер Mark I использовал "гарвардскую" архитектуру. В Гарвардской архитектуре:

1) память, где хранится программный код (последовательности команд) и память, где хранятся данные, которые обрабатываются этим кодом, физически разделены. Сегрегация и дискриминация: данные не могут стать кодом, а код не может порождать новый код;

2) шина, по которой код передаётся на выполнение, и шина данных физически отделены друг от друга. На картинке видно, что код (инструкции) и данные поступают в процессор (а именно, его часть: управляющее устройство) по разным шинам (каналам):



Преимущество гарвардской архитектуры в том, что имеется две шины: для данных и для программного кода. Данные и код могут передаваться в два раза быстрее, чем при использовании одной шины.

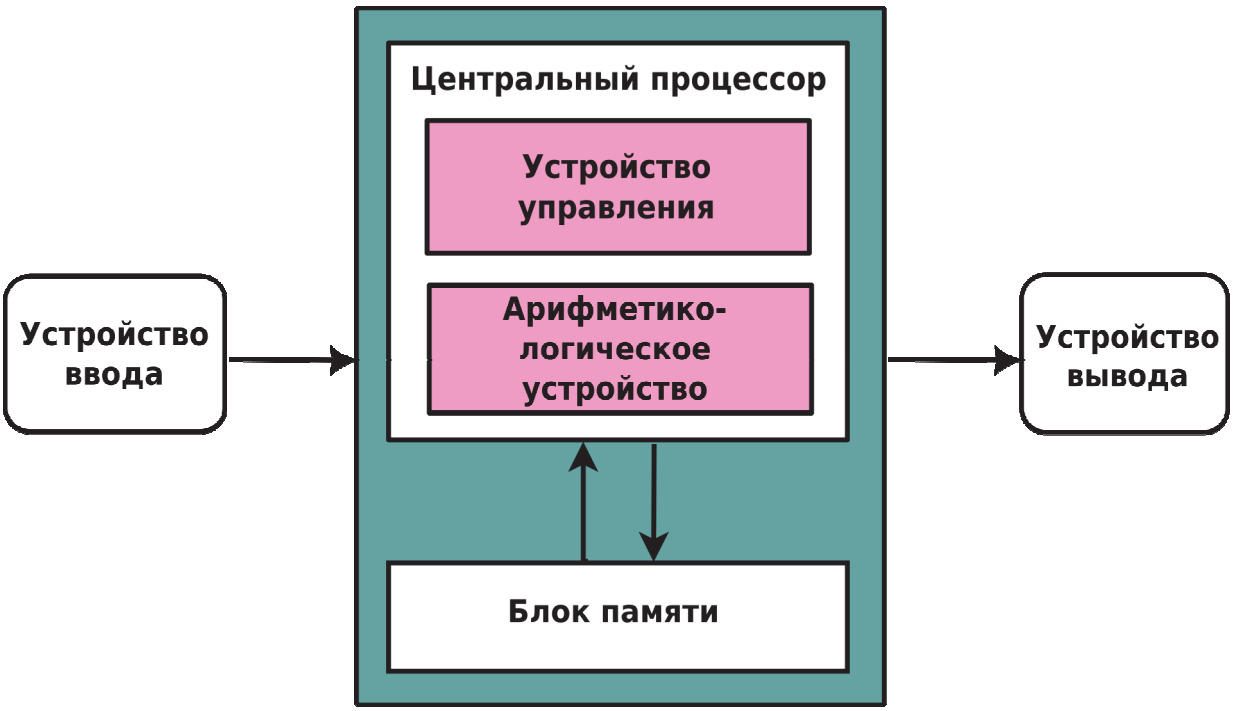
Однако, разделение кода и обрабатываемых им данных усложняет написание программ. Также программа не может создать программный код и передать ему управление. Например, в гарвардской архитектуре нельзя запустить операционную систему, в операционной системе создать программу, скомпилировать созданную программу и выполнить.

В гарвардской архитектуре разделение данных и программного кода позволяет добиться лучшей "безопасности" - данные не могут стать кодом, но максимальная "безопасность" - это отсутствие компьютера: "нет компьютера - нет проблемы".

В компьютере Mark I для инструкций использовалась перфорированная лента, а для работы с данными - электромеханические "регистры". Регистр - память для хранения числа.

Архитектура фон Неймана

В архитектуре фон Неймана ("Принстонская" архитектура) программы и данные хранятся совместно в общей памяти. Это позволяет производить над командами те же операции, что и над данными. Например, создать программный код и передать ему управление.



Узким местом в архитектуре фон Неймана является канал (шина) доступа к памяти. Для ускорения приходится использовать кэши, что усложняет создание процессоров. Этот недостаток перекрывается простотой создания программ и их функциональностью. Современные процессоры используют архитектуру фон Неймана.

Гарвардская архитектура нашла себя в контроллерах (процессорах, специализирующихся для решения одной задачи), обрабатывающих поток звуковых данных (Digital Sound Processor, DSP), в которых важно, чтобы не было задержек, звук не плавал во времени и не требовалось бы сложных решений по синхронизации звуковых данных с временем. Звуковые данные идут потоком с одинаковой скоростью по каналу данных, а поток команд идёт по другому каналу из той части памяти контроллера, куда загружен микрокод обработки звуковых данных.

Гарвардская архитектура также используется при работе с кэш-памятью первого уровня в современных процессорах. У таких процессоров кэш для команд и для данных разный. Для работы с кэшем первого уровня разработчик процессора, обычно, создаёт "микрокод" - управляющую программу, которую можно было бы при необходимости поменять. Необходимость возникает, когда обнаруживаются:

1) ошибки в работе микрокода;

2) нестабильность работы процессора под нагрузкой;
3) высокое тепловыделение;
4) что можно улучшить логику работы микрокода или добавить новые команды.

Если при обнаружении ошибки программу было нельзя менять, то пришлось бы менять бракованные процессоры на новые.

В 1994 году в процессорах Intel Pentium была обнаружена ошибка. При делении двух чисел с плавающей запятой результат иногда был неправильный. Команда процессора называлась FDIV (float divide), поэтому ошибку назвали "Ошибка Pentium FDIV". В справочной таблице процессора, используемой при операции деления, было ошибочное значение. Таблицу нельзя было заменить, так как в процессоре не использовался микрокод. Intel обнаружила проблему, когда процессоры уже продавались и умолчала о ней, так как считала, что проблема существенна только для научных вычислений, а программы, которые используют обычные пользователи редко делят числа с плавающей запятой. Покупатели, желающие заменить процессор, должны были обратиться в компанию и доказать, что ошибка для них существенна. Желание скрыть проблему и неуважение по отношению к покупателям вызвали недовольство и негативно повлияли на репутацию Intel. Intel исправилась и заявила, что будет свободно обменивать процессоры. Энди Грув, руководитель и один из основателей Intel, публично принес извинения за недостойное поведение. Ошибка стоила Intel 475 миллионов долларов, более половины прибыли в четвертом квартале 1994 года.

В 2022 году при производстве процессора Raptore Lake была использована загрязнённая медь. Медные отверстия в контактных площадках под кристаллом процессора окислялись, процессор начинал сбоить и через какое-то время выходил из строя. Intel обнаружила проблему, но не сообщила о проблеме общественности, даже когда с проблемой стало сталкиваться всё больше пользователей процессоров. В первоначальной реакции на проблему, 29 апреля 2024 года, Intel винила микрокод, якобы позволявший увеличивать напряжение питания процессора и перекладывала ответственность на производителей материнских плат, позволявших увеличивать напряжение питания процессора. 22 июля 2024 года Intel решилась признать брак в производстве процессоров, но заявление всё также было витиеватым (вводящим в заблуждение): "Мы можем подтвердить, что производственная проблема с окислением затрагивала некоторые ранние процессоры Intel Core 13-го поколения для настольных ПК. Тем не менее, проблема была устранена и решена с помощью производственных улучшений в 2023 году. Также мы изучили отчёты о нестабильности настольных процессоров Intel Core 13-го поколения, и в результате проведенного анализа выяснилось, что лишь небольшое количество отчетов о нестабильности может быть связано с производственной проблемой. Для проблемы нестабильности мы поставляем исправление микрокода, которое устраняет воздействие повышенного напряжения, являющееся ключевым элементом проблемы нестабильности". За несколько дней акции Intel потеряли треть стоимости.

Кэш-память используется для ускорения обработки данных оперативной (основной) памяти. Её объем небольшой, частота ее работы соответствует тактовой частоте процессора. В неё автоматически кодом микропрограммы загружаются данные из оперативной памяти. Скорость работы с оперативной памятью на порядки (в десятки или до сотни раз) медленнее, чем с памятью кэша. Для хранения бита используется 6 транзисторов из-за чего кэш-память более дорогая и занимает больше места, чем основная память. В основной памяти для хранения одного бита используется конденсатор и транзистор.

Микрокод для DSP и процессоров "прошивается" (flashing, загружается) во флэш-память (память постоянного хранения), физически отделённую от основной памяти процессора.

В биологии ДНК используется и для хранения данных и для кодирования. Можно сказать, что в клетках используется не Гарвардская, а Принстонская архитектура. фон Нейман разработал концепцию клеточного автомата - самовоспроизводящейся машины.

Машинное слово

Машинное слово - фрагмент данных фиксированного размера, обрабатываемый аппаратно как единое целое командами процессора. Например: команда сложения числа, находящегося в ячейке памяти `a` , с числом из ячейки `b` : `ADD a, b`. Вычислительная машина обрабатывает числа, при обработке используется двоичное представление чисел, поэтому алфавит у компьютеров несложный: "машинных букв" всего две: 0 и 1.

По аналогии с человеческим языком буква - то, что человек воспринимает как неделимое (атомарное, дискретное). Например, буква "А" на части не делится. В человеческих языках длина слов имеет произвольный размер.

В 1948 году Шеннон впервые использовал слово bit (binary digit) для обозначения наименьшей единицы количества информации в своей статье "Математическая теория связи".

Один бит - символ, который может принимать одно из двух значений: да или нет, истинно или ложно, включено или выключено. В двоичной системе счисления бит это 1 (единица) или 0 (ноль).

Словом "информация" Шеннон обозначает данные. В человеческой речи информация, информированность это что-то имеющее смысл. Например, данными является число: 299792458. Само по себе число не имеет смысла, просто набор цифр. Информация - это то, что число 299792458 является скоростью света в вакууме, измеренной в метрах в секунду. Информация - это то, что человек извлекает из данных.

Сначала компьютеры использовали только для обработки целых и вещественных (с плавающей запятой) чисел и называли их вычислительными машинами. Вещественные числа использовались для научных расчетов: логарифмов, расчета траектории снарядов, вычислений ядерной физики. Позже компьютерам нашли применение в экономике и стали использоваться для хранения и обработки не только чисел, но и букв. В английском алфавите 26 букв. В письменной речи, кроме букв, используются десятичные цифры (их десять штук `0123456789`) и знаки препинания (пунктуации), например: "`!?,.()`".

Посчитаем, сколько знаков получится: 26 букв плюс 10 цифр плюс штук 10-20 знаков препинания. В сумме получается 46-56 знаков. Для хранения 56 знаков достаточно 6 бит. Поэтому на заре компьютерной эры (в 1950-1960-х годах) длина "машинного слова" была равна 6 бит. 6-битная кодировка использовалась потому, что для представления всех цифр и букв английского алфавита, достаточно было 6 бит: комбинации битов позволяли закодировать 32 символа в одном регистре, 10 цифр и знаки препинания.

Пример: двумя битами можно закодировать четыре символа: `00, 01, 10, 11`. Тремя битами восемь символов: `000, 001, 010, 011, 100, 101, 110, 111`. Если продолжить, то можно обнаружить, что количество символов, которые можно закодировать числом байт, равно двойке возведённой в степень этого числа. В приведенном примере: два в степени два равно четыре, два в степени три (2*2*2) равно восемь.

В микропроцессоре Intel 4004, вышедшем в 1971 году, машинное слово было 4 бита, но этот микропроцессор был предназначен для калькуляторов и обработки чисел.

Позднее, для длины "машинного слова" стали использовать только степени восьмёрки: 8, 16, 32, 64 бита. Причина в том, что для чипа на кристалле без разницы иметь 6 линий или 8 линий проводника, это лишь немного увеличит площадь и тепловыделение кристалла микропроцессора. Разница же в числе бит в машинном слове существенно влияет на написание программ. С появлением микропроцессоров трудоёмкость написания программ стала играть большую роль, чем подешевевшее железо. В целях упрощения также унифицировали и адресацию памяти, она стала 8, 16, 32, 64-битной. Перенос программ на новые версии процессоров проста, если длина машинного слова удваивается. При появлении 16, 32-битных процессоров архитектуры x86 даже не требовалось перекомпилировать программы, написанные для 8-битных процессоров.

8 бит назвали словом байт (byte). Степени восьмёрки: 8, 16, 32, 64 бита нашли отражение в языках программирования. Во многих языках программирования есть типы данных для целых

чисел: byte размером 8 бит, short размером 16 бит, integer размером 32 бит, long размером 64 бита.

Слово "байт" было впервые использовано в июне 1956 года при проектировании транзисторного компьютера IBM 7030 для обозначения порции одновременно передаваемых битов по проводникам. Число проводников варьировалось от 1 до 6. Позже в том же проекте байт был расширен до восьми битов. Слово "байт" (byte) было выбрано как искажённое слово bite (переводится как порция), произносящееся так же. Замена буквы "i" на "y" понадобилась, чтобы не путать новое слово со словом bit.

Побайтовую адресацию памяти впервые использовали в линейке компьютеров IBM System/360. В более ранних компьютерах адресовать можно было только целиком машинное слово, состоявшее из 36 (IBM 701), 18 (PDP-1), 48 (CDC 1604) бит, что затрудняло обработку текстовых данных.

Пример работы компьютера архитектуры фон Неймана

В 1953 году был создан один из первых компьютеров "Стрела". В нём использовалась простая и понятная "трёхадресная система команд" с фиксированной длиной машинного слова. Это позволяло относительно легко писать машинный код и обучаться его написанию.

Память у Стрелы состояла из 2048 ячеек, каждая из которых хранила 43 бита.

Каждая ячейка памяти, в соответствии с архитектурой фон Неймана, хранила либо 43-битное число, либо команду длиной тоже 43 бита. То есть, машинное слово имело размер 43 бита.

Команда состояла из пяти частей. Пример команды, записанной в десятичной форме:

2045 2046 2047 0 **01**

Эта команда означает: число из ячейки с порядковым номером 2045 сложить (код команды **01**) с числом из ячейки №2046 и результат поместить в ячейку №2047. Каждая команда в этой машине имеет три операнда.

Для хранения адресов ячеек памяти использовались 12 бит: число ячеек 2028 равно двойке в степени одиннадцать, ещё один бит использовался про запас, чтобы память могла быть расширена до 4096 ячеек.

Три адреса ячеек, которые использовались в каждой команде, занимали 12*3=36 бит. Ещё остаётся 6 бит на команды и "контрольный знак". "Контрольный знак" занимал один бит и принимал значение 0 или 1. "Контрольный знак" использовался для отладки программ. Для отладки на панели компьютера "Стрела" оператор включал тумблер (переключатель) и выполнение программы приостанавливалось после выполнения команд программы, у которых "контрольный знак" был равен единице. Оператор окидывал взором панель "Стрелы", состоящую из лампочек и мог посмотреть с помощью лампочек содержимое ячеек памяти. Оператор мог продолжить выполнение программы до следующей остановки.

Под номер команды из 43 битов отводилось 6 бит. В стреле использовалось 44 команды. 6 бит позволяют использовать до 64 команд (двойка в шестой степени).

В Стреле была команда №20 условного перехода (аналог оператора GOTO или jump). Пример:

0031 0032 0000 0 **20**

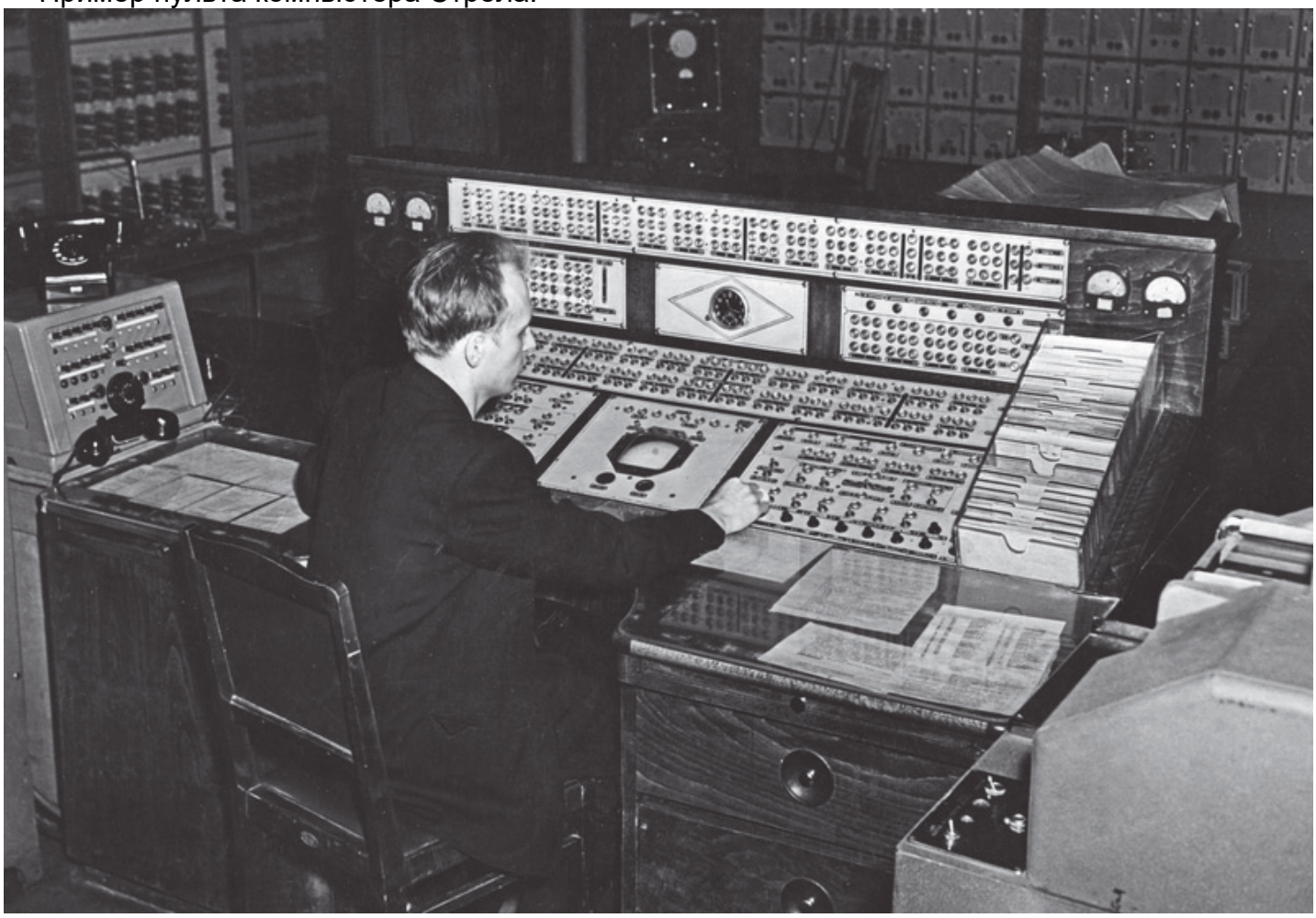
В арифметическом устройстве Стрелы вырабатывался сигнал (из двух состояний 0 или 1). Этот сигнал принимал значение 1, если результат арифметической операции был отрицательным числом. Команда №**20** проверяла этот сигнал, оставшийся от предыдущей команды, если он был нулевым, то передавала управление на первый адрес команды (в примере №0031), если единичкой, то на второй (№0032), третий адрес в команде №**20** не использовался.

Программа, состоящих из **пронумерованных** строк по пять чисел вида:

1) 2045 2046 2047 0 **01**
2) 0031 0032 0000 0 **20**

это пример программы в машинных кодах.

Пример пульта компьютера Стрела:



Данные и программы вводились в "Стрелу" с перфокарт, выводились на перфокарты или печатались на бумажной ленте. На одной перфокарте помещалось двенадцать 43-битных чисел. На одной перфокарте помещалось 12 машинных слов. Скорость работы "Стрелы" - 2000 команд в секунду.

В компьютерах, где используется разделение на регистры и оперативную память, команды принимают один-два операнда, а не три, как в Стреле. При меньшем числе операндов программировать в машинных кодах сложнее. Неудобство программирования в машинных кодах в том, что если нужно вставить команду в середину программы, то адреса всех команд ниже вставленной поменяются (сместятся). Придётся перепроверять все команды перехода, которые указывали на старые адреса и менять адреса на новые. При использовании ассемблеров этой проблемы нет: в процессе трансляции в машинный код из ассемблерного, адреса памяти и регистры назначаются автоматически.

Ассемблер

Ассемблер - это транслятор (преобразователь) программы из текста на языке ассемблера в машинные коды. Вместо машинных кодов в ассемблерах используют понятные человеку обозначения команд (например, `add` вместо `01`, `jump` вместо `20`) и операции, которые нужно выполнить. В ассемблерах можно использовать названия переменных и помечать строки кода, Ассемблеры позволяют не указывать адрес ячейки памяти, а использовать название переменной и не указывать ячейку памяти с командой, а использовать метки. При трансляции метки заменяются на адреса ячеек памяти, которые автоматически назначаются, исходя из числа свободных ячеек.

Ассемблеры упростили написание программ по сравнению с написанием программ в машинных кодах.

Эмуляторы ретро-компьютеров

На веб-странице http://tpmail.ru/dstef/m20/index_ru.html есть эмулятор (программа, имитирующая работу) компьютера М-20 1959 года с трёхадресной системой команд, как и в Стреле. У М-20 машинное слово длиной 39 бит, из них код операции 6 бит и три указателя на ячейки памяти по 11 бит каждый, что позволяет адресовать 2048 ячеек памяти. Этот эмулятор может работать в Windows.

На веб-сайте http://www.besm6.org есть документация и эмулятор (программа, имитирующая работу) компьютера БЭСМ-6 (компьютер был создан в 1967 году):

БЭСМ-6 использовала одноадресные команды с регистром и написание кодов более сложное. Трёхадресные команды более просты и лучше подходят для обучения. Преимущества эмулятора: есть приборная панель, на которой показываются значения в Буферных Регистрах Записи (БРЗ), операционная система ДИСПАК с наборами программ, ассемблеры (в документации к советским компьютерам ассемблер называют словом "автокод").

