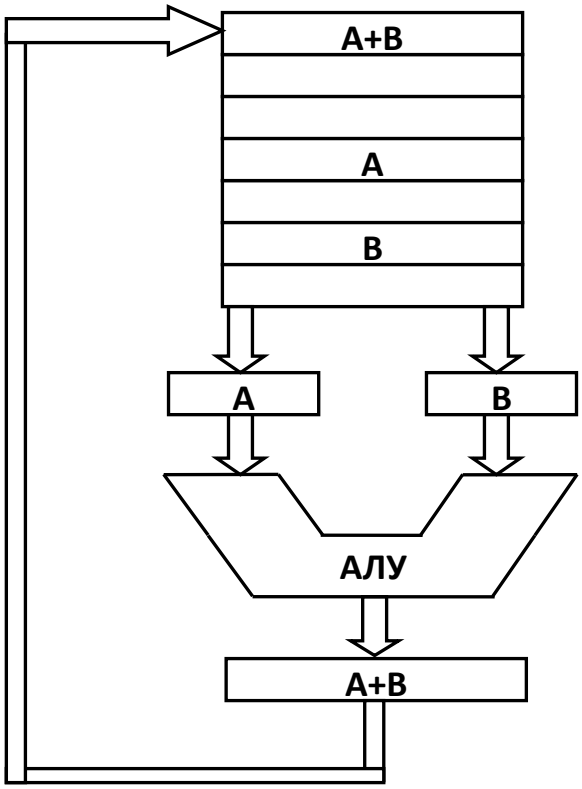
УНИВЕРСИТЕТ ИТМО

А.В. Павлов

АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ



Санкт-Петербург
2016

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

А.В. Павлов

АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Учебное пособие

Санкт-Петербург

2016

А.В.Павлов, Архитектура вычислительных систем – СПб: Университет ИТМО, 2016. – 86 с.

В пособии представлены методические материалы по курсу «Архитектура вычислительных систем». Изложены базовые сведения и освещен круг вопросов, связанных с построением и функционирование вычислительных устройств и систем. Уделено внимание перспективным идеям и технологиям построения вычислителей.

Для бакалавров, обучающихся по направлению подготовки 12.03.03 «Фотоника и оптоинформатика» по профилям подготовки: "Оптические и квантовые технологии передачи, записи и обработки информации", "Наноматериалы и нанотехнологии фотоники и опто-информатики", "Физика наноструктур".

Рекомендовано к печати Ученым советом факультета Фотоники и оптоинформатики, протокол № 6 от 25 мая 2016 г.

Университет ИТМО – ведущий вуз России в области информационных и фотонных технологий, один из немногих российских вузов, получивших в 2009 году статус национального исследовательского университета. С 2013 года Университет ИТМО – участник программы повышения конкурентоспособности российских университетов среди ведущих мировых научно-образовательных центров, известной как проект «5 в 100». Цель Университета ИТМО – становление исследовательского университета мирового уровня, предпринимательского по типу, ориентированного на интернационализацию всех направлений деятельности.

ОГЛАВЛЕНИЕ

Список обозначений и сокращений

АЛУ	Арифметико-логическое устройство
ГА	Гарвардская архитектура
ИР	Индексный регистр
КОП	Код операции
ОЗУ	Оперативное запоминающее устройство
ОП	Оперативная память
ПА	Принстонская архитектура
ПЗУ	Постоянное запоминающее устройство
РА	Регистр адреса
РК	Регистр команд
СОЗУ	Сверхоперативное запоминающее устройство
РОН	Регистр общего назначения
РПР	Регистр признака результата
СК	Счетчик команд (указатель инструкций IP)
УГО	Условное графическое обозначение
УкС	Указатель стека
УУ	Устройство управления
ЦК	Цифровой компьютер
ЦП	Центральный процессор
ЭВМ	Электронно-вычислительная машина

1. Введение

Наш курс называется «Архитектура вычислительных систем», т.е. включает в себя три термина.

К понятию **системы** мы подойдем в процессе изучения курса.

Слово «**архитектура**», как и очень многое в нашей культуре, греческого происхождения, образовано из двух: **αρχι** — старший, главный и **τέκτων** — строитель. Применительно к предмету нашего изучения архитектура большинством специалистов понимается в смысле совокупности общих принципов организации аппаратных и программных средств, определяющей функциональные возможности вычислителей при решении соответствующих классов задач. К термину «архитектура» близко примыкает термин «функциональная организация», смысл этих терминов, их специфику и различия мы более подробно обсудим отдельно.

Корень слова **вычислитель** – число. Понятие числа – тема отдельного и очень глубокого изучения. Изучения не только сугубо математического, но и философского. Если пифагорейцы, например, не различали числа и вещи, то Платон ввел понятие абстрактного числа, отличного от вещей. В рамках нашего курса мы ограничимся следующими двумя определениями:

Число – одно из основных понятий математики, служащее для количественной характеристики различных предметов и явлений, как принадлежащих реальности, так и абстрактных. [1]

Число – абстрактное, лишенное особенного содержания обозначение какого-либо члена некоторого ряда, в котором этому члену предшествует или следует за ним какой-нибудь другой определенный член; абстрактный индивидуальный признак, отличающий одно множество от другого того же рода. [2]

В раскрытие этого определения в математическом плане упомянем систему аксиом Пеано, предложенную им в XIX веке для натурального числового ряда:

Множеством натуральных чисел $\mathbb{N}$ будем называть множество, в котором зафиксирована функция следования $S:\mathbb{N}\to\mathbb{N}$, и для которого выполнены следующие аксиомы:

1. $1\in\mathbb{N}$ (элемент, называемый единицей, является натуральным числом);
2. Если $x\in\mathbb{N}$, то $S(x)\in\mathbb{N}$. (правило следования – число, следующее за натуральным, также является натуральным числом. Оно называется последователем (successor) и обозначается $S(x)=x'$;
3. $1\neq x'$ (единица не следует ни за каким натуральным числом);
4. Если $x'=y'$, то $x=y$. (числа, имеющие одинаковые последователи, равны).

Другая формулировка этой аксиомы: Если $S(b)=a$ и $S(c)=a$, то $b=c$.;

5. Аксиома индукции. Если Q есть свойство, которым, может быть, обладают одни и не обладают другие натуральные числа, и если: а) натуральное число 1 обладает свойством Q и б) для любого натурального числа x из того, что x обладает свойством Q следует, что и его последователь x' также обладает свойством Q, следует, что свойством Q обладают все натуральные числа.

Операции над числами – вычисления. Вычислитель – устройство, реализующее вычисления. Следовательно, вычислитель вторичен относительно системы счисления и его архитектура определяется реализуемой системой счисления.

Определение 1. **Система счисления** – совокупность алфавита (системы цифровых знаков) и правил их записи (цифрового синтаксиса), применяемых для их записи и чтения.