Трёхадресные команды используются в виртуальной "Учебной трёхадресной машине УМ-3" для изучения программирования на первом курсе факультета Вычислительной Математики и Кибернетики МГУ: http://cmcmsu.info/1course/um3.command.set.htm

На страницах эмуляторов есть инструкции как скачать, скомпилировать, запустить эмулятор, но нет инструкции, что делать дальше. Например, нет инструкции, как запустить простую игру, как после игры написать и запустить простую программу. Также есть технические недоделки. Например, в эмуляторе БЭСМ-6 используется раскладка клавиатуры, отличающаяся от раскладки современных клавиатур и набирать команды затруднительно. В УМ-3 мнемонические (сокращённые) названия команд СЛЦ, ВЧЦ, УЦЧ трудны для восприятия. Более понятными и похожими на названия команд и функций современных языков программирования были бы сокращения SUM, SUB, MUL.

На веб-сайте http://simh.trailing-edge.com/ проекта Computer Simulation and History можно найти эмуляторы других компьютеров, в том числе DEC PDP и VAX. Недостаток в том, что по собранным на сайте компьютерам мало документации и она на английском языке, что делает использование эмуляторов для этих компьютеров неудобными для учёбы.

Эргономика

В 1940 году, на выставке в Нью-Йорке, демонстрировалась игровая вычислительная машина на электронных реле "Nimatron". Это не был компьютер общего назначения, Nimatron мог выполнять одну программу - игру в "ним".

Хотя это был первый в истории игровой компьютер, ни игра, ни он не оказали влияние на развитие электронных игр и компьютеров. Примечательным было то, что впервые в истории намеренно замедлили работу компьютера. Компьютер рассчитывал свой ход за долю секунды, что обескураживало игроков, которые обдумывали каждый свой ход. Чтобы игроки не чувствовали ущербности, было добавлено замедление при выдаче результата хода, который делал компьютер.

Эргономика изучает, как приспособить должностные обязанности, рабочие места, предметы и объекты труда, компьютерные программы для безопасного и эффективного использования, исходя из физических и психических особенностей человека. Для этого

исследуются действия человека в процессе работы, скорость освоения новой техники, затраты умственной, психологической, физической энергии человека, производительность и интенсивность работы.

Если учебный курс, по которому вы изучаете предмет, не понятен, возможно, учебный курс неэргономичен, то есть не удобен для восприятия.

Nimatron разработал Эдвард Кондом, ставший в 1945 году директором Национального института стандартов и технологий (NIST),а в 1946 году президентом Американского физического общества.

Вторым игровым компьютером в истории был Nimrod, созданный в 1951 году на основе клона компьютера Mark I. Компьютер использовали на промышленных выставках для популяризации компьютеров. В компьютере использовалось 350 ламп для расчетов, 130 запасных ламп, 120 реле и несколько германиевых диодов. Компьютер потреблял 6 киловатт энергии.

Nimrod - имя царя в древнем Вавилоне, который хотел сжечь Авраама, но Авраам не сгорел, чем расстроил Нимрода.

*Если нагрузить микропроцессор вычислениями, то он начнет выделять много тепла и может даже сгореть. Чтобы этого не произошло, используют замедление частоты, с которой процессор обрабатывает (принимает, вычисляет, возвращает) данные. Компьютер Nimatron замедлял возврат данных. Если человека сильно нагрузить задачами, то он "сгорит" на работе. Чтобы сделать паузу, человек берёт отпуск. От процессора можно отводить тепло, использовать охлаждение (воздушное, жидкостное). Для уменьшения информационной составляющей, в человеческой речи и письме можно "лить воду" - разбавлять информацию лишними словами. Примером "литья воды" является этот абзац, так как в этом абзаце, в отличие от предыдущего текста, информации почти нет. Тем не менее, "вода" может быть полезна: фраза "*было добавлено замедление*" даёт полезную идею: если вы устали читать текст, то можно сделать паузу и не торопиться.*

Микропроцессоры

15 ноября 1971 появился микропроцессор Intel 4004.

Микропроцессор - это микросхема на кристалле кремния, в которой помещается всё, что относится к логике процессора: вычислительное и логическое устройство, управляющее устройство, регистры, таймер, устройства управления памятью, а также есть возможность выполнения на нём произвольных программ.

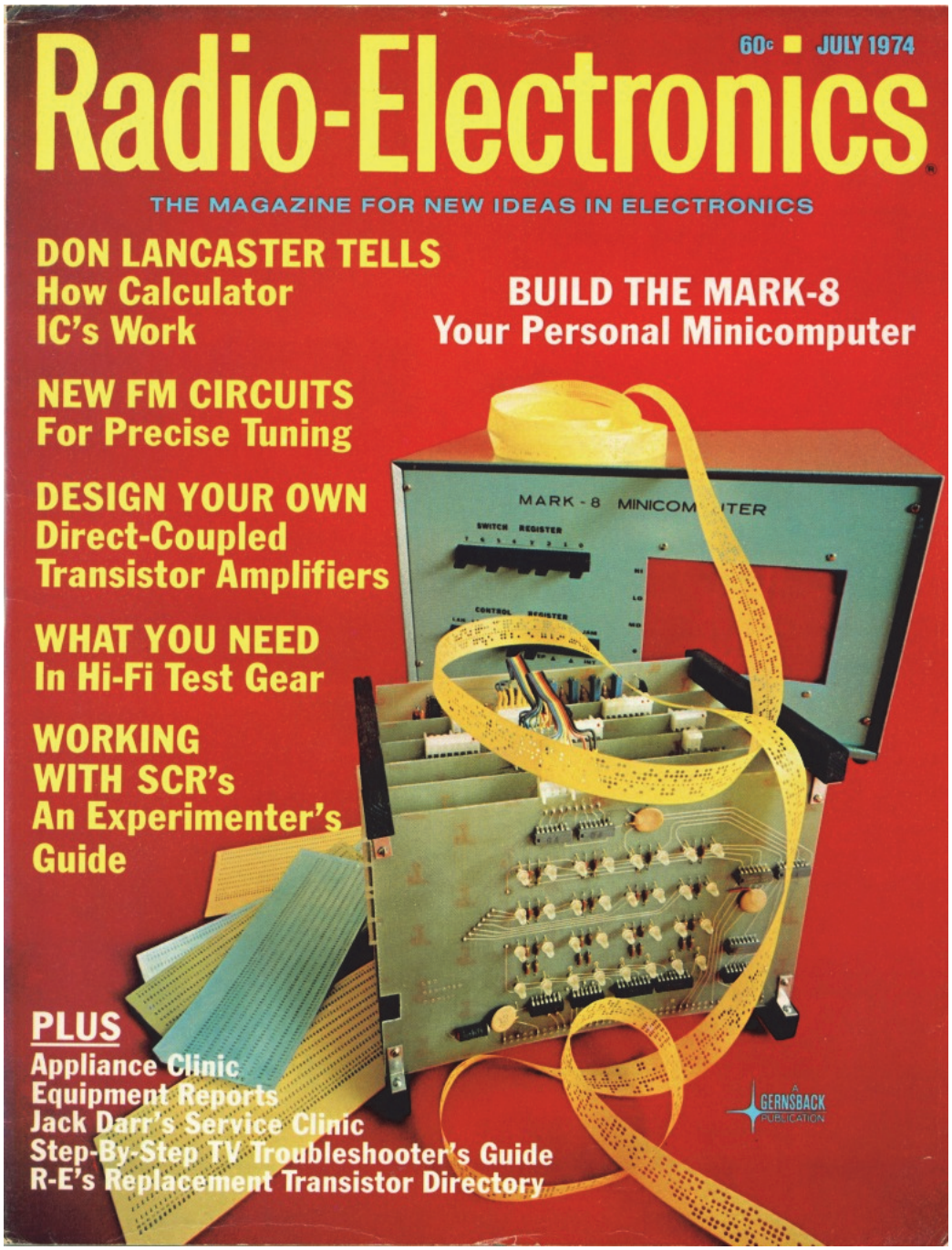
В микропроцессоре Intel 4004 использовалась Гарвардская архитектура, то есть раздельное хранение программ и данных, из-за чего процессор был не удобен для создания программ. Разрядность регистров микропроцессора и внутренняя шина передачи данных были 4 бита. Микропроцессор был разработан по заказу японской компании Nippon Calculating Machine для их калькулятора и состоял из 2300 транзисторов.

1 апреля 1972 года появился процессор Intel 8008 на архитектуре фон Неймана (с общей памятью для кода и данных) на 3098 транзисторах, с 8-битной шиной, 14-битной адресацией памяти, благодаря которой мог работать с 16 килобайтами памяти. Процессор 8008 не является развитием процессора 4004, у них разная архитектура и набор команд.

---

*В июле 1974 года, в журнале Radio-Electronics, была опубликована статья с рекламой компьютера на процессоре Intel 8008, который назвали "персональный **мини**компьютер Mark-8". Разработчик миникомпьютера обращался в журнал Popular Electronic, но этот журнал не стал публиковать статью. В статье предлагалось купить за 5$ схему сборки компьютера, за 50$ монтажные платы. Детали для компьютера предлагалось разыскивать и покупать самостоятельно. Персональный миникомпьютер успеха не имел.*

По традиции, первооткрыватели не получают прибыли, но скорее всего основная причина была в том, что радиолюбители не хотели тратить время на поиск радиодеталей для сборки. Люди любят дорабатывать и создавать что-то самостоятельно, но только, если это не слишком сложно.



обложка журнала Radio-Electronics за июль 1974 года

В 1974 году появился микропроцессор Intel 8080A. В микропроцессоре было 4758 транзисторов, 16 команд передачи данных, 31 команда обработки данных, 28 команд для перехода на другую команду, 5 команд управления процессором. Для работы с памятью в процессоре использовалась 16-битная шина, поэтому процессор мог работать с 64Кб памяти. Использовалась архитектура фон Неймана, то есть память не разделялась на память для программ и данных. В розницу процессор стоил 360 долларов. Процессор Intel 8080A был значительно улучшенной версией Intel 8008. В Intel 8080A был существенно расширен набор команд.

Процессор Intel 8080A использовался в микрокомпьютере Altair 8800, который появился в 1975 году. Журнал Popular Electronics опубликовал статью об Altair 8800. Компьютер продавался в форме конструктора за 439 доллара или в собранном виде за 621 доллар.

HOW TO "READ" FM TUNER SPECIFICATIONS

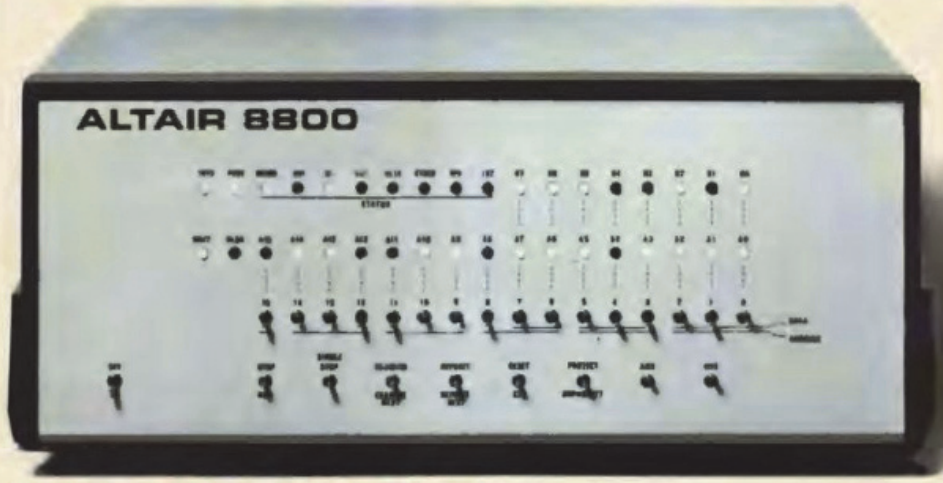
Popular Electronics

WORLD'S LARGEST-SELLING ELECTRONICS MAGAZINE JANUARY 1975/75¢

PROJECT BREAKTHROUGH!

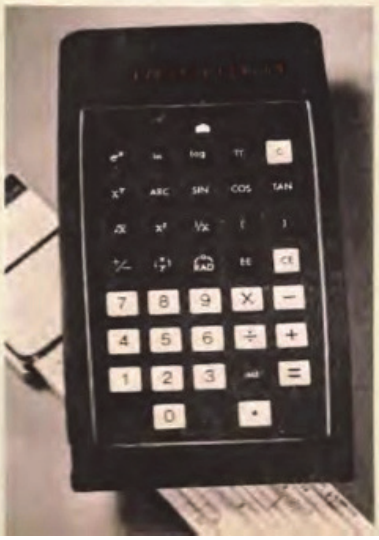
World's First Minicomputer Kit to Rival Commercial Models...

"ALTAIR 8800" SAVE OVER $1000



ALSO IN THIS ISSUE:

- An Under-$90 Scientific Calculator Project
- CCD's—TV Camera Tube Successor?
- Thyristor-Controlled Photoflashers



18101

TEST REPORTS:

Technics 200 Speaker System
Pioneer RT-1011 Open-Reel Recorder
Tram Diamond-40 CB AM Transceiver
Edmund Scientific "Kirlian" Photo Kit
Hewlett-Packard 5381 Frequency Counter

обложка журнала Popular Electronics за январь 1975 года

Спрос на Altair 8800 превысил прогнозы. За первый месяц с начала продаж было продано несколько тысяч наборов конструктора. Примечательно то, что подписчики журнала - радиолюбители, предпочитали приобретать именно конструктор, то есть одной из целей было собрать компьютер своими руками. У компьютера не было ни клавиатуры, ни дисплея, поэтому, использовать Altair 8800 для решения практических задач было затруднительно.

Популяризация компьютера через журнал для радиолюбителей, выгодная цена по сравнению с розничной ценой на процессор, удачная системная шина S-100, открытость архитектуры, сделала компьютер популярным. Радиолюбителями были созданы платы расширения, превратившие его в полноценный компьютер с периферией. Радиолюбители занимались любимым делом - конструированием и доработкой. Но самое главное, для этого компьютера стали писаться программы. Пол Аллен и Билл Гейтс написали интерпретатор языка BASIC для Altair 8800.

В СССР выпускался клон процессора Intel 8080A под названием КР580ВМ80А.

В журнале "Радио" в 1986 году публиковались схемы для самостоятельной сборки микрокомпьютера "Радио 86РК". Через какое-то время стали выпускать наборы для сборки компьютера Радио-86РК. Наборы назывались Электроника КР-01... КР-04 и продавались за 395 рублей. В 1987 стал выпускаться клон "Радио-86РК", называвшийся "Микроша", он продавался за 500 рублей:

Ранее в журнале "Радио" (на протяжении 1982-1983 годов) публиковались схемы компьютера "Микро-80", но в нём было примерно 200 деталей. Как и с компьютером Mark-8, детали предлагалось приобретать самостоятельно, поэтому "Микро-80" не получил распространения.

IBM PC

8 июня 1978 года появился микропроцессор Intel 8086, первый 16-битный микропроцессор компании Intel. Его набор команд стал основой для архитектуры x86. Набор команд был несовместим с набором команд Intel 8080 и программы на ассемблере нужно было переписывать, чтобы они смогли работать на новом процессоре.

В процессоре было 98 команд: 19 команд передачи данных, 38 команд их обработки, 24 команды перехода на другую команду, 17 команд управления процессором. Современные процессоры сохраняют возможность выполнять все эти команды. Процессор мог адресовать 1Мб оперативной памяти с помощью 20-битной адресной шины. Шина данных была 16-битной. Начальная стоимость процессора была 360 долларов. Был выпущен аналог, процессор Intel 8088, который стоил гораздо дешевле - 125 долларов.

IBM решила использовать процессор 8088 в новой линейке компьютеров IBM PC (Personal Computer, Персональный Компьютер), выпущенной в августе 1981 года.

IBM PC привели к успеху:

1) модульность - компьютер состоял из легко заменяемых модулей;

2) открытая архитектура - схемы компьютера не держалось в секрете и продавались за 49 долларов, благодаря чему сторонние компании могли производить периферию и клоны

3) наличие BIOS - программы, дающей стандартный программный интерфейс для доступа к устройствам, расположенным на материнской плате, что упростило разработку программ. Начиная с 1984 года стали появляться BIOS сторонних производителей: компаний Phoenix, American Megatrends, Award Software.

Открытость архитектуры сделала IBM PC стандартом персональных компьютеров, вытеснив конкурентов. В 1987 году IBM попыталась вытеснить сторонних производителей, выпустив серию персональных компьютеров под названием "PS/2" с закрытой архитектурой и шиной MCA, но не смогла конкурировать и, в конечном итоге, перестала самостоятельно производить персональные компьютеры. Из архитектуры PS/2 в архитектуру IBM PC перешел разъем (аппаратный "порт") клавиатуры и мыши, который стали называть PS/2-порт, а также дискеты уменьшенного размера 3 дюйма, вместо 5 дюймов.

1 февраля 1982 года был выпущен процессор 80286, который был полностью совместим по командам с процессором 8086. Шина адресации памяти была увеличена до 24 бит, поэтому процессор мог работать с 16Мб оперативной памяти.

В 1985 году был выпущен процессор 80386, первый 32-битный процессор семейства процессоров x86. Регистры процессора были расширены до 32 бит (размер машинного слова стал 32 бита) и упростилась адресация памяти. Процессор мог адресовать 4Гб оперативной памяти.

В 1989 году был выпущен процессор 80486, в 1993 году процессор, который получил самостоятельное имя: Pentium.

Little Endian и Big Endian

Данные измеряются и хранятся в байтах. Один байт состоит из 8 бит и может закодировать 256 (двойка в степени восемь) отличающихся друг от друга значений, что немного. В реальном мире используются числа большей размерности. Для чисел, не вписывающихся в диапазон от 0 до 255, нужно использовать несколько байт.

Числа сохраняются в памяти компьютеров. Память можно представить себе как упорядоченный набор байт: с первого до последнего. Например, десятичное число 255 можно записать в двоичной форме как `11111111`. Десятичное Число 256 требует для хранения два байта и его можно записать в двоичном виде двумя способами:

`00000001 00000000` - от старшего байта к младшему (big-endian, BE), старший байт имеет **меньший** адрес.

`00000000 00000001` - от младшего байта к старшему (litle-endian, LE)

Биты традиционно записываются от старшего разряда к младшему. С записью битов нет неоднозначности, а вот с порядком записи и передачи байтов исторически возникла неоднозначность.

Термины big-endian и little-endian взяты из романа Джонатана Свифта "Путешествия Гулливера". В романе лилипуты разделились на два лагеря - сторонников разбивать яйцо с острого конца (little-end) или с тупого (big-end).

Создатели терминов понимали комичность существования двух вариантов, но альтернативы постоянно появляются - всегда найдётся кто-то с альтернативным взглядом на предмет, даже если понятно, что альтернатива бессмысленна. Доказать бессмысленность можно, убедить нельзя, так как доказанная бессмысленность становится принципом.

Альтернативная точка зрения состоит в том, что альтернативы полезны тем, что делают жизнь нескучной - человек может выбирать.

Хотя запись big-endian кажется более естественной и понятной, little-endian имеет преимущество при обработке данных: у little-endian адрес числа (указатель на первый байт числа) не зависит от длины данных. Например, число 127 хранится в 16-разрядном виде и занимает два байта:

`00000000 01111111` - от старшего байта к младшему (big-endian, BE), старший байт имеет меньший адрес.

`01111111 00000000` - от младшего байта к старшему (litle-endian, LE).

Если число хранится в big-endian формате, то при считывании числа в однобайтный регистр придется пропустить первый байт. Если число хранится в little-endian формате, адрес этого числа не меняется - считывание начинается с первого байта в регистр любой размерности.

Big-endian использовался процессорами IBM/360, Motorola 68000, Sun SPARC.

Little-endian используется в процессорах Intel x86. Процессоры ARM используют little-endian. Архитектура процессоров RISC-V, появившаяся позже архитектуры ARM, использует только litte-endian. little-endian используют почти все современные процессоры, а big-endian процессоры стали историей.

Запись big-endian удобнее для чтения людьми и такой порядок записи байт используется по умолчанию в технической литературе.

Для текстовых файлов в кодировке UTF-16, в которой символы хранятся в двух байтах, первые два байта в начале файла (BOM, Byte Order Mask) определяют little-endian или big-endian формат. Если байты отсутствуют, то стандарт рекомендует считать порядок как big-endian.

Системы команд CISC и RISC

В 1970х годах при поиске возможностей ускорить работу компьютеров, выяснилось, что из всего разнообразия команд, которые мог выполнять процессор, чаще всего выполнялось несколько простых команд: загрузка данных из памяти в регистры процессора, команда перехода, несложные битовые операции и возврат значений в память.

Появилась идея, которую назвали RISC (Reduced Instruction Set Computing, уменьшенный набор команд) - убрать долго выполняющиеся команды и оставить наиболее простые, большинство которых выполнялось бы за один такт процессора. Уменьшение числа команд было целью, а побочным эффектом. Уменьшение времени достигалось унификацией длины команд вместе с операндами (параметрами команд), что позволяло уменьшить число транзисторов для реализации команд.

Параметры команд процессора (инструкций) называют операндами потому, что команды выполняют операции (вычисления) с данными. Пример: 1 + 2 равняется 3. "+" - это операция. Числа 1 и 2 это операнды. У операции "+" два операнда, поэтому оператор называется бинарным (bi - означает два). Есть унарные операции, у которых один операнд. Пример: NOT (отрицание): NOT true равняется false. NOT false равняется true.

Архитектуру команд, которая использовалась до появления RISC, назвали CISC (Complex Instruction Set Computing, сложный набор команд). В CISC - добавляли сложные команды, которые могут быть реализованы логическими элементами в железе, чтобы уменьшить число более простых команд, которыми могла бы быть реализована сложная команда.

Пример сложной команды: сравнить строки. Пример простой команды: вычесть одно машинное слово из другого. Сложные команды могут быть заменены набором простых команд. Реализация сложных команд на уровне железа позволяла экономить оперативную память, которая в то время была дорогой. Для реализации программной задачи меньшее число сложных команд требовали меньшего обмена с памятью. Например, если операция над строкой символов выполняется одной командой, то во время её выполнения, не нужно подгружать из оперативной памяти код с другими командами. Память постепенно дешевела, необходимость в экономии постепенно уменьшалась. Долгое время ускорение работы процессоров шло по пути CISC - аппаратной реализацией новых команд и увеличения их количества и это было оптимально. Для процессоров x86 Intel использовала архитектуру CISC и это было оправданно.

С удешевлением памяти узким местом становились вычислительные мощности процессора. Ускорять вычисления можно было распараллеливанием выполнения команд процессора. Для распараллеливания используют следующие техники.

1) Конвейер (pipeline). Сложные команды состоят из последовательности простых команд. Вместо последовательного выполнения (ожидания завершения конца одной команды и перехода к выполнению следующей команды), команды начинают выполняться параллельно или со сдвигом в один или несколько тактов процессора, если они не мешают друг другу (используют разные части процессора).

2) Суперскалярное выполнение команд. В процессоре делают не один, а несколько функциональных узлов одного вида: арифметические устройства, умножители.

3) Спекулятивное выполнение. Например, нужно проверить условие `a>0` и, если оно верно, выполнить команду `b=b+2`, а если не верно выполнить команду `b=b-2`. Это условие можно записать так:

```
if (a>0) then b=b+2 else b=b-2;
```

При спекулятивном выполнении сложение `b+2`, вычитание `b-2` и вычисление условия `a>0` начинают выполняться одновременно. Как только выражение будет вычислено, берётся один из двух уже вычисленных результатов и присваивается переменной `b`. Если число команд, которые могут выполняться параллельно небольшое, то используется "предсказание переходов", то есть выполняется та команда, которая вероятнее всего будет выполнена. Например, в большинстве случаев условие `a>0` истинно, так как в противном случае программист бы написал `a<=0`.

4) Внеочередное выполнение команд. Если значения операндов для выполнения очередной команды недоступны, может выполняться следующая команда, готовая к выполнению. Чем длиннее конвейер, тем эффективнее внеочередное выполнение команд.

5) Переименование регистров. Если при внеочередном выполнении две команды используют одни и те же регистры, то замена регистров при компиляции позволит выполнить вторую команду одновременно с первой, так как они будут использовать разные регистры. Чем больше регистров общего назначения, тем больше вероятность, что компилятор найдёт свободные регистры и скомпилирует код так, что удастся выполнить вторую команду.

Эти техники ускорения могут совместно использоваться.

Проблема CISC в том, что команды требуют разное время (число тактов) на своё выполнение и у команд может быть разная длина. Из-за этого распараллелить выполнение команд проблематично. Команды одинаковой длины можно загружать потоком из медленной памяти.

У процессора Intel 286 (CISC) было 357 команд и их вариаций, у ARMv1 (RISC) намного меньше - 45 команд и их вариаций. Меньшее число команд у RISC это следствие. Отличия RISC от CISC:

1) у CISC часть команд требует большое число тактов для выполнения. В RISC такие команды постарались убрать;

2) команды CISC могут загружать данные из памяти и возвращать их в память, а в RISC команды работают только с "регистрами" и есть команды: "загрузить данные в регистр из памяти" и "вернуть данные из регистра в память". Регистры - это небольшой участок памяти для хранения числа фиксированной длины (например, 32 бита) в самом процессоре, доступ к регистрам выполняется за один такт. Команды из оперативной памяти попадают в регистр, после чего микропроцессор обрабатывает их;

3) в RISC убрали стеки и оставили только регистры, число которых увеличили. У компиляторов появилась возможность для оптимизации: в какие регистры нужно положить данные для вычислений, чтобы минимизировать обмен данными с памятью.

---

*Стек (stack, стопка) - список, организованный по принципу LIFO (**L**ast **I**n - **F**irst **O**ut, последним пришёл - первым вышел). Стек сравнивают со стопкой бумаг: чтобы взять вторую бумагу сверху, нужно сначала взять первую сверху. В процессорах стеки используются для хранения данных.*

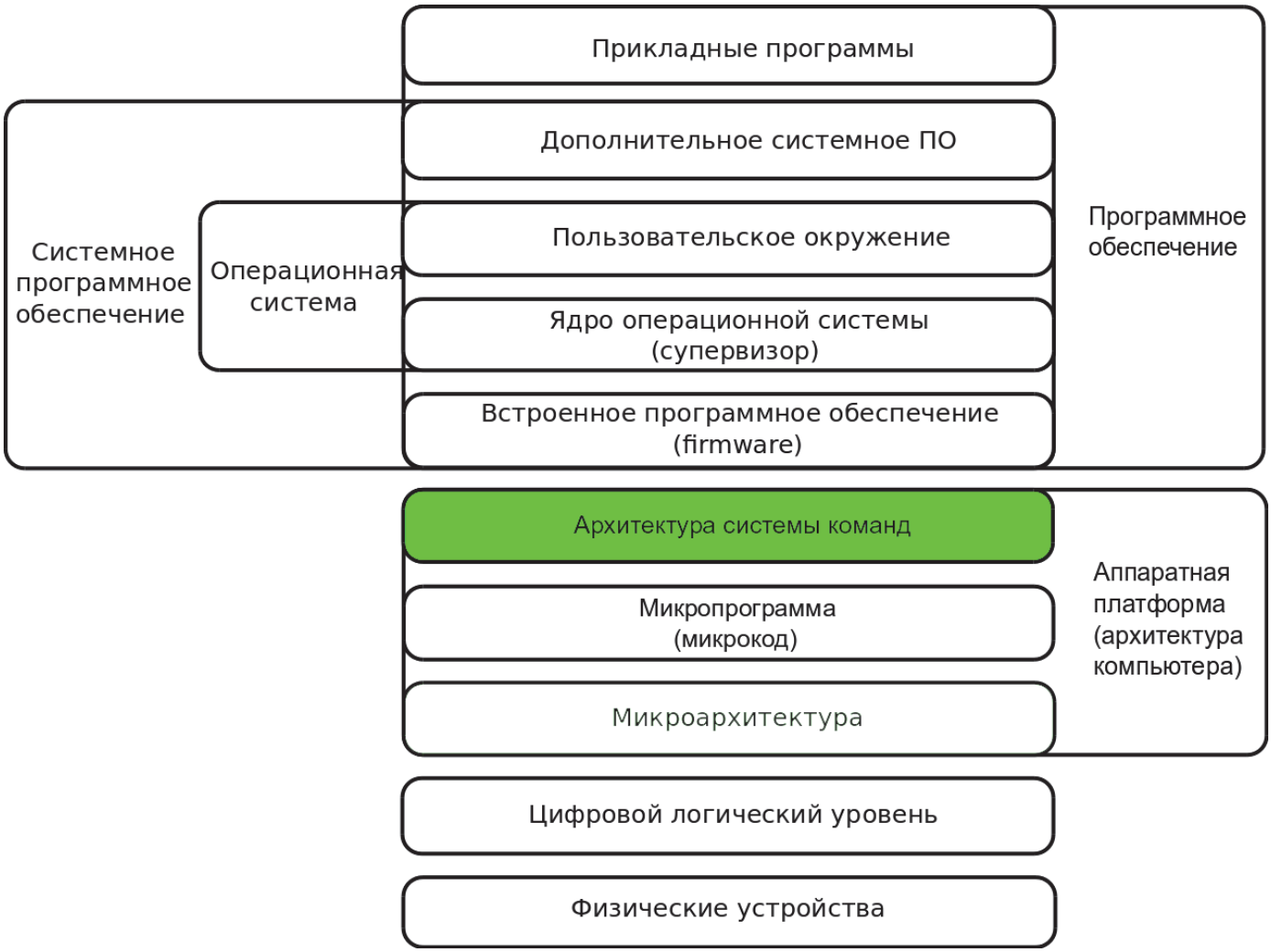
Недостаток RISC в том, что число команд в программах увеличивается в несколько раз по сравнению с CISC, а преимущество - декомпиляция кода и поиск уязвимостей в исполняемых программах сложны для злоумышленников, так как при анализе кода приходится обращать внимание на то, что находилось в общих регистрах до передачи управления из другой функции. Однако это является и недостатком: для реализации многозадачности операционной системе приходится программно сохранять состояние регистров. Сохранять состояние должна вызывающая функция, поскольку только ей известно, какие регистры она будет использовать. Если компилятор не сможет предсказать, какие регистры будут использоваться, то сохраняются все регистры, а это долго.

Использование ассемблера для написания программ для процессоров CISC было возможно и имело смысл. Использование ассемблера для написания кода для RISC возможно, но трудоёмко, так как из-за большого числа команд, которыми приходится реализовывать то, что у CISC делает одна команда, сложно проследить логику программы. Частично это сглаживается макросами ассемблера.

Железо развивалось по пути уменьшения размера элементов на чипах и токопроводящих соединений, что позволяло увеличивать частоту работы процессоров. Увеличение частоты работы процессора позволяет выполнять больше команд за то же время. Скорость тока не бесконечна, длинные соединения ограничивают частоту процессора. Если данные передаются по одновременно нескольким соединениям, то по более длинному соединению сигнал будет идти с задержкой.

Чем больше число транзисторов на единицу объема кристалла чипа и чем меньше сечение соединений между ними, тем больше тепловыделение, а скорость отведения тепла ограничена.

Так как у RISC-процессоров меньше транзисторов, то RISC имеют преимущество. Intel 386 был последним микропроцессором с классической CISC-архитектурой. Процессоры архитектуры x86 стали внутренне использовать архитектуру RISC, но для совместимости с набором команд CISC усложнился аппаратный декодер, который преобразовывал команды CISC в набор более простых команд (микрокод). Декодер содержит много транзисторов, занимает большую площадь на кристалле кремния и работает постоянно, что приводит к большому энергопотреблению процессора даже, когда он простаивает. В процессоре, использующем микрокод, сложные для выполнения команды реализуются в виде набора простых команд. В дальнейшем, для управления декодером стали использовать обновляемый микрокод, чтобы иметь возможность исправлять ошибки проектирования аппаратной архитектуры процессора и даже добавлять новые команды.



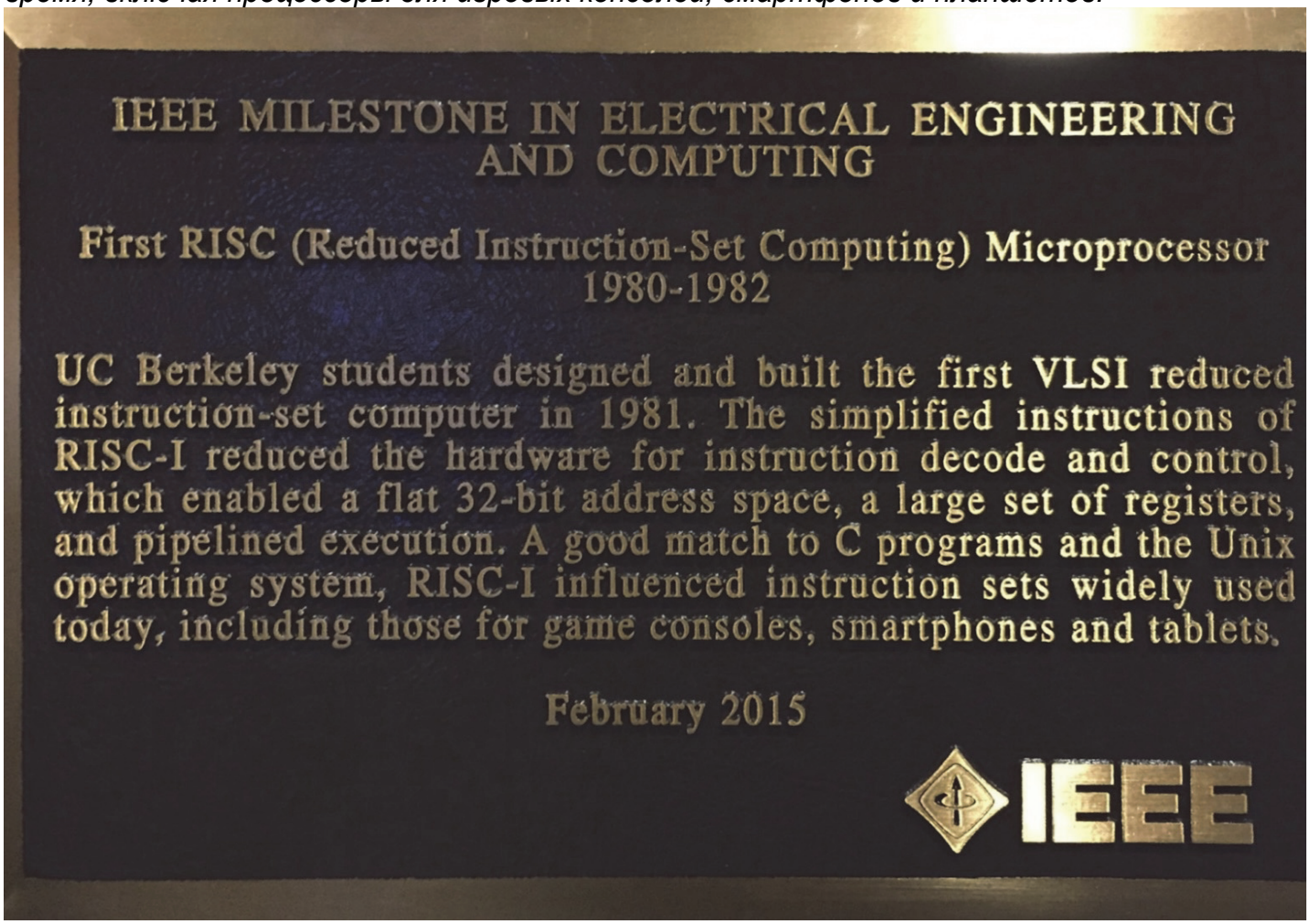
В настоящее время, система команд x86-64 используется преимущественно в компьютерах, а RISC используется в специализированной (встраиваемой) технике, где важно небольшое энергопотребление.

Идея RISC была реализована в процессорах RISC-I, MIPS, SPARC, PowerPC, DEC Alpha. Однако, эти процессоры не имели преимуществ (по производительности, энергопотреблению, стоимости) по сравнению с процессорами архитектуры x86 и остались в прошлом. Только процессоры MIPS продолжают использоваться в микроконтроллерах, где конкурируют с ARM.

Первым процессором, использующим идею RISC, был процессор RISC-I, созданный в 1982 году. В нем было 44420 транзисторов и 32 инструкции. RISC-II, выпущенный в 1983 году имел 40760 транзисторов, 39 инструкций и работал в три раза быстрее RISC-I.

IEEE установила мемориальную доску в Калифорнийском университете в Беркли:

*Студенты университета в Беркли спроектировали и построили первый компьютер с сокращенным набором команд (VLSI) в 1981 году. Упрощенные команды RISC-I уменьшили объем железа, необходимого для декодирования и выполнения команд, что позволило использовать 32-битное адресное пространство, большой набор регистров и конвейерное выполнение команд. Хорошо сочетаясь с программами на языке **C** и операционной системой Unix, RISC-I повлиял на наборы команд процессоров, широко используемых в настоящее время, включая процессоры для игровых консолей, смартфонов и планшетов.*



ARM

В Великобритании BBC (British Broadcasting Corporation, Британская Вещательная Корпорация) заказала разработку компьютера компании Acorn (переводится как жёлудь) для национального телевизионного шоу, посвященного компьютерной грамотности.

Из-за высоких технических требований BBC не отдала заказ компании Sinclair, чей компьютер ZX Spectrum доминировал в Великобритании среди домашних компьютеров. В требования были включены возможность создания программ, графика для проектирования (CAD - Compuer-Aided Design), поддержка звука, работа с периферийными устройствами (принтеры). Домашние компьютеры, такие как ZX Spectrum, ориентировались, в основном, на игры и имели производительность достаточную для игр, но не для требований BBC. В то время игры были простыми, компьютерную графику использовали программы для проектирования, обработки изображений. Обработку графики выполнял процессор и его вычислительные мощности были важны.

К концу 1981 года Acorn разработала компьютер, его назвали BBC Micro. Было выпущено 1,2 миллиона штук, эти компьютеры использовались в английских школах.

Для реализации технических требований Acorn понадобился 16-битный процессор. Рассматривались процессоры Intel 80286, National Semiconductor 32016 и Motorola 68000. Эти процессоры показались медленными, неэффективно использовали оперативную память и оказались сложными для программирования на языках высокого уровня для достижения желаемой производительности. В 1985 году появился процессор ARMv1 (Acorn RISC Machine), разработанный всего за полтора года инженерами компании Acorn. Процессор использовал новый набор команд и архитектуру и не был совместим с существовавшими процессорами. В процессоре было 27 тысяч транзисторов. У аналогичных процессоров Intel 80286 - 134 тысячи, Motorola 68000 - 68 тысяч. В ARMv1 отсутствовали кэш-память, аппаратные устройства для выполнения умножения и деления, вычислений с плавающей запятой, при этом программы выполнялись на порядок быстрее, чем на Intel 80286.

Также у процессора ARMv1 было низкое энергопотребление: 0,1 ватт. У процессоров с аналогичной производительностью, энергопотребление, а значит и нагрев, был в десятки раз больше - несколько ватт. Через много лет, когда появились мобильные устройства, небольшое энергопотребление архитектуры ARM стало основным преимуществом этой архитектуры.

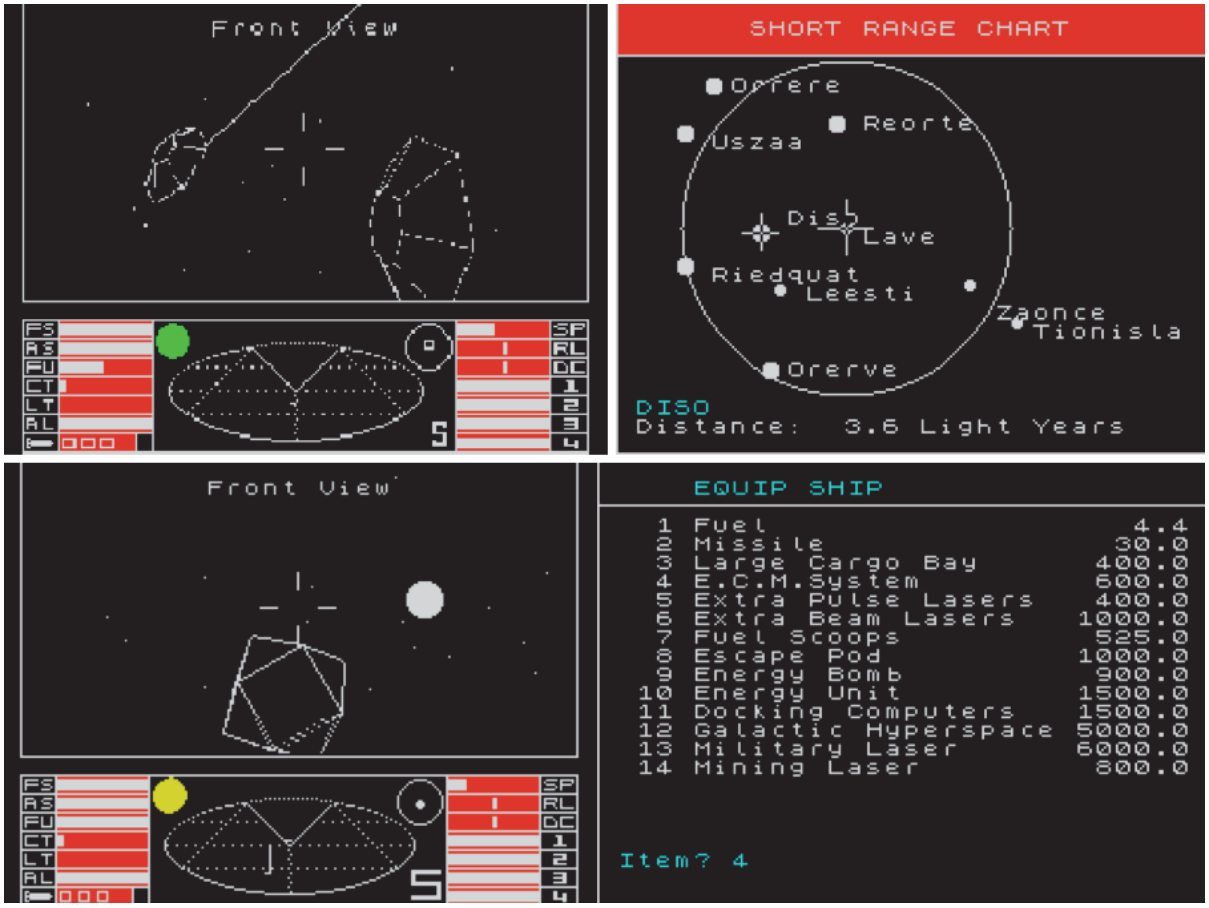
В 1987 году был выпущен процессор ARMv2 и компьютер Acorn Archimedes на его основе. Однако, существовавшие операционные системы и программы не поддерживали этот процессор. Из-за этого компьютер не стал популярным, хотя был довольно производительным. Рынок завоевали IBM-PC и Apple Macintosh, на которых работали программы PageMaker, Word, Excel.

Компания Apple решила выпустить карманный компьютер Newton и искала процессор с низким энергопотреблением. Процессор Acorn ARM лучше всех подходил под требования, а компании Acorn была нужна большая компания для популяризации процессора. В ноябре 1990 года компаниями Apple, Acorn, VLSI (производитель процессоров) была создана компания ARM (Advanced RISC Machines). С того времени все процессоры, используемые в Apple iPhone и iPad, используют архитектуру ARM.