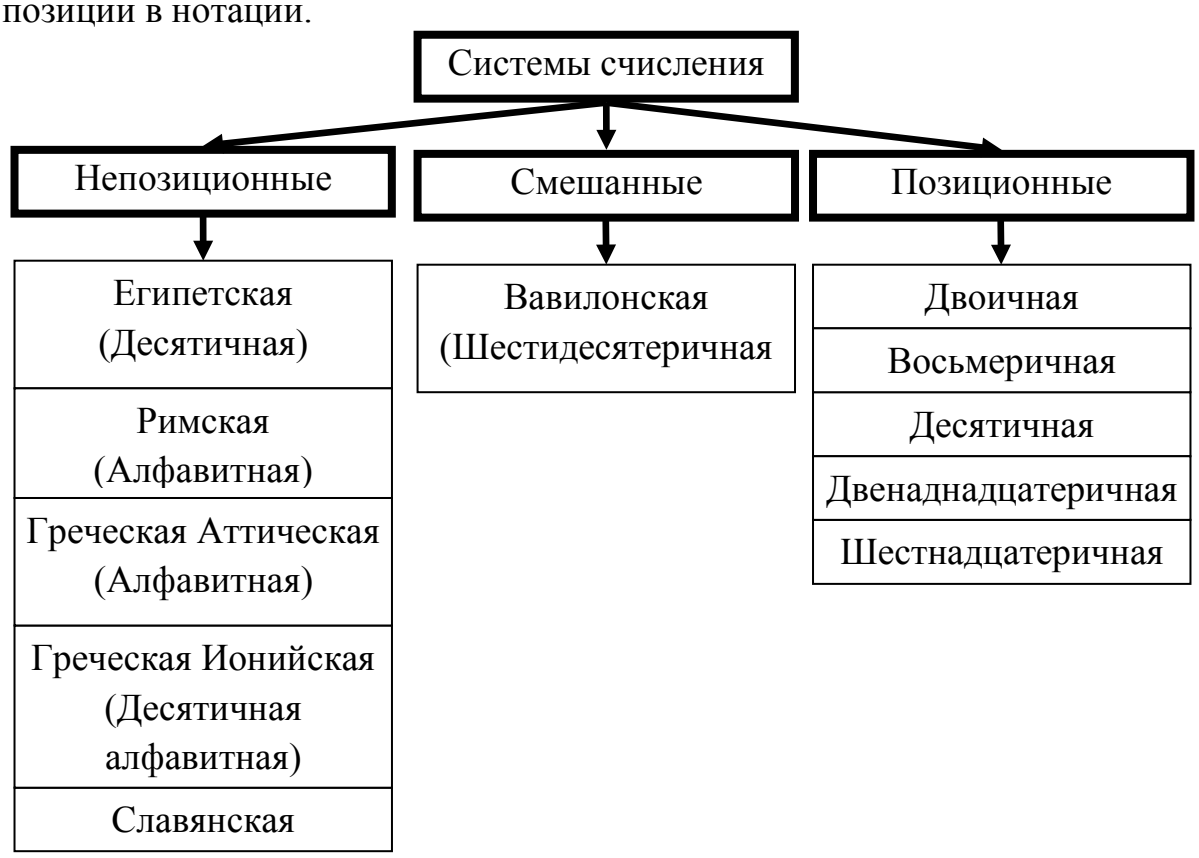
Система счисления должна:

1. давать представления о множестве чисел;
2. обеспечить уникальное представление каждого числа;
3. отражать арифметическую структуру.

Классификация систем счисления.

В позиционных системах счисления значение цифры определяется её разрядом (позицией в записи числа).

В непозиционных системах счисления значение цифры не зависит от её позиции в нотации.



Важнейшее достоинство позиционных систем счисления относительно непозиционных – простота выполнения арифметических операций.

Слабое место позиционных систем – наличие межразрядных связей (переносов и заемов) при выполнении арифметических операций над числами, то есть невозможность выполнения арифметических операций как поразрядных (когда результат операции в каждом разряде не зависит от ее результата в остальных разрядах).

Основная заслуга в переходе к десятичной позиционной системе счисления принадлежит древнеиндийским математикам, творчески переработавшим и объединившим в этой системе достижения индийской мультипликативной системы, вавилонской (идея специального символа для обозначения нулевого значения разряда) и греческой – символ «0» от буквы О (омикрон), вероятно – первой буквы греческого слова «ничто».

На сегодня в цифровой вычислительной технике преимущественно используются следующие системы:

1. двоичная;
2. восьмеричная;
3. шестнадцатеричная;
4. двоично-десятичная.

Двоичная система естественна для систем, использующих для реализации памяти и арифметических операций бинарные элементы, т.е. элементы с двумя устойчивыми состояниями.

Восьмеричная и шестнадцатеричная системы применяются исходя из соображений снижения трудоемкости ручной обработки машинных кодов.