Игры и компьютерная графика

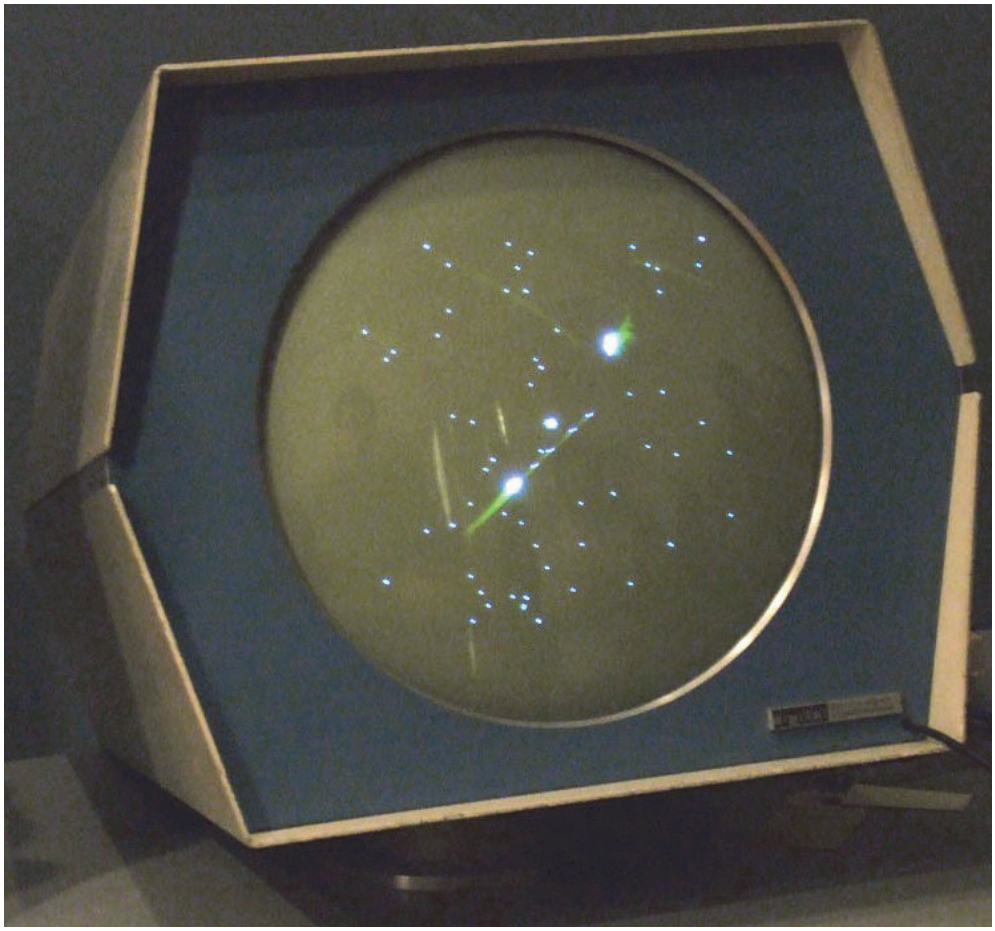
В 1984 году для компьютеров BBC Micro и Acorn Electron, производимых компанией Acorn, была создана компьютерная игра Elite. В игре использовалась каркасная трёхмерная графика. Игра представляла собой космический симулятор. Вселенная Elite состояла из восьми галактик по 256 планет в каждой. Галактики генерировались при запуске игры. Галактики кодировалась небольшим набором цифр, которые использовались как начальное значение (seed) для генератора псевдослучайных чисел, который использовался для наполнения галактик. При последующих запусках сохраненной игры, генератор инициировался тем же набором цифр, генерация галактик проходила те же самые этапы. Вселенная восстанавливалась в точности, как при первом запуске игры. Названия, координаты звёзд оставались теми же. Игра начиналась на орбитальной станции планеты Lave с 100 credits (денежных единиц) и кораблём. Игрок летал на корабле по звездным системам, сражался с другими кораблями, покупал и продавал товары. Заработанные credits тратились на оснащение корабля оружием, защитой, увеличением грузоподъемности. В игре не было цели, игрок исследовал космос и мог достичь ранга Elite, для получения которого нужно было выиграть 6400 сражений. С 1985 по 1988 года игра Elite была портирована на все платформы того времени: ZX Spectrum, Commodore, Apple II, Atari, Amiga, MSX и IBM PC. К 20 сентября 2014 года, 30-летнему юбилею игры, была создана игра Elite:The New Kind, приближенная к оригиналу.

Пример графики игры:



На Front view - каркасы (очертания) кораблей и звёзды в виде точек. На Short range chart - карта звезд и круг, показывающей дальность полета корабля на одной заправке. На Front view - каркас орбитальной станции в виде многогранника и кружок, означающий ближайшую звезду. Мощности процессора хватало, чтобы плавно отрисовывать линии трёхмерного каркаса и менять положение звёзд.

Идея игры Elite схожа с одной из первых компьютерных игр "Spacewar!" (Звёздные войны), которая была создана в 1961 году для компьютера DEC PDP-1 **Стивом Расселом**, сотрудником Массачусетского Технологического Института. Причина создания игры - у компьютера был дисплей, хотелось продемонстрировать возможности компьютера. В игре было реализовано управление траекторией корабля с тягой и вращением, запуска ракет. Для расчета траекторий корабля и ракет учитывались инерция и гравитация по законам Ньютона. Игрок должен был учитывать их, заранее предполагая, как это повлияет на результат. Такое несложное краткосрочное планирование давало интерес к игре и использовалось в игре Elite в части стыковки с орбитальной станцией и в игре Lunar Lander, имитирующей посадку на луну, в которой нужно нейтрализовать притяжение Луны, используя ускорители для маневрирования и замедления падения под действием притяжения. Второй момент, дававший интерес к игре - быстрая смена игрового поля, реализованная "прыжками в гиперпространство". Для звёздного фона в виде точек использовался генератор случайных чисел. Разработка игры заняла 6 недель и 200 человеко-часов.



Несложная игра, обычно, является первой программой, которую создают при изучении программирования. Человек стремится повторить то, с чем сталкивается при изучении чего-то нового. При изучении языков программирования может даже возникать желание изобрести собственный язык программирования.

Пол Аллен, основатель Микрософт, в своей книге "Idea man" писал:

Каждому новичку нужен наставник, и в C-Cubed их было трое. Все они были программистами мирового класса, нердами с налётом эксцентричности. В отличие от бизнес-руководства, они не относились к нам как к досадной помехе. Я думаю, они видели в нас себя в молодости. Порой мне казалось, что из старших классов я попал на семинар аспирантов по продвинутому системному программированию.

***Стив Рассел**, по кличке "Тормоз", главный по железу, был невысоким и полным, с едким чувством юмора. В тридцать один год он вслед за Джоном Маккарти перебрался из Дартмута в Массачусетский технологический институт. Там Рассел создал на компьютере PDP-1 первую по-настоящему интерактивную компьютерную игру "Звёздные Войны".*

Билл Вайер, худой очкарик, говорил мало. Известный тем, что разработал SOS (Son of STOPGAP, один из первых текстовых редакторов), он выглядел как средневековый писец. Я видел, как он без устали корпит за своим терминалом, создавая сложные структуры замысловатого кода.

Дик Грюн, бывший консультант DEC, познакомившийся с Расселом и Вейхером в Стэнфорде, был самым общительным из всех, фанатом фастфуда и скабрезных шуток, с копной кудрявых волос. По словам Грюна, не родилась еще та операционная система, которую он не смог бы подвесить (сбой операционной системы, когда она не может ничего выполнять и требует перезагрузки). И ему нельзя было не поверить.

Для них мы были "дети из Лейксайда" (название школы) или "тестировщиками". Изредка они говорили нам с Биллом Гейтсом одновременно запускать несколько экземпляров шахматной игры, чтобы подвесить операционную систему. Это задание отлично соответствовало подростковому стремлению ломать всё подряд ради развлечения,

одновременно направляя это стремление во что-то полезное. Как я позже сказал одному журналисту из Сиэтла: "Самый эффективный способ обучения - это практический опыт работы с лучшим на тот момент компьютером, изучение того, как он работает, что нужно, чтобы сделать на нём что-то полезное или не сделать (make it or break it)".

Другой подход заключался в запуске стресс-тестов (вид тестирования для определения пределов, при которых программа способна работать без ошибок) до тех пор, пока программа не зависала. Мы документировали на бумаге, какие наши действия привели к зависанию и продолжали тестировать. Настоящим хитом было подвесить операционную систему, что можно было определить по зависанию и жужжанию терминала, если жать на клавиши. Рассел и Грюн находили причину сбоя, и радовались как дети, зная, что им не выставят счёт за использование процессорного времени компании DEC, так как биллинг завис вместе с операционной системой. Мы с Биллом тоже радовались. Пока мы находили ошибки, нас не прогоняли и позволяли работать с компьютером.

RISC-V

Компания ARM продаёт лицензии на выпуск процессоров архитектуры ARM. С лицензией даётся схема ядра процессора, средства разработки программ (компилятор, отладчик) и право продавать произведенные процессоры с небольшой платой за каждый выпущенный процессор. Лицензии разнообразны, например, есть лицензии на архитектуру ARM, которая позволяет разрабатывать собственные процессора с набором инструкций ARM и использующие патенты компании ARM, которая не требует платы за каждый выпущенный процессор.

В 2010 появилась система команд RISC-V. В 2015 году право стандартизации было передано международной организации RISC-V International. В 2022 году в ядро Linux была добавлена поддержка команд RISC-V. В 2022 году производитель микроконтроллеров (SoC, систем на кристалле) ESP32 для "интернета вещей" стал использовать систему команд RISC-V. Переходу на RISC-V способствует отсутствие лицензионных отчислений, которые есть у ARM.

Система команд RISC-V определяет 32 целочисленных регистра общего назначения и 40 базовых команд для 32-битной архитектуры. Длина команд 32 бита, порядок байт только little-endian. Система команд RISC-V развивается, в неё добавляются новые команды. Описания команд (Instruction Set Architecture, ISA), которые обязательно должны быть, называются "профилями". Например, в октябре 2024 года был описан профиль RVA23, до этого были профили RVA22 и RVA20. В профиле RVA23 стали обязательными команды векторных операций, виртуализации, инструкций с плавающей точкой, атомарных операций.

Производитель операционной системы Ubuntu, одной из популярных сборок Linux, объявил, что в будущем будет поддерживать процессора только с профилем RVA23. Это означает, что в коде будущих версий Ubuntu будут использоваться векторные операции и виртуализация. Если в процессоре команды будут отсутствовать, то операционная система на таком процессоре не сможет работать. Требование наличия обязательного набора команд упрощает разработку операционных систем - разработчикам не нужно писать код, программно воспроизводящий те действия, которые может эффективно выполнять процессор. Создание кода эмуляции - пустая трата времени разработчиков, так как такой код заведомо будет работать медленно, а программные решения не имеют научной ценности.

Успех архитектуры RISC-V будет определяться тем, насколько удачно некоммерческая организация RISC-V International, находящаяся в Швейцарии, будет развивать стандарты по сравнению с британской компанией ARM, которой владеет японская компания Softbank. Будет ли у RISC-V выдерживаться баланс между простотой реализации команд на уровне железа и удобством генерации процессорного кода.

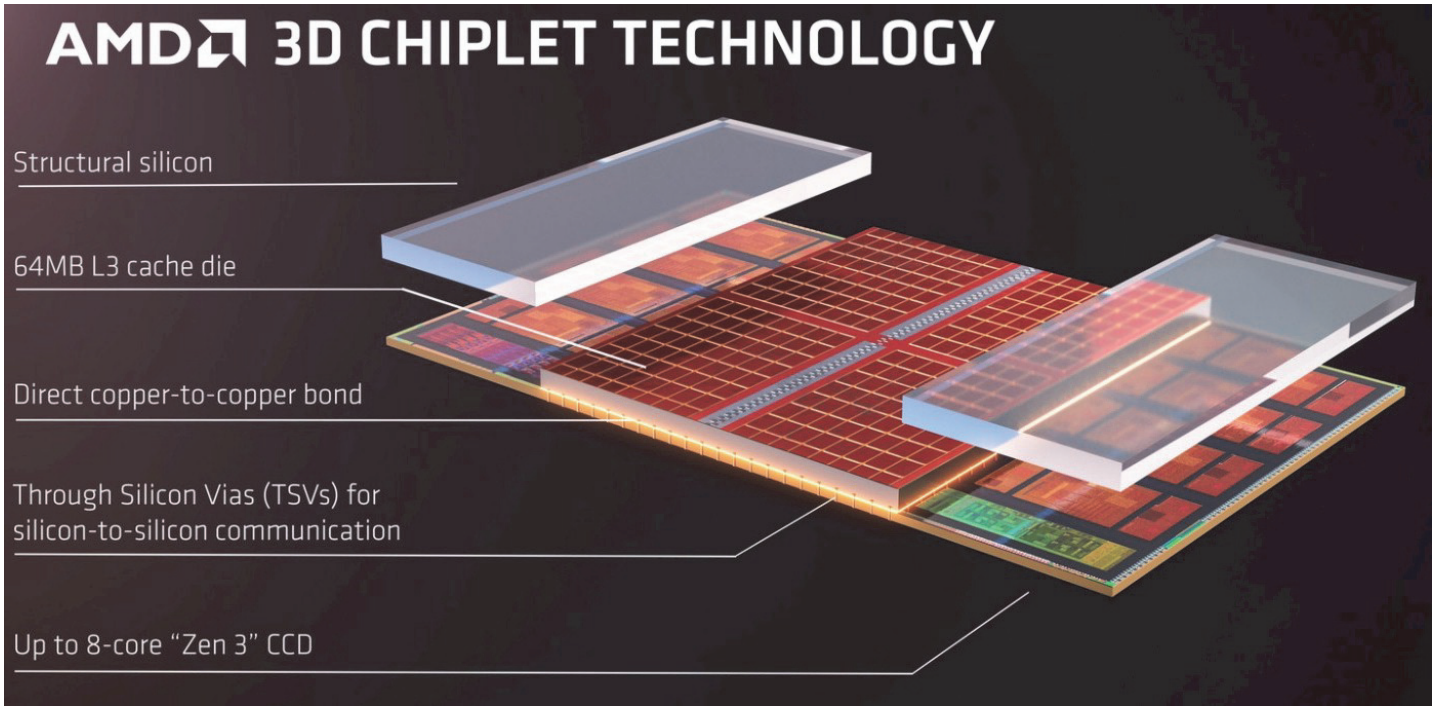
EPYC это не EPIC

EPYC - название современной серии процессоров AMD. Поколения процессоров называются как города Италии: Naples (Неаполь), Rome (Рим), Milan (Милан), Genoa (Генуя). В архитектуре этих процессоров используется удачная пространственная компоновка. Кэш третьего уровня расположен поверх ядер процессора, общаясь с ними через медные проводники. Компоновка позволила увеличить кэш третьего уровня, уменьшить длину соединений. Уменьшение длины соединений позволило увеличить частоту работы с кэшем и уменьшить тепловыделение. Быстрый и большой кэш третьего уровня обеспечивает высокую производительность и преимущество по сравнению с конкурентами.

EPIC (explicitly parallel instruction computing, явное распараллеливание команд процессора) - развитие архитектуры процессоров VLIW (Very Long Instruction Word, очень длинная машинная команда). Идея VLIW в том, чтобы одна "длинная" (много операндов) команда описывала несколько "коротких" (мало операндов) команд, которые могут выполняться параллельно. "Длинные" команды должны формироваться на этапе компиляции. У скалярной архитектуры распараллеливанием команд занимается аппаратное устройство процессора - планировщик команд. Изобретатели VLIW подумали, что удаление планировщика уменьшит сложность процессора, а компилятор проанализирует программный код более эффективно. Однако большой размер операндов приводил к тому, что надо было загружать их из основной памяти через кэши. Компилятор не может предсказать задержки при подгрузке операндов из кэшей памяти в регистры процессора. Компания Intel решила попробовать разработать процессор архитектуры VLIW, добавив архитектурные решения, позволяющие сгладить проблему. Эти архитектурные решения назвали EPIC. Слово epic означает легендарно, поразительно. Однако, "эпично" используется и в словосочетании epic fail (эпичный провал). Это не насторожило руководство Intel. Более того, при выпуске на рынок процессор назвали Itanuim, а это созвучно названию "непотопляемого" корабля Titanic, который затонул в первое же плавание. Неудивительно, что процессор Intel Itanium эпично провалился.

Были попытки использовать архитектуру VLIW и в других процессорах. Например, Transmeta, основанная в 1995 году, когда технология VLIW ещё считалась многообещающей, создала процессоры Crusoe и Efficeon, которые не получили успеха. Transmeta прекратила выпускать процессоры и пыталась продавать свои патенты более удачным производителям процессоров, но не преуспела. В конечном итоге, Transmeta была куплена другой компанией, которая обанкротилась. Архитектура Эльбрус (ELBRUS, ExpLicit Basic Resources Utilization Scheduling, явное планирование использования основных ресурсов) является развитием VLIW.

Архитектура процессора Itanium имеет техническое название IA-64, что не имеет отношения к архитектуре x86-64, так же как EPIC не имеет отношения к EPYC. Создателем архитектуры x86-64 является компанией AMD. AMD создала архитектуру x86-64 так, чтобы она была совместима с x86.



Arduino

В начале 2000 года в итальянском городе Ivrea проводились курсы обучения программированию. На курсах использовались недорогие платы с PIC-микроконтроллерами, а для программирования использовался вариант языка BASIC. PIC и BASIC не самые удачные решения, но самые простые и дешёвые. Студенты с преподавателями создали плату на микроконтроллере ATmega 128 и простую среду разработки, использующую вариант языка C. Простота использования, свободное лицензирование сделали проект Arduino популярным. Название Arduino связано с баром "Arduino" в городе Иврея, где собирались основатели платформы. Бар назвали по имени короля Италии, избранного королём в 1002 году (в те времена королей избирали), маркграфа Ивреи.

Использование Arduino не требует больших знаний электроники и программирования. Технология Arduino обрела популярность, появилась периферия: платы расширения со стандартизованными разъемами, позволяющими состыковать платы с платами Arduino. Позднее, вместо ATmega 128 стали использоваться более быстрые процессоры ARM и ESP32.

Кроме Arduino имеются и более производительные одноплатные компьютеры типа Raspberry Pi, но их стоимость существенно выше.

BBC micro:bit

Через 30 лет после создания микрокомпьютера BBC Micro, разработка которого привела к появлению архитектуры процессоров ARM, британская вещательная корпорация BBC решила повторить успех и запустила проект по обучению программированию школьников. Разработали и бесплатно раздавали британским школьникам 11-12 лет одноплатный микрокомпьютер, который назвали BBC micro:bit. В 2015 году в британские школы передали 1 миллион микрокомпьютеров для учащихся. В 2018 году было выпущено 2 миллиона экземпляров BBC micro:bit для распространения в 50 странах мира. В микрокомпьютере BBC micro:bit используется процессор ARM Cortex-M0.

Компания Microsoft решила участвовать в общественно полезном деле обучения школьников и создала для BBC micro:bit среду визуального программирования под названием MakeCode. Среда разработки бесплатно доступна как веб-приложение на сайте microbit.org или как отдельное приложение. В MakeCode есть эмулятор микрокомпьютера BBC micro:bit (программа, имитирующая работу микрокомпьютера). Благодаря наличию эмулятора можно тестировать программы не имея микрокомпьютера BBC micro:bit. Кроме использования

визуального программирования в MakeCode можно писать программы на языках JavaScript и Python. Визуальное программирование рассчитано для использования детьми от 8 лет.

Разработка программ для BBC micro:bit возможна на Arduino IDE с использованием языка C, который использует среда разработки Arduino IDE.

Для отображения результатов работы программы в micro:bit имеется 25 светодиодов и две кнопки для передачи сигналов программе.

Хошен

В Библии (первая в истории книга, напечатанная типографским способом Гуттенбергом, а не переписанная вручную) описывается нагрудная доска (хошен), к которой были прикреплены 12 разноцветных камней (светодиодов в то время не было), расположенных по три камня в четырех рядах. На камнях было изображено по 6 букв, в алфавите 22 буквы. Доска использовалась главным священником как терминал для получения ответа на свои вопросы. С доской были связаны два устройства с названиями "урим" и "туммим" (on/off). Буквы, изображенные на камнях, подсвечивались, а священник интерпретировал ответ.

Язык программирования BASIC

Язык BASIC (Beginner’s All-purpose Symbolic Instruction Code, универсальный символьный код инструкций для начинающих) был создан в 1964 году для обучения студентов-непрограммистов новому предмету "Программирование".

Язык BASIC был спроектирован для использования на терминалах в интерактивном режиме. В то время вместо экрана и клавиатуры использовались "телетайпы" - электрические печатные машинки.

Altair 8800 - Video #7.1 - Loading 4K BASIC with a Teletype

Пример программы на языке BASIC из одной строки: `PRINT 2+2`

Можно было набрать одну или несколько строк программы и получить результат. Язык BASIC стал набирать популярность после появления микрокомпьютера Altair 8800. BASIC был правильно выбран для микрокомпьютеров того времени по причинам:

1) язык прост и будет наиболее удобен для пользователей микрокомпьютеров;

2) язык прост и интерпретатор языка не будет занимать много памяти. Конкуренцию по критерию размера кода интерпретатора мог создать только язык FORTRAN, но он был сложнее BASIC;

3) основатели Microsoft, Пол Аллен и Билл Гейтс знали язык, так как изучали его в школе, где они вместе учились.

2+2 или Hello Word!

В первых компьютерах для тестирования работоспособности писали программу, которая должна была вычислить 2+2 и вывести результат на устройство вывода этого компьютера. Устройства вывода назывались: экран, монитор, терминал, дисплей, принтер, консоль.

Позднее первая программа должна была отобразить фразу "Hello word!".

Программа состоит из небольшого фрагмента кода и используется:

1) для проверки того, что программное обеспечение установлено и работает правильно, может компилировать и выполнять программы;

2) для иллюстрации базового синтаксиса языка;

3) по традиции пишется как первая программа при изучении языка программирования.

Впервые фраза "Hello word!" была использована в книге 1978 года Брайана Кернигана и Денниса Ритчи "Язык программирования Си".

Похожая фраза "hello word" использовалась чуть раньше (в 1972 году) Брайаном Керниганом в руководстве по языку B.

Microsoft

Обложка журнала Popular Electronics за январь 1975 года с изображением компьютера Altair 8800 воодушевила Билла Гейтса и он написал интерпретатор языка BASIC. Это был первый язык программирования для первого персонального компьютера и первая программа компании Micro-Soft.