Восьмеричная система включает восемь цифр: 0, 1, 2, 3, 4, 5, 6, 7. Поскольку основания двоичной и восьмеричной систем связаны простым выражением $8 = 2^3$, то для перевода числа из двоичной записи в восьмеричную первую надо разбить на триады – группы по три цифры. Каждая триада двоичной нотации соответствует одной цифре в восьмеричной системе.

Шестнадцатеричная система включает десять цифр и шесть прописных латинских букв: A, B, C, D, E, F. Правило перевода из двоичной в шестнадцатеричную систему аналогично предыдущему с тем отличием, что поскольку $16 = 2^4$, то двоичная нотация делится на тетрады – группы по четыре цифры.

Ниже дан пример перевода двоичного числа в шестнадцатеричное (верхняя строка) и восьмеричное – нижняя строка:

$$\overbrace{0011111000001100}^{3\quad E\quad 0\quad C}$$

$$\underbrace{0}_{0}\underbrace{011}_{3}\underbrace{111}_{7}\underbrace{000}_{0}\underbrace{001}_{1}\underbrace{100}_{4}.$$

В двоично-десятичной системе десятичные цифры представляются 4-х разрядными двоичными комбинациями от 0000 до 1001 – двоичными эквивалентами первых десяти цифр шестнадцатеричной системы. Этот

способ обеспечивает простое выполнение преобразования из двоично-десятичной системы в двоичную, равно как, и обратного. Для такого преобразования достаточно заменить четыре двоичные цифры одной десятичной. Две двоично-десятичные цифры составляют один байт. В этой системе один байт позволяет представить числа от 0 до 99 (восьмиразрядное двоичное число позволяет представить числа от 0 до 255).

Для примера: возьмем число 1001 0101 0011 1000.

Если это число двоичное, то его десятичный эквивалент

$(1001010100111000)_2 = (38200)_{10}$

а если как двоично-десятичное, то

$(1001010100111000)_{2-10} = (9538)_{10}$.

Сложение двоично-десятичных чисел, имеющих один десятичный разряд, аналогично сложению 4-х разрядных двоичных чисел. Но если результат сложения превосходит 1001, то необходимо добавить двоичный код числа 6, т.е. 0110. Эта процедура называется коррекцией, пример дан ниже.

$$\begin{array}{r} 4 \\ +\ 5 \\ \hline 9 \end{array} \qquad \begin{array}{r} 0100 \\ +\ 0101 \\ \hline 1001 \end{array} \qquad \begin{array}{r} 5 \\ +\ 9 \\ \hline 14 \end{array} \qquad \begin{array}{r} 0101 \\ +\ 1001 \\ \hline 1110 \\ +\ 0110 \\ \hline 1\,0100 \end{array}$$

В рамках нашего курса термины «компьютер», «вычислитель» и «ЭВМ» будем понимать как равноправные и равнозначные. Обратим внимание, что эти термины не определяют способ представления и обработки данных – цифровой или аналоговый. Поэтому в ряде случаев будем использовать термин «цифровой компьютер», сокращенно ЦК, а там, где из контекста ясно, о каком типе компьютера идет речь, любой из вышеприведенных терминов без специальных оговорок.

Понятие информации

В названии нашего курса присутствует термин "вычислительные". Но в настоящее время значение вычислителей или, как их сейчас чаще называют – компьютеров, вышло за рамки собственно вычислений в узком смысле. Сегодня компьютеры рассматриваются, прежде всего, как важнейшие элементы систем обработки информации. Поэтому необходимо вкратце рассмотреть понятия информации и связи этого понятия с вычислениями.

Наше время часто называют информационной эпохой, веком информации. Термин "информация" встречается повсеместно, употребляется всеми, кому не лень. Но что означает этот термин?

На сегодня существует понимание информации как одного из трех фундаментальных атрибутов нашего материального мира. Но любая дискуссия и любая наука начинаются с договоренности о терминах и понятиях. Следовательно, нам надо начать с определения понятия информации.