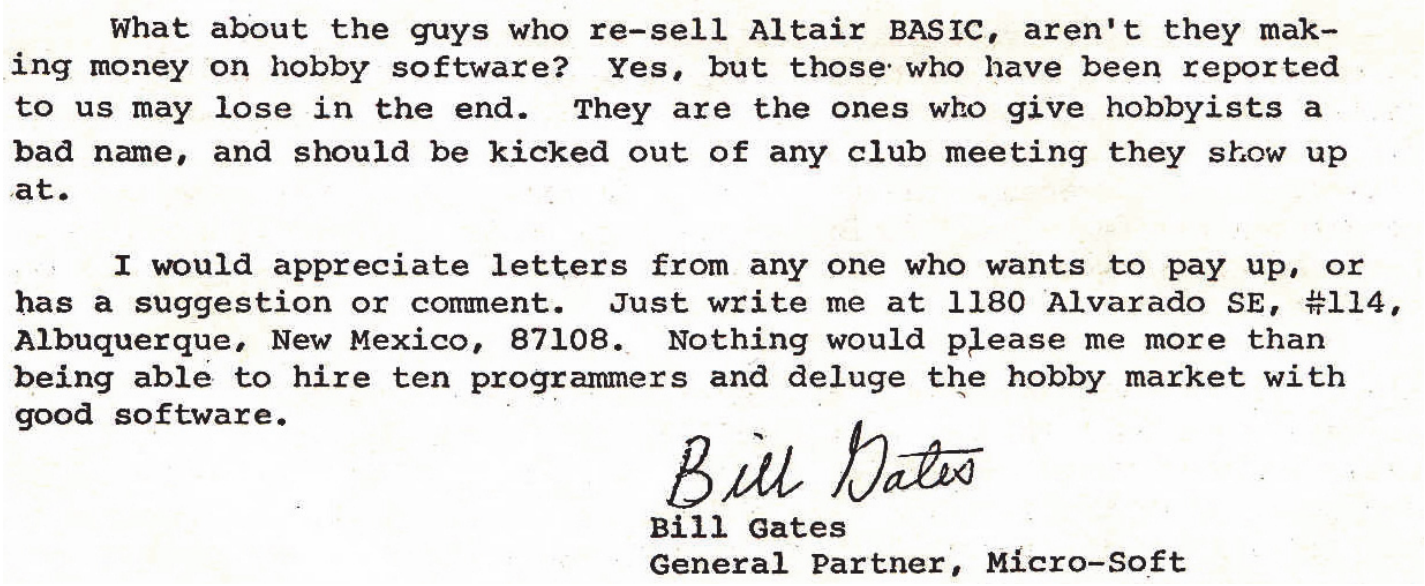
BASIC создавался без самого компьютера, разработка велась на компьютере DEC PDP-10, для которого Пол Аллен, ещё до появления журнала с фотографией Altair, написал эмулятор и отладчик для процессора Intel 8008. Пол Аллен и Билл Гейтс использовали эмулятор для написания программы изучения транспортных потоков. Они хотели продавать муниципалитетам услуги по анализу трафика для оптимальной настройки светофоров. Как только они написали программу, штаты стали предоставлять такие услуги муниципалитетам бесплатно. Бизнес-проект двух друзей потерпел фиаско, но у них появился эмулятор процессора Intel 8008.

Пол Аллен адаптировал эмулятор и отладчик для процессора Intel 8080A, который использовался в Altair, а Билл Гейтс написал интерпретатор языка BASIC для Altair. Размер программы был небольшой - чуть меньше 4 килобайт.

Билл Гейтс предположил, что цены на компьютеры снизятся, компьютеры станут популярны, компьютерам будут нужны программы и можно будет заработать на продаже программ.

BASIC для Altair продавался в розницу за 150$. Он стал популярен среди владельцев компьютера Altair 8800, однако, владельцы предпочитали делиться друг с другом копиями Altair BASIC, а не приобретать их. С самого основания Microsoft страдала от компьютерного пиратства.

3 февраля 1976 года Билл Гейтс написал открытое письмо компьютерному сообществу, в котором выразил разочарование, что большинство компьютерных энтузиастов, использовавших Altair BASIC, не заплатило за программу. Причиной написания письма было то, что полученный Биллом и Полом доход от продажи программы соответствовал оплате труда всего 2 доллара в час, что было довольно мало.



What about the guys who re-sell Altair BASIC, aren't they making money on hobby software? Yes, but those who have been reported to us may lose in the end. They are the ones who give hobbyists a bad name, and should be kicked out of any club meeting they show up at.

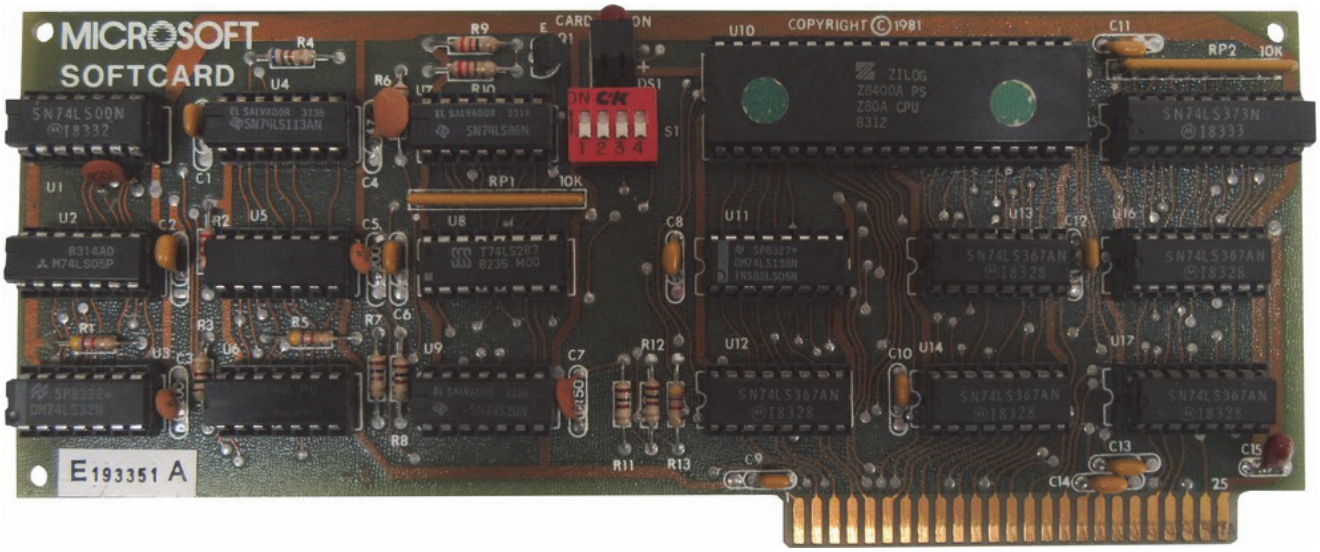
I would appreciate letters from any one who wants to pay up, or has a suggestion or comment. Just write me at 1180 Alvarado SE, #114, Albuquerque, New Mexico, 87108. Nothing would please me more than being able to hire ten programmers and deluge the hobby market with good software.

Bill Gates

Bill Gates
General Partner, Micro-Soft

В дальнейшем, компания Microsoft создала интерпретаторы BASIC для всех микрокомпьютеров. Интерпретаторы BASIC оставались основным источником дохода Microsoft до появления операционной системы MS-DOS.

2 апреля 1980 года Microsoft выпустила плату расширения Z80 SoftCard для компьютера Apple II за 349 долларов и лицензировав CP/M, успешно продавала экземпляры CP/M в тех же количествах, что и Digital Research. Более того, старт продаж Z80 SoftCard принесли Microsoft половину годового дохода. Основной причиной спроса стало то, что пользователи смогли запускать популярный текстовый редактор WordStar на Apple II. Компьютеры Apple II были популярны, но не поддерживали программы для Intel 8080A и совместимого с ним по набору команд процессора Zilog Z80.



CP/M и MS-DOS

Операционная система CP/M (Control Program for Microcomputers, управляющая программа для микрокомпьютеров) была написана программистом Гари Килдалл на написанном им же языке программирования, который он назвал PL/M (Programming Language for Microcomputers).

В 1976 году Гари и его жена создали компанию Digital Research для продажи копий операционной системы CP/M. К 1980 году компания продала 250 тысяч копий операционной системы CP/M. CP/M была наиболее популярной операционной системой для компьютеров на процессорах Intel 8080A. С выходом процессоров Zilog Z80, Motorola 68000, Intel 8088, Intel 8086 CP/M была адаптирована для этих процессоров.

В 1980 году компания IBM осознала потенциал микрокомпьютеров как платформы для корпоративного рынка. IBM обратилась к Microsoft (которая к тому времени продала полмиллиона копий интерпретатора BASIC), желая иметь в новой платформе IBM PC (рабочее название проекта "Шахматы") компиляторы и интерпретаторы языков программирования. В IBM понимали, что без программ компьютеры бесполезны, а пишут программы на языках программирования, поэтому нужно, чтобы у новой платформы был качественный набор языков программирования. Microsoft выпускала компиляторы и интерпретаторы множества языков, в том числе FORTRAN, COBOL, Pascal.

Билл Гейтс склонил руководителей проекта "Шахматы" к использованию более нового процессора Intel 8086, вместо Intel 8080A. В сентябре 1980 года IBM стала искать операционную систему для этого процессора, так как поняла, что операционная система была бы тоже полезна. Пол Аллен в своей книге "Idea man" описал события того времени:

Билл Гейтс позвонил Гари Килдаллу и сказал: "Я отправляю к тебе несколько человек и хочу, чтобы ты хорошо с ними обращался, потому что мы оба заработаем на этой сделке много денег". Он не назвал IBM по имени, поскольку компания настаивала на максимальной конфиденциальности. По стечению обстоятельств Кидалл был в командировке, а его жена и бизнес-партнер отказалась подписывать соглашение о неразглашении и предложила собственную форму документа. Руководитель проекта IBM вернулся в Microsoft и сказал: "Я не думаю, что мы сможем работать с этими ребятами, нашему юридическому отделу понадобится полгода, чтобы оформить все документы. У вас есть другие идеи? Вы могли бы справиться с этим самостоятельно?". Билл Гейтс был взбешён, так как под угрозу ставился весь проект, ведь у Microsoft не было собственной операционной системы.

После поисков, Microsoft приобрела права на 16-разрядный клон CP/M у компании Seattle Computer Products за 10000 долларов плюс 15000 долларов для каждой компании, которая бы купила лицензию у Microsoft. В сумме 25000 долларов, так как у Microsoft был только один клиент - IBM.

Откуда появился клон? Тим Паттерсон из Seattle Computer Products продавал плату расширения SCP-200B для системной шины S-100 с процессором Intel 8086 с начала 1980

года, но спрос был небольшой, так как клиентам была нужна операционная система. Килдалл обещал выпустить CP/M-86, но не выпустил.

Тим Паттерсон, на скорую руку, с апреля по июль 1980 года, написал 16-битную операционную систему с рабочим названием QDOS (Quick and Dirty Operating System, сделанная на скорую руку). Закончив писать QDOS, Паттерсон переименовал её в 86-DOS.

Паттерсон говорил: "Мы были бы счастливы, если бы операционную систему сделал кто-то другой. Если бы Digital Research выпустила ее в декабре 1979 года, сегодня в мире не было бы ничего, кроме CP/M".

6 ноября 1980 года IBM заключила договор с Microsoft стоимостью 430000 долларов: 75000 долларов за адаптацию, тестирование и консультации; 45 000 долларов за операционную систему; 310 000 долларов за интерпретаторы и компиляторы языков.

IBM устанавливала операционную систему на компьютеры своего производства под названием PC DOS (Personal Computer Disk Opeating System), Microsoft продавала копии операционной системы под названием MS-DOS за 40 долларов. Через какое-то время Digital Research выпустила CP/M-86 для IBM PC, но из-за высокой стоимости в 240 долларов CP/M-86 не пользовалась спросом.

В 1982 году появилась версия CP/M-68K для процессора Motorola 68000. Изначально она была написана на языке Pascal/MT+68k, но позже переписана на языке C. Перенос на язык C позволил легко адаптировать CP/M-68K для 16-битных процессоров Zilog Z8001 и Z8002.

В 1988 году Digital Research на основе CP/M выпустила для компьютеров архитектуры Intel 80x86 MS-DOS-совместимую операционную систему, назвав её DR-DOS версии 3.41. Номер версии был выбран так, чтобы соответствовать текущей версии MS-DOS. DR-DOS продавалась за 270 долларов, а MS-DOS за 40 долларов. Разницу в цене Килдалл обосновывал тем, что считал свою операционную систему профессиональным продуктом, а MS-DOS "игрушечным". Профессиональность была малозаметна пользователям. Практическим преимуществом DR-DOS была возможность защищать директории паролями от доступа и файлы от записи. Идея разграничения дискового пространства была взята из CP/M. Это было полезно на компьютерах организаций, которые совместно использовались многими пользователями, например, в учебных заведениях. Для обычных пользователей такие возможности не оправдывали разницу в цене. Возможности DR-DOS оказались несовместимыми с Microsoft Windows. DR-DOS перестала пользоваться спросом после появления Windows. Важны не только возможности, но и совместимость программ.

Часть 3. Программы

Поколения языков программирования

Машинные коды (команды процессора) - это первое поколение языков программирования (1GL, first-generation programming language). У первых компьютеров команды вводились переключателями на консоли компьютера или с помощью перфокарт и перфолент.

Ко второму поколению языков (2GL) относят ассемблеры - сборщики (трансляторы, преобразователи) программ из текста на языке ассемблера в машинные коды. Вместо машинных кодов в ассемблерах используют обозначения команд и операндов.

2GL = ассемблеры

Ассемблеры зависят от системы команд процессора и способов адресации ячеек памяти. Общеупотребительного синтаксиса ассемблера не существует. Базовой конструкцией языка ассемблера является мнемонический код (мнемокод) - сокращённое до 2-4 символов название команды процессора. Пример для процессора x86 в синтаксисе Intel:

```
mov eax, 7
```

пересылает (move) в регистр `eax` число `7`. Регистр `eax` и `7` являются операндами команды `mov`. Операндами могут быть регистры, константы, адреса ячеек памяти и портов ввода-вывода, метки. Команда с такими операндами будет транслирована в машинный код `B8`.

Для ассемблеров есть два вида синтаксиса: AT&T и Intel. Основное отличие - разный порядок операндов. В синтаксисе AT&T приведённая команда записывается как:

```
movl $7 , %eax
```

Разный синтаксис двух видов ассемблеров создаёт неудобство при чтении программ. Синтаксис AT&T был разработан для ассемблера PDP-11 и используется для архитектур процессоров, отличных от x86.

Более того, даже комментарии в разных ассемблерах несовместимы друг с другом, используются три варианта: "`;`" "`//`" "`#`".

Написание программы на ассемблерах трудоёмко: нужно пользоваться стеком, ограниченным количеством регистров для выполнения простейших действий. Программы на языке ассемблера сложно перенести на машину с другой архитектурой.

С появлением языка С необходимость в использовании ассемблера стала снижаться. Программы на языке С были настолько же эффективны, как и программы, написанные на ассемблерах. Применение ассемблеров сузилось до микрокомпьютеров, а потом до микроконтроллеров. То есть устройств, у которых мало памяти и невысокая производительность, а значит, программы просты настолько, что человек сможет написать небольшую программу на низкоуровневом языке ассемблера. Также применялись "ассемблерные вставки": большая часть кода пишется на языке высокого уровня типа С, а участки, для которых критична производительность, либо требующие обращения непосредственно к аппаратным ресурсам, писались на ассемблере.

Язык С повлиял на формирование всех языков, создаваемых начиная с 1980-х годов. Часть языков создавались как прямые наследники (C++, C#, Objective C), но не стали лучше самого С, часть языков использовали его синтаксис и идеи.

Деннис Ритчи: "С - это причудливый, несовершенный, но невероятно успешный язык".

Ассемблером пользовались в случаях, когда новый процессор или микроконтроллер начинали выпускать, а программное обеспечение к нему не успевали написать. Отсутствие программного обеспечения не препятствие для начала поставок процессоров. Компиляторы для высокоуровневых языков под выпускаемый процессор писались и совершенствовались уже после выпуска процессоров и микроконтроллеров. Как только компиляторы с языков высокого уровня дописывались, от ассемблерных вставок постепенно избавлялись. В настоящее время, ассемблеры почти не используются, так как компиляторы высокоуровневых языков генерируют более производительный машинный код, чем мог бы написать средний программист на ассемблере.

С появлением ассемблеров возникло понятие "исходного кода" (текстового представления программы, которое отличается от машинного кода, который понимает процессор). Ассемблерный код называли исходным кодом. Ассемблерный код транслировался (преобразовывался) в машинный.

В ассемблерах имеются идентификаторы начала частей кода, которые сильно упростили написание программ. При трансляции идентификаторы заменяются адресом участка кода, который становится известен только после завершения написания программы.

В ассемблерах впервые появились конструкции, которые затем получили развитие в других языках:

1) Макросы - возможность дать название участку кода и указывать название макроса наравне с командами. Транслятор подставит вместо макроса участок кода. Макросы с параметрами похожи на подпрограммы (процедуры). Макросы - "строительные кубики" программ на ассемблере. Они существенно упрощали написание кода.

2) Метки для передачи управления (jump, jmp) на помеченный код.

3) Директивы - команды транслятору. Позволяют влиять на процесс трансляции. Директивы - предшественники аннотаций ("прагм"), которые в высокоуровневых языках добавляют элементы декларативности или расширяют возможности языка без изменения его синтаксиса.

4) Именованные константы, которые позволяют давать имя числу, и упрощают изменение значения константы в исходном коде.

5) комментарии. Без комментариев сложно разобраться в ассемблерном коде.

Пока компьютеры были простыми и решали несложные вычислительные задачи, программирование было не слишком утомительным. По мере усложнения задач, которые решались компьютерами, написание программ на ассемблерах становилось более сложным.

3GL = высокоуровневые языки

Третьим поколением (3GL) стали языки программирования высокого уровня абстрагирования (отдаления) от машинного кода. Первым языком высокого уровня стал FORTRAN (formula translator). FORTRAN был создан в 1954-1957 годах группой программистов, работавших в IBM под руководством Бэкуса. Можно было бы упомянуть и язык Plankalkül, но он разрабатывался изолированно от мирового сообщества и не внёс вклад в развитие программирования.

Вторым языком высокого уровня был язык B-0 (Business Language Version 0), который также назывался FLOW-MATIC. Он был создан для компьютера "UNIVAC I" группой программистов под руководством Грейс Хоппер. На основе B-0 был создан язык COBOL, который в 1960х годах стал основным языком разработки деловых и экономических программ. Грейс Хоппер называют "бабушкой COBOLa". Изучать COBOL и писать на нём программы было сложно, в нём было более 300 зарезервированных слов и 43 оператора.

К языкам 3GL относятся почти все языки программирования, поэтому деление языков на поколения потеряло смыл. Языки стали классифицировать по другим критериям, например, по списку используемых парадигм.

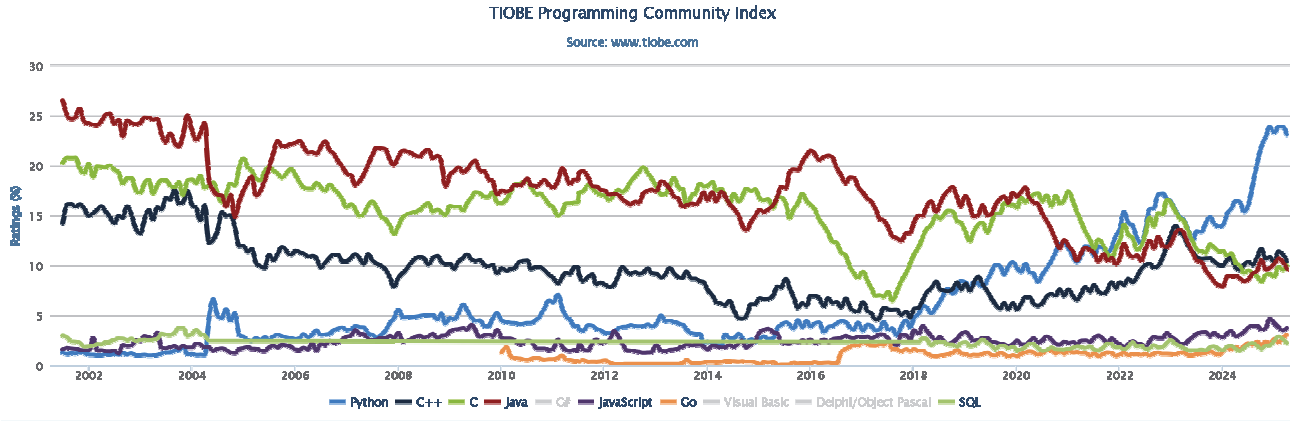
Можно встретить выделение поколений 4GL и даже 5GL, но языки, которые относят к ним не лучше языков 3GL. К 4GL относят узкоспециализированные языки, которые решают конкретную задачу (domain-specific, под конкретную предметную область). Это даже не языки, а средства разработки, в которых визуально создают декларативное описание свойств программы. Граница отнесения к 3GL или 4GL условна.

Пятым поколением стали называть языки четвертого поколения в рекламных целях. Люди думают, что 5 больше (новее), чем 4, а значит лучше. К пятому поколению относят средства автоматизированного создания программ с помощью визуальных средств разработки, без знания программирования ("low-code"). То есть, программист что-то конфигурирует в визуальной среде разработки, а среда разработки сама создаёт текст программы на каком-нибудь универсальном языке. Идея, что можно создать среду разработки, где не надо писать код, а только кликать мышкой посещала программистов всегда. Обычно всё сводилось к тому, что программисту надо было знать: как и какой код генерируется при выполнении манипуляций в среде разработки, то есть сам язык программирования и среду разработки. Получалось более трудоёмко, чем сразу писать на языке программирования.

Индекс TIOBE

Это индекс, оценивающий популярность языков программирования на основе подсчёта числа поисковых запросов, содержащих название языка. То есть запросы вида: "что-то язык". Индекс не оценивает языки по качеству или объему написанного кода.

Наиболее популярны запросы по языкам: C, Java, Python, C++.



Парадигмы программирования

Парадигма (модель, стиль мышления) - то, как программист представляет себе процесс (логику) обработки данных. Современные языки мультипарадигменные - можно писать в разных стилях, но какой-то стиль ведущий и его удобнее использовать. Примеры парадигм:

- Управление потоком выполнения: свободное (ассемблер, GOTO)/структурное;
- Императивное/Декларативное
- Процедурное/Функциональное
- Алгоритимическое/Объектно-ориентированное

Алгоритм

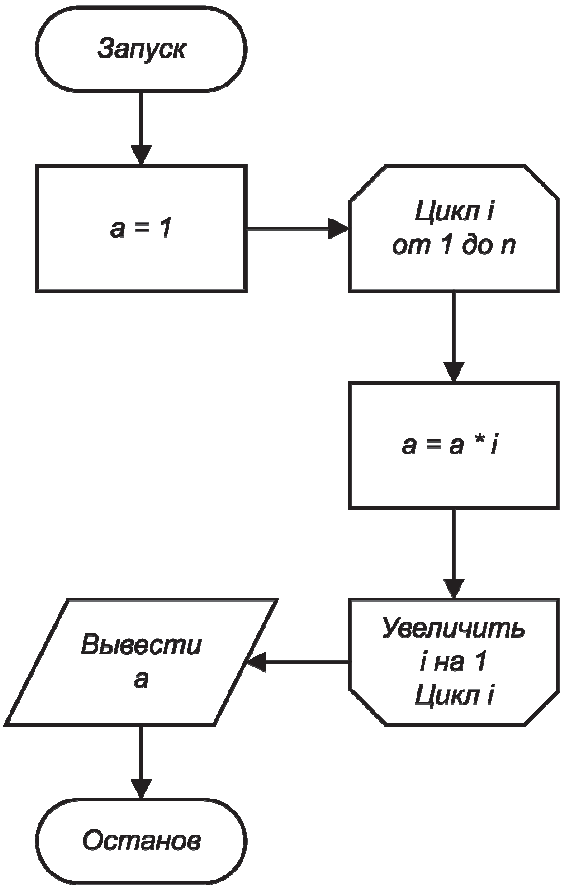
Алгоритм - инструкции, описывающие порядок действий, чтобы решить задачу. Вместо слова "порядок" когда-то использовалось слово "последовательность", но по мере реализации техник распараллеливания в работе процессоров слово "последовательность" стали заменять более общим словом "порядок". Инструкции могут быть упорядочены в коде программы, а внутри процессора выполняться параллельно, а не последовательно.

Алгоритмы естественны для мышления людей, так же как и целые числа. Например, рецепт приготовления блюда является алгоритмом, по которому блюдо готовится.

Блок-схема

Блок-схема - графическое представление алгоритма. Используется для документирования того, как работает алгоритм. Блок-схемы использовали, когда программы писались на ассемблере или в машинных кодах. Программа на языках программирования высокого уровня с комментариями более просто описывает алгоритм, чем блок-схема. Программы на объектно-ориентированных языках программирования в виде блок-схемы описать нельзя.

Блок-схемы использовали при изучении алгоритмов перед переходом к изучению языков программирования. Это было оправданно в языках, где часто используется GOTO (оператор перехода на строку программы, название которой указывается в параметре оператора). Пример блок-схемы, описывающей алгоритм вычисления факториала:



Программа на языке C, соответствующая блок-схеме проще:

```
a=1;
for(i=1; i<=n; i++)
{
 a=a*i;
}
return a;
```

До появления языка C, алгоритмы в научных публикациях описывали на языке Algol.

Графическое представление может быть удобным, однако, графическое представление не является самоцелью. Например, диаграммы, использующие UML (универсальный язык моделирования) насколько "универсальны", настолько и бесполезны.

Обычно, документирование выполняется по остаточному принципу и не самыми квалифицированными специалистами. Если вы увидите диаграммы, которые вам будут непонятны, это нормально.

Если диаграммы упрощают разработку, то они полезны. Например, полезны диаграммы классов в объектно-ориентированных языках программирования. Польза в том, что среда разработки на основе диаграммы создаёт набор файлов с шаблонным программным кодом, которые можно дорабатывать, добавляя программный код.

Расширенная форма Бэкуса-Наура (РБНФ)

РБНФ разработана Виртом и является упрощением "Формы Бэкуса-Наура" (БНФ). БНФ предназначена для описания синтаксиса языков программирования. Преимуществом РБНФ является простота и понятность для людей. В РБНФ используется всего 10 специальных знаков: `[ ] { } ( ) | = "` (три вида скобок, вертикальная черта, знак равно, кавычки, запятая). Синтаксис определяется пятью правилами. РБНФ является стандартом ISO/IEC 14977. Других стандартов для описания синтаксиса нет, поэтому РБНФ используют в документации. Пример описания команды в РБНФ:

```
CREATE OPERATOR имя (
    {FUNCTION|PROCEDURE} имя_функции
```

```
    [, LEFTARG тип_слева ] [, RIGHTARG тип_справа ]
    [, COMMUTATOR коммут_оператор ] [, NEGATOR обратный_оператор ]
)
```

Фигурные скобки `{|}` - набор фраз, разделенных знаком `|`, имеющим смысл OR. В квадратных скобках опциональные фразы. Пример команды, которая подходит под это описание:

```
CREATE OPERATOR == (
    FUNCTION equals,
    LEFTARG string,
    RIGHTARG string,
    NEGATOR <>
)
```

Бэкус разработал язык FORTRAN. Вирт разработал язык Pascal для обучения студентов структурному программированию, но он перестал развиваться и использоваться. В учебных целях язык Pascal не плох, но изучать синтаксис языка, который на практике не используется, неразумно.

На волне популярности объектно-ориентированных языков программирования в 1986 году в Apple был создан язык Object Pascal. Вирт не участвовал в создании Object Pascal, точно так же как Томпсон не участвовал в разработке языка C++. Примечательно то, что диалектом Object Pascal является популярный, в своё время, язык Delphi, о котором можно сказать примерно то, что написал Дейкстра о языке BASIC в 1975 году: "Программирование на COBOL калечит мозг. Поэтому обучение ему должно трактоваться как преступление". В настоящее время pascal-подобные языки не встречаются и не представляют опасности для разума программистов.

О языке BASIC Дейкстра писал: "Студентов, ранее изучавших BASIC, практически невозможно обучить хорошему программированию. Как потенциальные программисты они подверглись необратимой умственной деградации". Нет ничего плохого в том, чтобы писать первые программы на BASIC, но использовать его как основной язык для изучения программирования не стоит.

С долей шутки можно сказать то же самое про JavaScript, распространенность которого так же высока, как была в своё время у языка COBOL. Использование JavaScript сужает память программиста до нескольких строк кода, отчего появляется неспособность к проектированию программ (концептуальному мышлению).

Дейкстра был известен едкими афоризмами. Его коллега Алан Кей придумал афоризм в стиле Дейкстры: "Высокомерие в компьютерной науке измеряется в нано-дейкстрах", а по поводу афоризмов Дейкстры Кей говорил: "Две самые большие проблемы в нашей не совсем-то области - это те, кто слушал его слишком внимательно, и те, кто слушал его недостаточно внимательно".

Структурное программирование

В 1966 математики Бём и Якопини доказали следующую теорему. Программа, заданная в виде блок-схемы может быть представлена в виде трёх управляющих структур:

1) последовательность выполнения - стрелочка перехода из блока в блок на блок-схеме;

2) условный оператор IF THEN ELSE, реализующий branching, ветвление потока выполняемых команд;

3) цикл.

Суть теоремы в том, что реализация алгоритмов возможна без оператора перехода GOTO.

В то время часто использовался оператор перехода GOTO и программы получались запутанными, было сложно понять, как работает программа. Причины использования GOTO в том, что в машинных кодах и ассемблерах не было команд, реализующих циклы. Логика циклов и управление потоком выполнения команд реализовывались командой перехода `jump`

("перепрыгни"), аналогичной GOTO. Программисты, писавшие низкоуровневый код при переходе на высокоуровневые языки продолжали использовать то, к чему привыкли.

Проблема GOTO иллюстрирует то, что выбор языка программирования влияет на мышление и в будущем снижает вероятность написания эффективного и качественного кода при переходе на другие языки программирования и средства разработки.

Хоар, Вирт, Дейкстра создали структурное программирование. Дейкстра сформулировал простые принципы:

1) следует отказаться от оператора перехода GOTO;

2) программа строится из трёх управляющих структур: последовательность выполнения, ветвление, цикл;

3) управляющие конструкции (блоки) могут быть вложены друг в друга;

4) повторяющиеся фрагменты (кода, набора блоков) можно оформить в виде "подпрограмм" (процедур и функций) - программ с одним входом и выходом;

5) логически связанную группу инструкций стоит оформить как блок;

6) все структуры должны иметь один вход и один выход;

7) программы стоит разрабатывать "сверху-вниз" (от общего к частному). Начать с блоков входа (какие параметры программа принимает) и блока выхода (какой статус или данные должна вернуть программа), затем основные части в виде подпрограмм. Реализацию (кодирование, создание последовательности операторов на языке программирования) выполнять позже. Можно создавать "заглушки" подпрограмм, не реализуя их, что позволяет быстро создавать прототипы программ.

Блоки

Блок - группировка (обозначение) идущих подряд команд (инструкций, операторов, вызовов функций, других блоков) в исходном коде программы. Переменные (структуры данных, функции), определённые внутри блока "локальны" - видны только в пределах блока. При выходе из блока локальные переменные исчезают и память, которую они занимали, освобождается. Блоки:

1) помогают управлять временем жизни данных и областью видимости переменных (структур данных);

2) позволяют представлять набор команд как единое целое;

3) могут вкладываться один в другой.

Блок можно представить как промежуточный шаг к подпрограммам (процедурам, функциям, модулям, методам), так как блок всегда является телом подпрограммы. У блоков нет параметров, но так как они определяются в том же месте, где и код блока, то в блоках видны все переменные, определённые в том блоке, в котором они находятся. То есть у блоков "прозрачная" граница.

В JavaScript блок связан с малозаметными особенностями, которые усложняют их использование. Эти особенности нелогичны и интуитивно непонятны и, если о них не знать, приводят к побочным эффектам.

Обозначение блоков

В современных языках программирования блоки используются очень часто и синтаксис их использования должен быть простым. В языке C и C-подобных языках блоки обозначаются значками фигурные скобки: `{}`. Это интуитивно понятно и естественно. Ада Лавлейс использовала в первой программе фигурные скобки для группировки операторов в тела циклов. Удивительно, но до появления языка C фигурные скобки не использовали. Вместо них использовались английские слова `BEGIN` и `END`, пришедшие из Algol. В этих словах много букв, что ухудшает читабельность программ. Следование идее, что английские слова более приятны и понятны программистам, сделало написание кода неудобным. Большинство из этих языков больше не используется. Они не используются не только из-за слов "`begin-end`" вместо "`{}`". Эти слова - маркер, показывающий, что в языках и в остальном были выбраны неестественные решения.

В языке Ada попытались устранить недостаток, сделав использование begin и end опциональным, но ввели завершающие суффиксы для управляющих конструкций. Написание кода стало ещё менее понятным. Языки PL/SQL и plpgsql, используемые в реляционных базах данных, унаследовали синтаксис языка Ada. Пример:

```
begin
 if a>0 then
  a=a+1;
  a=a+1;
 end if;
end;
```

После `then` идёт набор команд без обозначения блока, это нелогично и делает код сложнее для воспиятия. На C-подобных языках запись была бы такой:

```
{
 if (a>0)
 {
  a=a+1;
  a=a+1;
 }
}
```

В языке Python, который появился в 1991 году, то есть позже, чем появился язык C, для обозначения блоков используются отступы, то есть невидимые символы: пробелы и табуляции. Это напоминает использование пробелов вместо нуля в вавилонской системе счисления.

Создатель языка Python обосновал использование отступов так:

"Использование отступов уменьшает визуальное нагромождение и делает программы короче, тем самым сокращая объём внимания, необходимого для восприятия базовой единицы кода. Во-вторых, это даёт программисту меньше свободы в форматировании, тем самым делая возможным более единообразный стиль, что облегчает чтение чужого кода (сравните, например, три или четыре различных соглашения о размещении фигурных скобок в Си, каждое из которых имеет сильных сторонников)".

Другими словами:

1) экономия на символах {}

2) склонять программистов к использованию отступов для форматирования (indentation). Отступы улучшают читабельность программы, но неудобно подсчитывать невидимые символы.

Не стоит принимать на веру причины, которые привёл создатель языка. Он мог забыть упомянуть, что сделал это для развлечения. Название языка Python происходит из комедийного телешоу "Летающий цирк Монти Пайтона".

В конечном итоге, история сохраняет лучшие решения, но в каждый момент времени существует много решений и существовать они могут долго. Долгое время люди считали, что солнце вращается вокруг земли или что ноль не натуральное число. Какая доля решений не сохранится? Скорее всего, по принципу Парето 80 к 20. Поэтому, не стоит ориентироваться на большинство. Стоит доверять себе, а не следовать чужим идеям или "принципиальным соображениям".

Размещение фигурных скобок

Можно сказать, что размещение фигурных скобок дело вкуса каждого программиста. Стоит проследить, как возникает разнообразие.

В языке C место фигурных скобок было задано Керниганом и Ритчи в их книге "The C programming language":

```
main()
{
```

```
 printf("hello, world\n");
}
```

Про расположение в книге написано: "Расположение скобок менее важно, хотя люди страстно придерживаются своего стиля. Мы выбрали один из популярных стилей. Выберите стиль, который вам подходит, и придерживайтесь его".

Те, кто читал книгу, станут располагать фигурные скобки как в книге. Это естественно и логично и 20% программистов на C-подобных языках это используют. Как располагают скобки 80% программистов? Так как они располагались в языках, на которых они изучали программирование.

Можно предположить, что стиль:

```
 if a>0 then
  a=a+1;
 end if;
```

привел к появлению следующего стиля расположения фигурных скобок:

```
 if a>0 {
  a=a+1;
 }
```

Даже Керниган и Ритчи используют этот стиль для операторов, но не для тел функций и блоков.

Также есть вариация:

```
if a>0
  {
   a=a+1;
  }
```

которую называют "стиль GNU". GNU популяризировалось Столлманом. Предполагают, что на использование Столлманом отступов перед фигурными скобками, повлиял его опыт программирования на языке LISP.

Allman

```
while (x == y)
{
    func1();
    func2();
}
```

Kernighan & Ritchie

```
while (x == y) {
    func1();
    func2();
}
```

GNU

```
while (x == y)
  {
    func1 ();
    func2 ();
  }
```

Whitesmiths

```
while (x == y)
    {
    func1();
    func2();
    }
```

Horstmann

```
while (x == y)
{   func1();
    func2();
}
```

Haskell style

```
while (x == y)
  { func1()
  ; func2()
  ;
  }
```

Ratliff style

```
while (x == y) {
    func1();
    func2();
    }
```

Lisp style

```
while (x == y)
  { func1();
    func2(); }
```

В стилях Allman и GNU блоки сбалансированы и симметричны. Несимметричные блоки сложнее для восприятия.

Происхождение языка C

В 1957 году в компании IBM был разработан язык FORTRAN, первый язык высокого уровня.

В 1958 году на недельной конференции в Цюрихе был разработан язык Algol (algorithmic language). Он многое унаследовал от FORTRAN, при этом основные понятия были собраны в более логичную структуру. В Algol появились блоки кода и добавили возможность рекурсии. Algol - процедурный, императивный, структурный язык со строгой типизацией. Через два года спецификацию языка доделали и назвали Algol-60.

В 1963 году на основе Algol-60 был создан язык CPL (Combined Programming Language), который предоставлял больше возможностей по взаимодействию с железом. Algol-60 был минималистичен и не удобен для решения сложных задач. Язык CPL был переусложнен, но через несколько лет после его появления на его основе создали язык BCPL (Basic Combined Programming Language), убрав из CPL функционал, который усложнял компиляцию (трансляцию в машинный код) программ, поэтому к названию добавили слово Basic (базовый).

Язык CPL был малопримечательным, но в 1969 году на его основе сотрудники компании AT&T Томпсон и Ритчи создали язык, назвав его "B". В языке B была семантика BCPL и синтаксис языка small Algol (подмножество языка Algol-60, предназначенное для небольших компьютеров). Другими словами, B выглядел как small Algol, а работал как BCPL. Поскольку язык B вобрал в себя из BCPL только то, что Томпсон посчитал наиболее полезным, а это примерно четверть от того, что было в BCPL, то решили из названия "BCPL" выбросить треть букв. Так из "BCPL" получилось "B".

Язык B был разработан для написания системных программ. Это был бестиповой язык, с единственным типом данных - машинным словом. В зависимости от контекста, значения этого типа обрабатывалось как целое число или адрес памяти. Языки B и BCPL были интерпретируемыми, то есть компилятор для них не создавали.

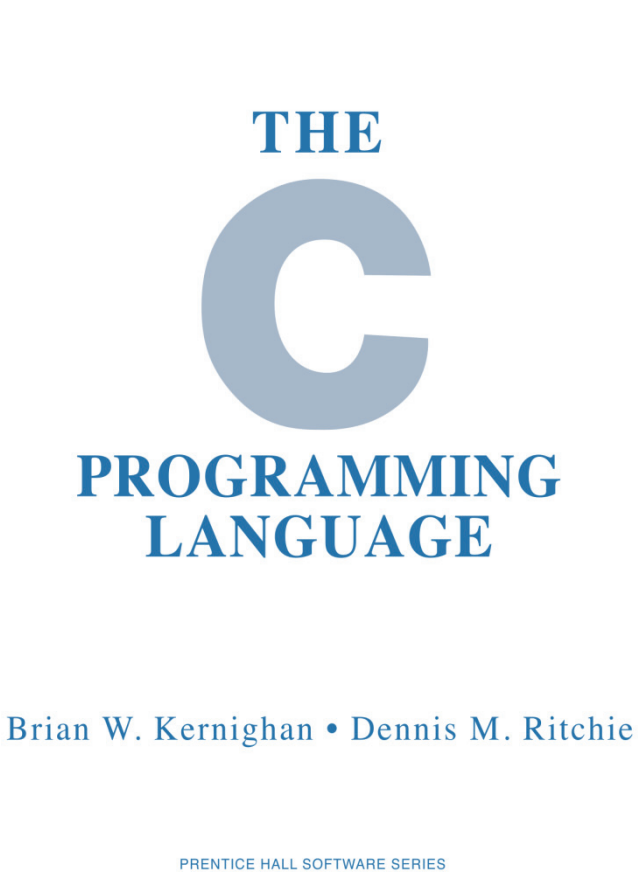
С 1969 по 1973 год Томпсон и Ритчи переписывали код операционной системы Unics (Uniplexed Information and Computing Service, переименованный в Unix) на язык B. В 1972 году была выпущена вторая версия Unix, переписанная на язык B. При переписывании язык правился. В язык были добавлены структуры - переменные, хранящие множество отдельных значений связанным (структурированным) образом, что существенно упростило программирование. Ритчи и Томпсон посчитали добавление структур, которых не было в B, small Algol, BCPL и CPL, значительным изменением, чтобы дать языку новое имя и язык B превратился в язык C. Так получился язык "C" (произносится как "Си").

Язык C

Язык C создавался несколько лет в процессе переноса на него кода операционной системы и получился удобным и простым. Язык C был первым языком, который не пытался навязывать какой-либо стиль программирования или следовать какой-то парадигме. C был первым высокоуровневым языком, предоставляющим доступ ко всем возможностям процессора, таким как "указатели" - ссылки на участки оперативной памяти и операциям побитовых сдвигов, которые стали появляться в командах процессоров.

Компилятор для нового языка C входил в поставку третей версии операционной системы Unix, вышедшей в 1973 году. В том же году вышла 4 версия Unix с ядром, полностью переписанным на язык C.

В 1975 году вышла 5 версия Unix, полностью переписанная на язык C. С 1974 года исходный код Unix на языке C распространялся среди университетов. Unix и язык C стали популярны. К 1978 году Unix была установлена более чем на 600 компьютерах. Успеха языку C добавила публикация в 1978 году книги "The C Programming Language" Кернигана и Ритчи. Книга на 228 страниц была простой и понятной. В книге не используется форма Бэкуса-Наура (BNF) для описания синтаксиса языка.



В седьмом классе Кен Томпсон заинтересовался двоичной арифметикой. Он прочёл несколько книг по математике, изучил арифметические операции в двоичной системе. У него был механический арифмометр, похожий на счёты и он создал похожее устройство для вычислений в двоичной арифметике. В выпускном классе Томпсон увлёкся электроникой и хотел собрать аналоговое вычислительное устройство, но поступив в университет Беркли он получил доступ к компьютерам и дальше занимался только ими.

Из выступления Томпсона в Национальном зале славы изобретателей:

Я учился в седьмом классе. Я сдружился с владельцем местного радиомагазина. Он позволял мне заниматься так называемой "работой" - ремонтом радиоприемников и всего, что было в моих силах, которые были не так уж велики.

Позже я захотел собрать радиоприёмник на пяти транзисторах. Первые и единственные коммерчески доступные транзисторы стоили больше пяти долларов за штуку. Это было далеко за пределами моего бюджета, но я всё равно начал копить. Владелец магазина сказал мне, что сможет найти эти транзисторы дешевле доллара за штуку. Именно столько я и накопил на эти пять транзисторов. Итак, я купил транзисторы, собрал радиоприёмник, и теперь я здесь, на этой сцене.

Мне и в голову не приходило, что мой отец и владелец магазина купили транзисторы и продали их мне ровно за ту цену, которую я мог себе позволить.

Это лишь одна история из многих, которые подтолкнули меня к тому, где я сейчас.

Игры и программирование

Игры помогают в детстве изучать мир. Когда люди впервые знакомились с компьютерами, то самое простое, что было понятно - это компьютерные игры. Игры дают побуждение к изучению компьютеров. У части людей возникает желание изучить логику игры, как она устроена, потом - написать свою игру.

На компьютерах того времени был интерпретатор или компилятор какого-нибудь языка программирования, обычно, BASIC. Начинали с игр с простой логикой и реализацией: крестики-нолики, Го, Ladder, Xonix. Однако, создание компьютерных программ даёт большее удовольствие. Научившись создавать свои программы, интерес играть проходил, как и в детстве, после того, когда достаточно изучили мир.

Из интервью Кена Томпсона "Pushing the Limits of Technology: The Ken Thompson and Dennis Ritchie Story":

Unix был создан для меня.

Я не создавал Unix как операционную систему для других людей,

Я стал создавать Unix, чтобы играть и для других своих занятий.

Я всегда увлекался играми, игры были моей стихией.

Я играл на игровых автоматах в игру "Пинбол" и вскрывал замки на задних дверях игровых автоматов. Я изучал принципиальные схемы, которые там имелись.

Именно так я и научился логике построения программ.

Игра Microsoft Flight Simulator была выпущена в 1982 году и могла использоваться даже для изучения пилотирования самолётов.

Из книги Ричарда Фейнмана "Вы, конечно, шутите, мистер Фейнман!":

А что касается мистера Френкеля, который затеял всю эту деятельность, то он начал страдать от компьютерной болезни - о ней сегодня знает каждый, кто работал с компьютерами. Это очень серьезная болезнь, и работать при ней невозможно. Беда с компьютерами состоит в том, что ты с ними играешь. Они так прекрасны, столько возможностей - если четное число, делаешь это, если нечетное, делаешь то, и очень скоро на одной-единственной машине можно делать все более и более изощренные вещи, если только ты достаточно умен.

... если вы когда-нибудь работали с компьютерами, вы понимаете, что это за болезнь - восхищение от возможности увидеть, как много можно сделать.

Игры - это этап развития людей. После того, как интерес к играм уменьшается, появляется следующий интерес - найти ошибки в программах и доказать себе, что можешь сделать лучше, то есть "взломать" систему, с чего начинали Пол Аллен и Билл Гейтс. Стив Возняк, основатель компании Apple, в юности создал цифровое устройство BlueBox, которое создавало звуки, похожие на управляющие сигналы телефонных станций. Устройство позволяло устанавливать междугородные телефонные соединения без оплаты. Стив Возняк со своим другом Стивом Джобсом развлекались, устраивая розыгрыши по телефону. Однажды Возняк дозвонился до Ватикана и, представившись Генри Киссинджером, попросил к телефону Папу. Друзья производили и продавали BlueBox, но это было рискованно и не очень законно. Они прекратили этим заниматься и создали свой компьютер и компанию Apple.

Стремление "взломать" систему приходит из детства, это стремление разобрать предмет изучения на части. В детстве разбирают машинки и игрушки, чтобы посмотреть, что там внутри. Также в детстве тестируют границы дозволенного, "бунтуют" и это тоже один из этапов развития. Хакинг - это исследование работы программ. Для языков, где программы компилируются в машинный код, например, для программ на языке C, возможна декомпиляция в ассемблерный код. По результатам исследования создают свои программы или цифровые устройства, то есть создают что-то новое. Разрушение - промежуточный этап и без перехода к созиданию человек не развивается. Стив Возняк и Стив Джобс, быстро пройдя этап "разрушения", создали компанию Apple и достигли успеха в жизни.

Игры, хакинг - способ познания мира. Те, кто задерживается на этих этапах, перестают развиваться. Например, можно разобрать компьютер или ноутбук, с целью выяснить, как они работают. Вряд ли удастся найти что-то интересное, самое интересное (программы) находится в памяти компьютера. Созидание - разобрать без разрушения, почистить-починить, может быть доработать и собрать заново.

Сборка схем из полупроводниковых деталей это также один из этапов познания мира. Кен Томпсон собрал радиоприемник, Стив Возняк телефонное устройство BlueBox. Они прошли эти этапы и перешли к учёбе в университете и написанию программ. Использование физических (материальных) устройств (конструкторов, роботов, плат микрокомпьютеров) интересно в детском возрасте. Когда развивается абстрактное мышление, физические устройства становятся необязательными, достаточно использования программных эмуляторов физических устройств. Пол Аллен и Билл Гейтс разрабатывали BASIC для компьютера Altair не имея его, разработка велась на эмуляторе процессора Intel 8080A на компьютере DEC PDP.

Системное и прикладное программирование

Системное программирование - это создание программ, обслуживающих другие программы. Пример: операционные системы; среды разработки, компиляции, сборки, запуска, выполнения программ; драйвера; утилиты; системы управления базами данных; сервера приложений, виртуализации; эмуляторы устройств.

Прикладные программы - те, которые решают бизнес-задачи и взаимодействуют с пользователями. Например, текстовые и визуальные редакторы, игры, почтовые клиенты, браузеры, бухгалтерские, складские программы.

Понятие системного программирования появилось в 1950-1960х годах как создание компиляторов, элементов операционных систем для первых компьютеров. В то время программы создавали в машинных кодах и ассемблерах. В 1960х годах стали применяться языки высокого уровня типа Algol. Для прикладных программ использовался COBOL.

Для создания операционных систем и драйверов лучше всего подходят языки программирования, обеспечивающие прямой доступ к железу на "низком" (близком к железу) уровне портов, шин ввода-вывода. Такой доступ дают ассемблеры и язык C, поскольку он был создан и совершенствовался в процессе создания операционной системы.

В системном программировании важно знание архитектуры железа; структур данных, которые близки к тем, что использует железо; эффективных алгоритмов обработки данных;

знания как работает память всех уровней. Решение задач системного программирования может быть довольно сложным и идея, как реализовать задачу, может не сразу возникнуть. К части задач приходится через какое-то время возвращаться. Например, решение задачи приостанавливается до реализации других задач, от которых зависит приостанавливаемая задача.

В прикладном программировании задачи удаётся разбивать на небольшие изолированные подзадачи, решение которых возможно за короткое время. Многие задачи шаблонны (повторяются), поэтому задачи прикладного программирования могут решаться с помощью генеративного искусственного интеллекта. В прикладном программировании можно не уделять внимание эффективности использования железа, памяти и сосредоточиться на реализации задачи наиболее простым (с точки зрения трудозатрат) путём - главное, чтобы программа работала и выдавала результат. Прикладное программирование менее трудоемкое, чем системное, потребность в создании прикладных программ более высока, порог вхождения (навыки, необходимые для эффективной работы) низкий. Решение шаблонных задач автоматизируется проще, чем сложных, их смогут решать генеративные языковые модели.

Типы данных

В 1976 году была издана книга Вирта "Algorithms + Data Structures = Programs".

Программы используют управляющие структуры для обработки данных. Данные - носитель информации. Данные обрабатываются командами (операторами).

Структура данных - их носитель, она позволяет хранить однотипные и/или логически связанные данные. Для "обработки" (добавления, поиска, изменения, удаления данных) структура данных предоставляет интерфейс взаимодействия с ней, который состоит из функций (операторов). Оператор - короткий способ вызвать функцию. Пример: "`a + b`" - выражение, в котором используется оператор сложения, имеет два операнда (аргумента, параметра). Эквивалент в виде функции: "`sum(a,b)`". Запись с оператором более компактная, так как используется меньше символов.

Для создания высокоуровневых программ, которые решают какую-то задачу по обработке данных реального мира, удобно использовать типы данных, приближенные к реальным. Например, даты, время, текстовые строки, координаты и цвет точек на экране. Поэтому высокоуровневые языки имеют набор наиболее часто использующихся в задачах, для которых предназначен язык программирования типов данных. Для удобства хранения данных языки имеют массивы и другие структуры, которые позволяют обрабатывать наборы данных как единое целое. Также языки могут позволять конструировать свои типы (классы объектов) и подключать библиотеки (модули), в которых определены типы и инструменты работы с ними: операторы и функции.

Типизация

Программы принимают, обрабатывают, выдают данные. Один из уровней классификации языков: бестиповые и типизированные. Бестиповые - это язык ассемблера и язык B, в них данные - это числа, соответствующие размерности регистров процессора: 16, 32, 64 бита. При работе программы не выполняются проверки соответствия типов, только базовые проверки, основанные на физической архитектуре. Например, при сборке программы на ассемблере будет ошибка сборки для инструкции: `mov cx, eax`; так как регистр `cx` 16-битный, а регистр `eax` 32-битный. Отсутствие проверок при выполнении программы увеличивает производительность. Недостаток бестиповых языков в том, что неудобно работать с данными, имеющими сложную структуру: строками (приходится работать с частями строк длиной в машинное слово), коллекциями. Сложные типы данных нужны для прикладных программ.

Языки с типизацией бывают: со статической/динамической типизацией, сильной/слабой, явной/неявной. Язык **C** имеет статическую, слабую, явную типизацию. Язык Java имеет статическую, сильную и преимущественно явную типизацию, хотя есть элементы неявной типизации, но их почти не используют.

Динамическая типизация означает, что проверки работы с данными выполняются на этапе выполнения, а не компиляции. Для программ, создаваемых на языках с динамической типизацией нужно создавать массу детальных тестов, но все ошибки не удаётся выявить.

Со статической типизацией проверки правильности работы с типами данных выполняется на этапе компиляции программы. На этапе выполнения проверки выполнять не нужно и программа работает быстрее. Статическая типизация упрощает написание кода, так как среды разработки, зная тип данных, могут выдавать список допустимых операций, методов работы с данными, завершать слова и выдавать подсказки. Пример объявления переменной (variable) со статической типизацией:

```
int i; // переменная i будет хранить целые числа типа int
```

Динамическая типизация:

```
var i; // переменная i будет хранить неизвестно что
```

С виду, динамическая типизация позволяет создавать "универсальный" код, обрабатывающий данные любого типа. Также создаёт видимость простоты - не надо ломать голову над выбором типа. Простота обманчива, так как усложняется поиск ошибок и тестирование. Код, обрабатывающий произвольные данные, легко создаётся в языках со строгой типизацией, таких как Java. Динамическая типизация используется в языках Python, JavaScript.

логотип языка Ada с надписью "Мы верим в строгую типизацию". Надпись и цвет - аллюзия (allusio, намёк) на надпись на долларах и их цвет

Слабая типизация позволяет выполнять неявные преобразования (приведения) типов. Благодаря этому можно не писать конструкции приведения типов. Недостаток в том, что может произойти потеря точности или приведение типа, о котором не узнает разработчик. Граница слабая/сильная типизация размыта. Например, в Java есть понятие boxing/unboxing - автоматическое приведение примитивных типов (например, int) к объектным и обратно, чтобы устранить неприятную особенность Java. В ней коллекции не могут хранить примитивные типы. Тем не менее, считается, то Java язык со строгой типизацией. Язык C относится к слаботипизированым, но на практике это не приводит к неоднозначностям. Причина в том, что в C не так много типов данных, а создание своих типов сложное и язык склоняет программистов к использованию простых типов. Язык C++ позволяет использовать множество сложных типов и при создании программ на C++ проявляются недостатки слабой типизации и создавать качественные программы сложно. Сложные типы используются в прикладном программировании. В прикладном программировании удобно писать программы в объектном стиле. Для изучения объектно-ориентированного программирования подходит язык Java, но не язык C++.

Явная типизация означает, вместе с именем переменной нужно явно указать её тип. При использовании явной типизации легко читать код. Сразу видно, какой тип может хранить переменная или какой тип вернёт функция. Пример явной типизации:

```
boolean equals(int x, int y);
```

При неявной типизации запись короче:

```
def equals(x,y);
```

В языках с явной типизацией могут быть элементы неявной. Например, в Java есть diamond operator ("<>"). Языки программирования эволюционируют и в них появляются возможности, которые хорошо зарекомендовали себя в других языках.

Модели

Модель - представление объекта, процесса в какой-либо форме (математической, физической, символической, графической или просто описательной).

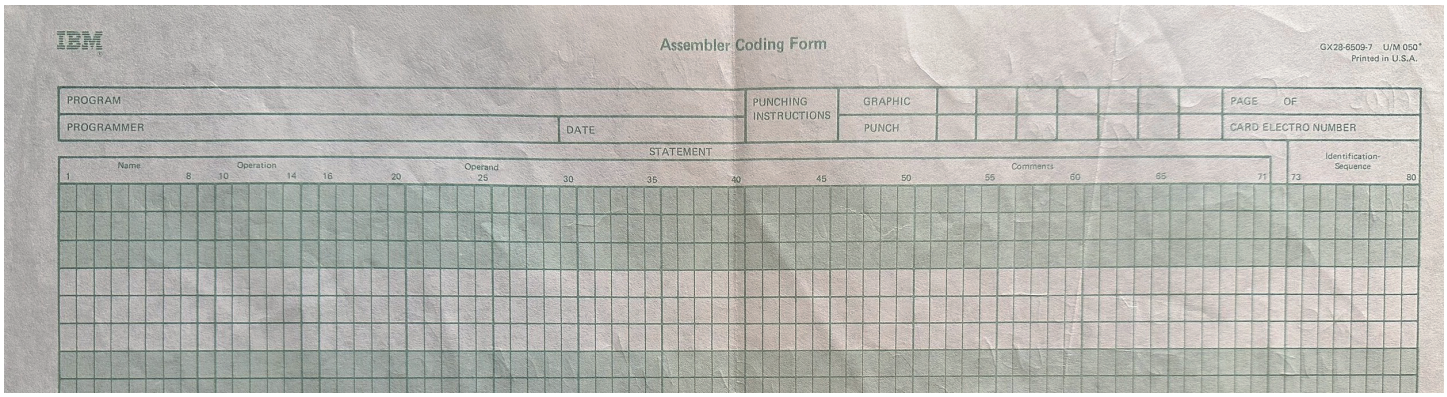
Сначала создается "эвристическая" модель - образы в воображении человека. Способность к такому моделированию зависит от фантазии, опыта, эрудиции.

Дальше для детализации могут создаваться промежуточные модели: схемы, блок-схемы, диаграммы, чертежи.

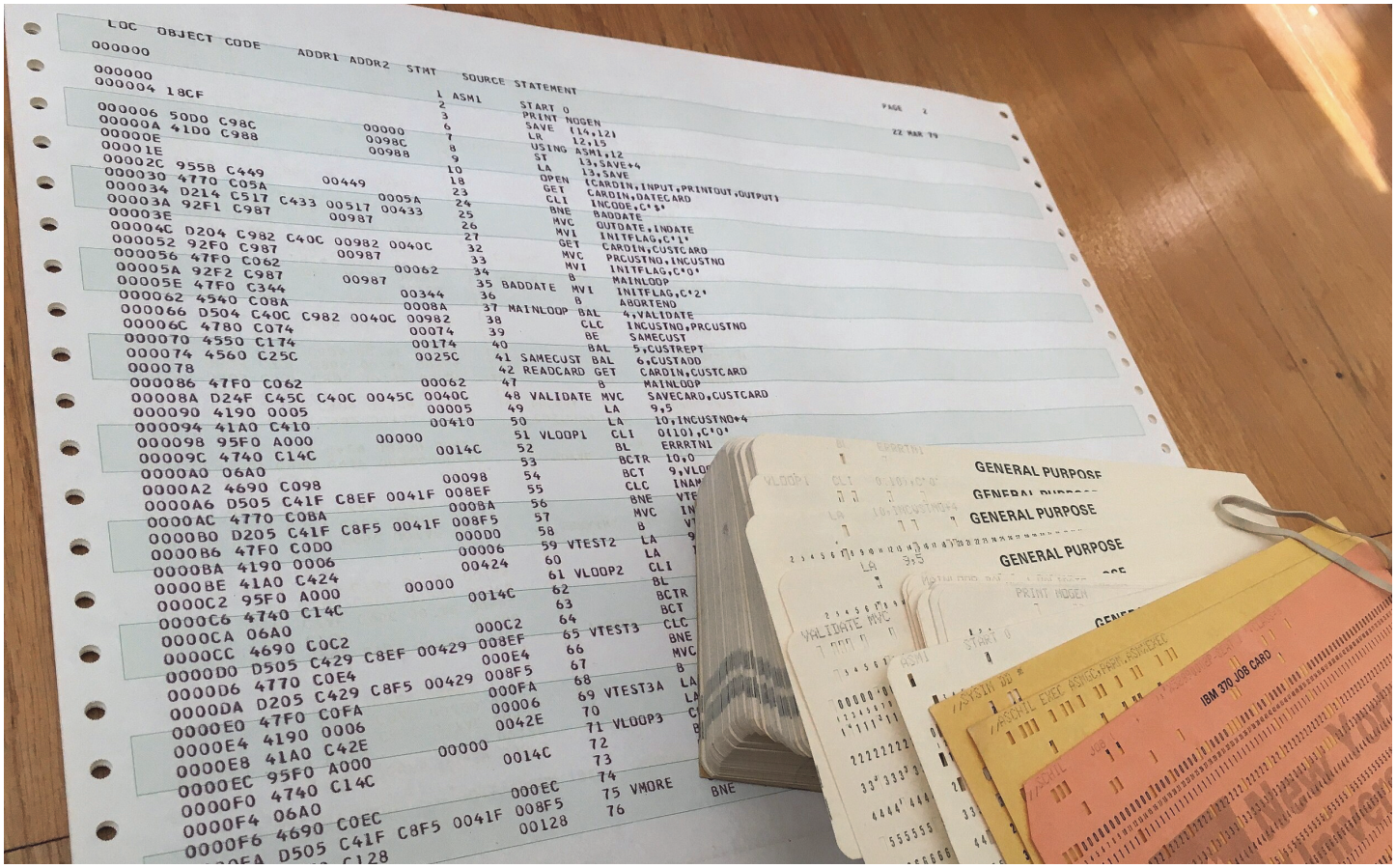
Дальше создаётся "информационная" модель - описание существенных свойств, возможных состояний объекта, процессов смены состояний, взаимосвязь с внешним миром (ввод и вывод данных), выбираются типы данных. К свойствам относятся: связи, ограничения, правила, операции.

Взаимодействие с компьютером: диалоговый и пакетный режимы

Взаимодействие с первыми компьютерами шло в "пакетном" режиме. Программы писались на листах бумаги в формате, похожем на тот, который использовала Ада Лавлейс для записи первой в истории компьютерной программы. Пример формы для записи программы на ассемблере:



Затем программа переносилась в машинные коды на перфорированные карты (перфокарты) и перфокарты отдавались на выполнение. Пример распечатки программы в машинных кодах для IBM System/360 и стопка перфокарт с программой:



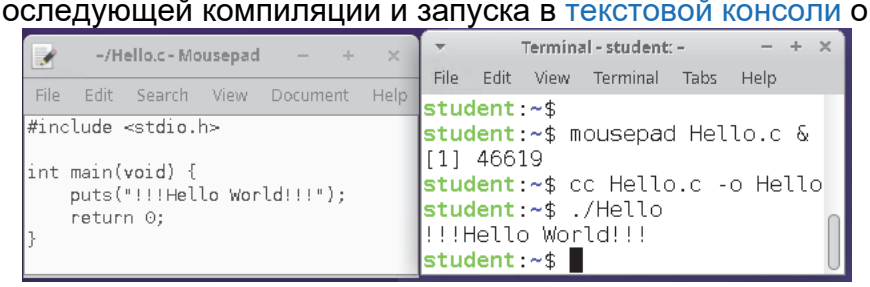
Через какое-то время выдавался результат. Если программа содержала ошибку, то приходилось заново делать перфокарты, отдавать их на выполнение и ждать результат. Причина использования пакетного режима в том, что не было операционных систем или программ, которые разграничивали бы доступ клиентов к компьютеру.

После пакетного был реализован "диалоговый" режим. Компьютеру передавали команды и получали ответ. К одному компьютеру подключали несколько терминалов и несколько человек могли совместно использовать один компьютер. Такой режим используется для общения с языковыми моделями в режиме чата: терминалом является интернет-браузер или мобильное приложение. В качестве клиента использовался телетайп (электронная пишущая машинка). Позже, вместо телетайпа стали использоваться мониторы (дисплеи) с клавиатурами, а вместо перфокарт и перфолент - дискеты. В настоящее время, вместо дискет используются флэш-накопители. Позже появились персональные компьютеры, к которым не надо было разграничивать доступ.

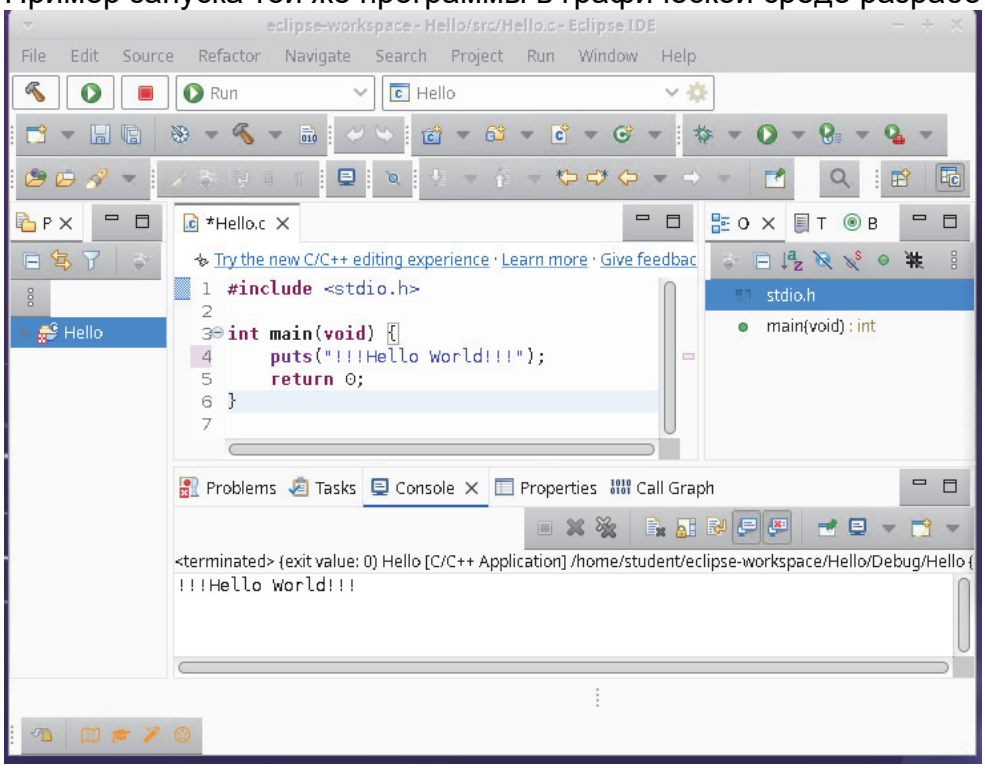
Сначала терминалы отображали только текст и печатали его построчно. Операционные системы и базы данных, в качестве стандартного клиентского приложения, до сих пор используют текстовые консоли, работающие в диалоговом режиме. Позже в командной строке появились символы "псевдографики" - линии, графические символы типа ╔╦╗ ☺, потом графический режим. Сначала в текстовом, а потом и в графическом режиме стали использовать окна, в которых отображались данные. Окна удобны тем, что легче воспринимать информацию по частям. Была создана операционная система Windows (Окна). Со временем информационное наполнение графического интерфейса стало снижаться, так как появилось большое число графических элементов, картинок, которые не имеют смыслового содержания.

При написании и тестировании программ достаточно, чтобы программа могла выводить текст, для этого достаточно консоли. Передавать программе входные данные можно устанавливая значения переменных в тексте программы. При знакомстве с программированием в книгах и обучающих курсах поначалу используют простой текстовый редактор и утилиты командной строки. Это позволяет сконцентрироваться на основном, что делается. Для этого подходит черно-белая текстовая консоль. Если человек изучает что-то новое, то объем данных, которые он может запомнить небольшой.

Пример редактирования в текстовом редакторе `mousepad` программы на языке C и её последующей компиляции и запуска в текстовой консоли операционной системы Linux:

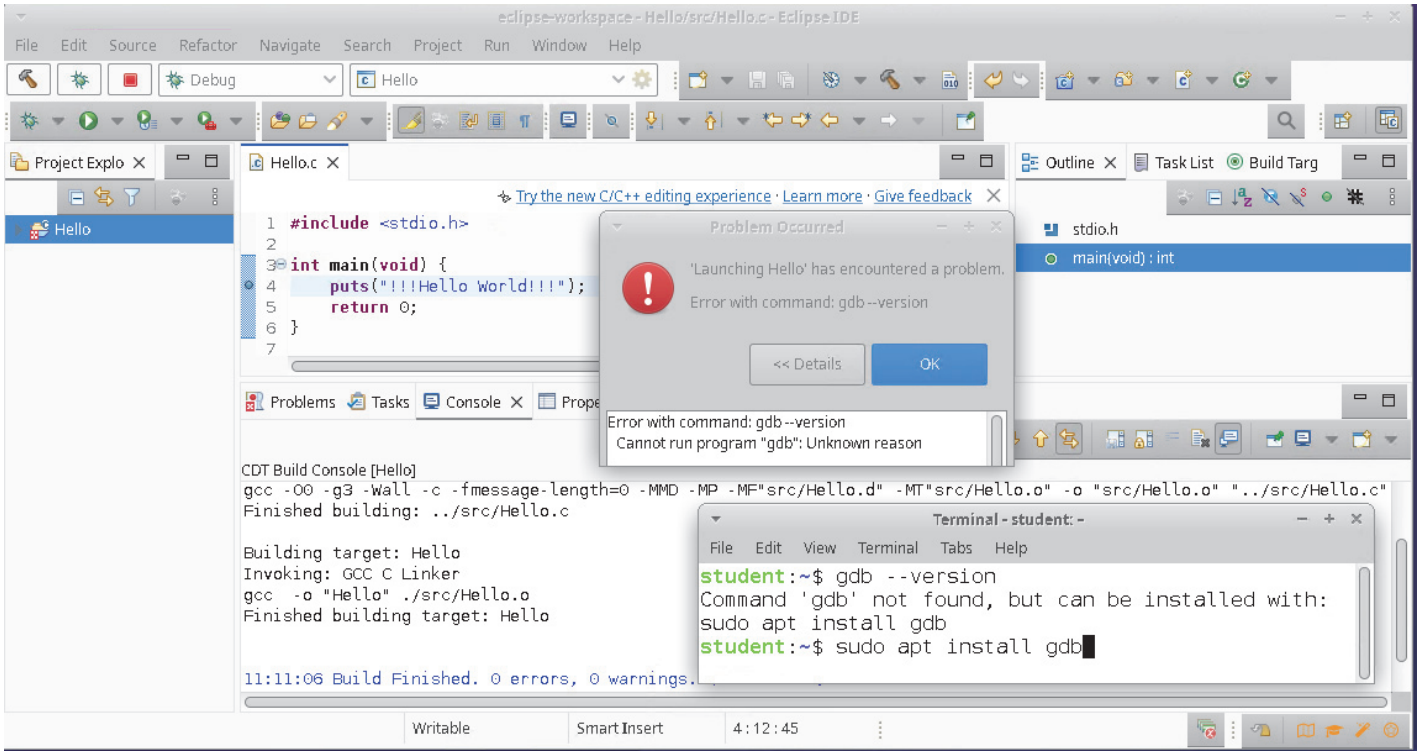


Пример запуска той же программы в графической среде разработки Eclipse:



В окне Eclipse много кнопок и меню. Если установить Eclipse и добиться компиляции выполнения простой программы, то писать программы и изучать язык будет удобнее, чем в консоли. В Eclipse можно выполнять программу по шагам, используя отладчик. Пошаговое выполнение упрощает запоминание того, как работает программа.

Настройка среды разработки, как и начало работы в любой программе, может показаться сложной. Нужно иметь наблюдательность, базовую логику и интуицию, чтобы справиться с ними. Например, хочется запустить программу в отладчике. Как это сделать? На картинке можно найти кнопку с изображением зелёного жучка. Догадаться кликнуть правой кнопкой мыши на номере строки и выбрать "Toggle breakpoint", чтобы отладчик остановился, дойдя до строки, где установлена точка остановки. Пример сложностей:



При нажатии на кнопку с изображением зелёного жучка (debugger = отладчик, устраняет bugs = жучки, баги) в Eclipse появилось окно с ошибкой "Не могу запустить программу gdb: причина неизвестна". Причина в том, что разработчики Eclipse не создали проверку - установлена ли программа-отладчик gdb перед тем, как её запустить. Проверка наличия программы перед ее запуском аналогична проверке того присвоено ли значение переменной в программе перед считыванием значения переменной. Забыть присвоить значение переменной - типичная ошибка при программировании. Eclipse создан для программистов и небольшие проблемы они должны учиться решать.

На картинке приведен пример вызова в окне терминала команды `gdb --version` на которую терминал выдал ошибку: "программа `gdb` не установлена, но вы можете установить её командой `sudo apt install gdb`". Это пример удобного для пользователей (user friendly) сообщения. После выполнения команды отладчик в Eclipse успешно запустится.

Рекурсия

Рекурсивный вызов подпрограммы - один из алгоритмов. Рекурсия - это вызов функции (процедуры) из неё же самой, непосредственно (простая рекурсия) или через другие функции (сложная или косвенная рекурсия). Рекурсия более сложна для понимания, но позволяет создавать эффективные программы.

Реализация рекурсивных вызовов функций в практически применяемых языках и средах программирования опирается на стек вызовов.

Стек можно представить себе как трубку, в которую запихиваются шарики один за другим. Шарики можно вынуть из трубки, начиная с последнего. Чтобы добраться до первого вставленного шарика, нужно вынуть все шарики, которые были вставлены за ним. Такой порядок извлечения шариков идеально подходит для рекурсивного вызова функций и возврата результатов из них.

Адрес, куда нужно вернуть управление при выходе из функции и локальные переменные функции, записываются в стек, благодаря чему каждый следующий рекурсивный вызов этой функции пользуется своим набором локальных переменных и за счёт этого работает корректно. Недостаток в том, что на каждый рекурсивный вызов требуется память и при большой глубине рекурсии в стеке может не хватить памяти.

Теоретически, любую рекурсивную функцию можно заменить циклом или ручной работой со стеком. Однако такая модификация, как правило, бессмысленна, так как приводит лишь к замене автоматического сохранения контекста в стеке вызовов на ручное выполнение тех же операций с тем же или большим расходом памяти. В языках FORTRAN, COBOL, PL/1 поначалу не было рекурсии и это было недостатком этих языков.

Пример рекурсивного вызова функции для вычисления факториала на языке C:

```
#include <stdio.h>
long factorial(long n) // определение функции
{
 if(n<=1) return 1; // проверка условия выхода из функции
 return n * factorial(n-1); // рекурсия - вызов функции внутри неё самой
}
int main() // основная функция, с которой начинается выполнение программы
{
 printf("%ld\n", factorial(20)); // печать результата и вызов функции
 return 0; // возврат статуса программы
}
```

Компиляция:

```
cc factorial.c -o factorial
```

Выполнение:

```
./factorial
2432902008176640000
```

Полученное число 64-битное (8-байтное). Хотя тип `long` гарантированно хранит только 32-битные числа, но на 64-битных процессорах вычисления 64-битные и менять название типа "`long`" на "`long long`" и формат "`%ld`" на "`%lld`" необязательно.

При вычислении факториала 21! = 51 090 942 171 709 440 000 программа выдаст неверный результат, так как он не вписывается в 64 бита. Максимальное значение для беззнакового целого 64-битого числа 9 223 372 036 854 775 807.

Функцию с рекурсивным вызовом можно заменить функцией с циклом:

```
long factorial(long n)
{
 long result=1;
 for (long i=2; i<=n; i++) result = result * i;
 return result;
}
```

Можно посмотреть, как выглядел бы код программы, если бы его писали на ассемблере. Компилятор языка C выведет ассемблерный код в файл `factorial.s` командой:

```
cc -S factorial.c
```

В созданном файле factorial.s с ассемблерным кодом будет не меньше 80 строк из них вычисление факториала около 10 инструкций (мнемоник) на ассемблере.

Пример функции, вычисляющей факториал, на ассемблере архитектуры x86. Файл fact.s:

```
.globl fact
 fact:
       movl $1, %eax    # поместить число 1 в регистр eax (eax=1)
       movl %edi, %ebx  # скопировать значение edi в ebx (ebx = edi)
   L1: cmpl $0, %ebx    # сравнить значение в ebx с нулем (ebx==0)
       je L2            # переход на L2, если сравнение истинно (goto L2)
```

```
        imul %ebx, %eax # умножение (eax = eax * ebx)
        decl %ebx       # декремент (ebx = ebx-1)
        jmp L1          # переход на L1 (goto L1)
    L2: ret             # выйти из функции (return)
```

Регистры `eax, ebx, edi` есть и на 32-битных процессорах x86 и на них, максимальное число, для которого можно вычислить факториал, это 12!=1932053504.

На 64-разрядных процессорах архитектуры x86-64 код будет скомпилирован с использованием 64-битных регистров. Можно использовать названия 64-битных регистров, заменив букву "`e`" на "`r`":

```
.globl fact
 fact:
        mov $1, %rax    # поместить число 1 в регистр rax (rax=1)
        mov %rdi, %rbx  # скопировать значение rdi в rbx (rbx = rdi)
    L1: cmp $0, %rbx    # сравнить значение в rbx с нулем (rbx==0)
        je L2           # переход на L2, если сравнение истинно (goto L2)
        imul %rbx, %rax # умножение (rax = rax * rbx)
        dec %rbx        # декремент (rbx = rbx-1)
        jmp L1          # переход на L1 (goto L1)
    L2: ret             # выйти из функции (return)
```

Программа на языке C, для вызова ассемблерной функции (файл `fact1.c`):

```
#include <stdio.h>
extern long fact(long);
int main()
{
 printf("%ld\n", fact(20));
 return 0;
}
```

С 64-битными числами можно вычислить факториал числа 20.

Зачем используется код на языке C? Программа на ассемблере, которая выводит текст на консоль, имела бы много команд, а на языке C достаточно использовать функцию `printf()`.

Компиляция программы на языке C и ассемблерного кода:

```
cc fact1.c fact.s -o fact1
```

или другим популярным компилятором:

```
clang fact1.c fact.s -o fact1
```

Выполнение скомпилированной программы:

```
./fact1
2432902008176640000
```

На компьютерах архитектуры x86 в вызываемую функцию через регистры можно передать **до двух** целочисленных параметров. Первый параметр передаётся через регистр `rdi` (`edi`), второй через регистр `esi` (`rsi`). Возвращаемое значение забирается из регистра `eax` (`rax`). На 32-битных процессорах возможно вернуть 64-битное значение, старшие разряды забираются из регистра `edx`. Если возвращать значения большего размера, то только через память, доступ к ней намного медленнее, чем к регистрам.

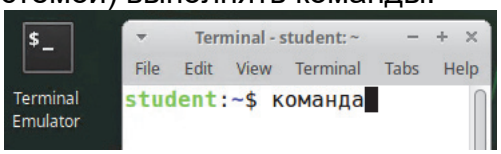
На x32-64 в функцию через регистры можно передать **до шести** параметров включительно. Параметры с третьего по шестой передаются через регистры `rdx, rcx, r8, r9`. Передача параметров через регистры чрезвычайно быстрая. Если нужно передать больше параметров или типы параметров сложные, то значения передаются через оперативную память, а это медленно.

В примере, использование ассемблера не добавило функционала по сравнению с программой **`factorial.c`**, написанной на языке C. Более того, ассемблерный код подходит

только для x86, на других процессорах он не скомпилируется и не сможет работать. Ассемблерный код полезен для изучения того, как операторы языка С преобразуются в команды процессора, как запускается программа или как она вызывает функции операционной системы.

Скриптовые языки программирования

В операционных системах есть служебная программа - текстовая консоль (терминал, командная строка, шелл), где можно интерактивно (в режиме "диалога" с операционной системой) выполнять команды.



Для выполнения команд по алгоритму - условию, или повторно (в цикле) используют сценарии (скрипты), написанные на скриптовых языках программирования. В названии языков, обычно, присутствуют слова "script" или "shell" (оболочка), например, ECMAScript (его диалекты JScript, JavaScript, ActionScript), VBScript, bash (bourn again shell).

Скриптовые языки могут использоваться для изучения базовых операторов: присвоения значения переменным "`i=1`" (переменной с названием i присваивается значение 1); арифметических `i++, i--, i+=1, i-=1, i*=1;` перехода `if, switch, return, continue;` циклов `while, do while, for, loop;` реализации простых алгоритмов. Преимущество скриптовых языков в том, что если удалось запустить консоль, поддерживающую язык, то сразу можно писать код на этом языке и этот код будет выполняться.

fork-бомба

Пример программы на скриптовом языке bash, которую можно выполнить в командной строке Linux:

```
:(){ :|:& };:
```

В этой конструкции знак ":" является названием функции. В продуманных языках программирование название функции должно начинаться с буквы английского алфавита, но bash не имеет таких правил. Заменим двоеточие на букву "f":

```
f()
 {
  f | f &
 };
f
```

Определяется функция без параметров. В теле функции (тело внутри фигурных скобок):

```
f | f &
```

вызывается сама функция. В продуманных языках программирования при вызове функции обязательно использовать круглые скобки после названия функции f(). Язык bash и большинство скриптовых языков программирования этого не требуют. Даже если это язык высокого уровня, то идея в "гибкости" - пусть программист решает, ставить круглые скобки или нет. Это не добавляет гибкости, так как не ставят круглые скобки из-за опечаток, забывчивости или потому, что скопировали откуда-то код, не вдумываясь в него (иначе бы добавили круглые скобки и сделали бы код читаемым) и этот код работает.

Перепишем тело функции с круглыми скобками:

```
f() | f() &
```

Значок "`|`" означает - результат выполнения команды слева передай команде справа от этого значка. Когда выполняется то, что слева, возникает рекурсия. Стек вызовов функции исчерпывается и процесс завершается с ошибкой. Хотя у функции нет параметров, при рекурсивном вызове функции сохраняется адрес, куда передать управление потоку

выполнения команд при выходе из функции ("адрес возврата") и на это тратится память. Этот адрес сохраняется в памяти, организованной в виде стека (накопитель по типу первым вошел, последним вышел), поэтому называется стек вызовов.

Значок "`&`" означает - не дожидайся результата выполнения того, что слева от значка "`&`", породить (**fork**) новый процесс и продолжить выполнение.

Этот код вызывает исчерпание ресурсов операционной системы, если они не были ограничены, поэтому называется бомбой. Это приводит к зависанию операционной системы или невозможности в ней работать. Менее "вредоносный" код (без порождения нового процесса значком "&"):

```
f(){ f|f;};f
```

или без значка пайпа "`|`", только с рекурсией:

```
f(){ f;};f
```

Выполнение этих программ можно остановить, закрыв терминал, в котором эти программы выполняются.

Эзотерические (удивительные) языки программирования

Эзотерическими назвали языки, созданные для исследования границ возможностей разработки языков программирования, для доказательства потенциально возможной реализации какой-то идеи, в качестве произведения программного искусства или в качестве шутки. По большей части эти языки придуманы для развлечения, часто они пародируют настоящие языки программирования или являются абсурдным воплощением концепций программирования. На практике такие языки бесполезны, однако программирование на некоторых из них является неплохой тренировкой, поэтому их часто включают в список разрешённых языков на конкурсах по программированию.

Первый из известных эзотерических языков был создан в 1972 году и назывался InterCAL. Он был создан студентами, как пародия на существующие языки программирования и как разминка для ума.

В 1993 году был создан язык FALSE для того, чтобы можно было написать компилятор для него размером не более одного килобайта и чтобы придумать синтаксис, который бы выглядел как шифровка (случайный набор символов).

Для этого в языке используются экзотические знаки пунктуации и отсутствуют пробелы. Пример реализации условия `if (a>1) b=3;` (если a больше 1, то присвоить b значение 1) на языке FALSE:

```
a;1>[3b:]?
```

Напоминает программу на скриптовом языке bash, хотя bash не относят к эзотерическим языкам.

Знак "`?`" это оператор "`if`". Знак "`:`" это присваивание. "`3b:`"означает присвоить `b` значение `3`. Знак "`=`" это сравнение. Знак "`;`" означает положить значение переменной в стек. Единичка сравнивается со значением переменной, находящейся в стеке.

Стековые языки программирования

Стековые языки программирования используют обратную постфиксную нотацию (начертание), в которой параметры команды (оператора) должны быть записаны перед самой командой (оператором), причем первый параметр расположен правее левого. Например, обычное сложение двух чисел "`1+2`" (которое наукообразно называется "инфиксная нотация") будет записано как "`1 2 +`" (обратная постфиксная нотация). В префиксной нотации сложение выглядело бы "`+1 2`" или "`+(1 2)`". Круглые скобки делают выражение понятным, но цели сделать понятнее нет. Используется правило: если число операндов фиксированное, то скобки необязательны.

Стековые языки дальше от человеческой, но ближе к машинной реализации работы с ячейками памяти (регистрам) по логике стека: запихни (логика стека) в ячейку памяти число `2`, потом возьми число `1` и прибавь к тому, что находится в ячейке памяти. В процессоре может быть один или несколько стеков с разной размерностью данных.

Постфиксную нотацию называют польской, так как её изобрел в 1920 году польский логик. Число параметров команды (операндов оператора, аргументов предиката) называют N-арностью. Например: унарный оператор - значит оператор с одним операндом. Например, "a++" называется постфиксный унарный, что означает увеличение на единицу: "a=a+1" или "a+=1". Бинарный оператор - оператор с двумя операндами. Например, оператор сложения "a+b". Тернарный оператор - с тремя операндами. Например, условный оператор z=(x>y)?a:b эквивалентен:

```
if (x>y)
 {
  z=a;
 }
else
 {
  z=b;
 }
```

Тернарный оператор возвращает значение, а условный оператор if не возвращает.

Стековые языки неудобны для написания программ человеком, поэтому не распространены. Пример стекового языка - PostScript, который используется принтерами. Пример программы на PostScript:

```
%!PS-Adobe-1.0
0 10 1 {       % цикл от 0 до 10 с шагом 1
 (Hello) show % напечатать слово Hello
} for
showpage
```

На языке C аналогичная программа выглядит так:

```
#include <stdio.h>          //добавить файл, где описана функция puts
int main(void)              //функция, с которой начнётся выполнение
{
 for(int i=0;i<=10;i++)     //цикл от 0 до 10 с шагом 1
 {
  puts("Hello");            //напечатать Hello
 }
}
```

В языке C не так много постфиксных и префиксных операторов и все они с одним операндом. Префиксные операторы были добавлены в язык C, но можно было бы обойтись без них. По большей части префиксные операторы нужны, чтобы преподаватели на уроках информатики спрашивали "чем отличается префиксный инкремент от постфиксного". Инкремент - английское слово, означающее увеличение, декремент - уменьшение, мультипликация - умножение.

Пример программы на языке C

Программа, сортирующая числа методом пузырька.
Создать файл с названием bubblesort.c

```
#include <stdio.h>
 int main()
 {
  int a[]={3,5,4,1,2};        // массив чисел для сортировки
  int n = sizeof(a)/sizeof(a[0]); // число чисел в массиве
  for(int i=0; i<n-1; i++) // сортируем
  {
   for(int j=0; j<n-i-1; j++) // сравниваем соседние элементы массива
```

```
    {
     if(a[j] > a[j+1]) // если элементы не по порядку, меняем их местами
     {
      int temp = a[j];
      a[j] = a[j+1];
      a[j+1] = temp;
     }
    }
   }
   for(int i=0; i<n; i++) // печать результата
   {
    printf("%d ", a[i]);
   }
   printf("\n"); // печать символа возврата каретки
   return 0;      // возврат статуса выполнения программы
  }
```

Скомпилировать созданную программу командой:
cc bubblesort.c -o bubblesort
Выполнить скомпилированную программу:
./bubblesort

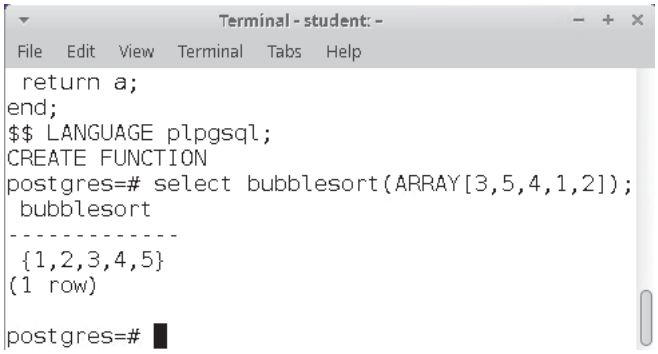
```
1 2 3 4 5
postgres@student:~$ gcc bubblesort.c -o bubblesort
postgres@student:~$ ./bubblesort
1 2 3 4 5
postgres@student:~$
```

Пример программы на языке PL/PgSQL

Посмотрим, как выглядит программа на языке семейства Ada. Язык PL/PgSQL похож на синтаксис языка Ada, только в языке Ada массивы обозначаются не квадратными, а круглыми скобками, что создаёт путаницу и затрудняет чтение кода программы: массивы в языке Ada легко спутать с функциями.

```
student:~$ psql
postgres=# CREATE OR REPLACE FUNCTION bubblesort(a int[])
RETURNS int[] AS
$$
DECLARE
 n int = array_length(a, 1);
 temp int;
begin
 for i in 1..n loop
  for j in 1..n-i loop
   if a[j] > a[j+1] then
    temp = a[j];
    a[j] = a[j+1];
    a[j+1] = temp;
   end if;
  end loop;
 end loop;
 return a;
end;
$$ LANGUAGE plpgsql;
```

Выполнение созданной функции командой `select bubblesort(ARRAY[3,5,4,1,2]);`



Для создания электрических схем используются языки описания аппаратуры: Verilog - на основе языка C и Verilog - на основе языка Ada. Программы на этих языках описывают действия по обработке данных. Данные - электрические сигналы, действия - операции типа `NOR, AND, XOR`, которые реализуются элементами электрических схем. Из текста программы генерируется пространственная схема из электрических компонентов, токопроводящих дорожек между ними. Схемы слоёв преобразуются в картинки, из которых можно создать фотошаблоны для вытравливания дорожек на кремниевой пластине методом ультрафиолетовой литографии. Фотошаблоны используются, так как у света небольшая длина волны, а значит высокое разрешение. В настоящее время процессоры могут производиться с разрешением 2-5 нанометров. Процессор Apple A16 создаётся с разрешением 4 нанометра. Микросхемы для массового рынка используют процессы с худшим разрешением, так как их производство дешевле. Процессоры ESP32 используют разрешение 40 нанометров и при этом имеют низкое энергопотребление. Размеры транзисторов больше разрешения, которое определяет минимальный размер соединений, а не размер транзисторов. Поэтому, при улучшении разрешения меняют пространственную компоновку транзисторов. Названия компоновок: VTFET, GAAFET, FinFET. В GAAFET (Gate-all-around) затвор транзистора полностью окружает канал, а в FinFET частично.

Что дальше?

Для изучения работы в Linux и программирования может быть полезна книга "Азы программирования", которую можно найти на сайте http://stolyarov.info/

Для тех, кто хочет изучить архитектуру компьютеров и познакомиться с ассемблером https://teach-in.ru/course/architecture-and-assembler/about

Если на компьютере не установлен Linux, то можно установить Virtualbox https://www.virtualbox.org/wiki/Downloads , скачать образ виртуальной машины в формате open virtual appliance (ova, конфигурации) https://disk.360.yandex.ru/d/eUu522ezEGuXvA , в которой можно сразу писать программы на языке C, компилировать и выполнять. Также удобно установить в виртуальную машину среду разработки Eclipse IDE for C/C++ Developers https://www.eclipse.org/downloads/packages/ . Возможно, вы справитесь с установкой.