Определение информации

В литературе можно найти множество самых разных определений информации. Например, такие:

- **Информация** – от лат. informatio, разъяснение, изложение, осведомлённость. Для обыденных целей достаточно, но достаточно ли для работы с информацией как одним из трех "слонов", на которых стоит наш мир? Например, отвечает ли это определение на вопрос о том, откуда берется информация? Или о закономерностях её преобразования?
- **Информация** есть сведения, сообщения…
- **Информация** есть мера упорядоченности… . Уже лучше, как мы увидим далее, здесь отражен очень важный аспект, но вопросы все еще остаются.
- **Информация** есть содержание процессов отражения…
- **Информация** есть алгоритм или инструкция… . А как быть с неформализуемыми и неалгоритмизируемыми задачами, т.е. именно теми задачами, решение которых, относящееся к категории творчества, и представляет наибольший интерес? Ведь ни алгоритм, ни инструкция ничего нового не создают - они лишь повторяют ранее существующее. А человек наделен способностью к творчеству, т.е. к созданию нового, ранее не существовавшего.
- «**Информация** есть информация, а не материя и не энергия. Это третье». (Норберт Винер) С этим можно согласиться, но какова ценность такого определения? Что из него можно конструктивного извлечь, как применить?
- «**Информация** есть там, где есть разнообразие …». (Р. Эшби) Здесь отражен ключевой, фундаментальный момент – необходимость разнообразия, без которого невозможно появление информации. Обратим внимание, что этот пункт резко контрастирует с основным трендом общественной жизни – унификации и стандартизации.
- «**Информация** есть там, где есть неоднородность распределения энергии …» (акад. В.М.Глушков).
- «**Информация** появляется лишь тогда, когда начинается изучение материального мира с целеполаганием» (акад. Н.Н.Моисеев) – здесь мы видим грандиозный прорыв – отказ от традиционной Европейской научной парадигмы, предполагающей объективность всех процессов в нашем мире. В научную картину мира вводится познающий субъект как важнейший, определяющий фактор. Без субъекта нет информации, информация субъективна!

- Часто используется т.н. "энтропийный подход" к определению информации. Этот подход мы рассмотрим чуть ниже, при обсуждении количественной оценки информации, сейчас же отметим лишь, что он относится к вопросу измерения количества информации, но никак не к её определению.

Рассмотрим теперь определение, предложенное Генри Кастлером:

- «**Информация** есть запомненный выбор одного варианта из нескольких возможных и равноправных» [3].

Достоинство этого определения – оно отвечает на вопрос как о механизме возникновении информации – механизме выбора, так и необходимых для этого условиях:

1. "внешнее" или объективное условие – наличие альтернатив;
2. и "внутреннее"или субъективное – наличие свободы выбора и памяти.

Обратим особое внимание, что это определение, также как и определение акад. Н.Н.Моисеева, акцентирует внимание на необходимости наличия творца информации или автора – того, кто осуществляет выбор одного варианта. Иными словами, информация не возникает сама по себе, ниоткуда – она всегда кем-то создается, у неё всегда есть автор.

Это определение было развито Ириной Вигеновной Мелик-Гайказян:

«**Феномен информации** есть многостадийный, необратимый процесс становления структуры в открытой неравновесной системе, начинающийся со случайного запомненного выбора, который эта система делает, переходя от хаоса к порядку, и завершающийся целенаправленным действием согласно алгоритму или программе, отвечающим семантике выбора » [4].

Рассмотрим ключевые моменты этого определения:

1. **Необратимость процесса.** Следуя [5] будем понимать систему логически необратимой, если по сигналу на выходе нельзя однозначно определить сигнал на входе. Необратимость обусловлена открытостью системы [6] – наличием в ней диссипативного фактора [7, 8].

2. **Становление структуры**. Внутренняя структурированность, внутренняя связность элементов – важнейший атрибут, отличающий информацию от шума. Мера внутренней структурированности – длина корреляции. [9-11]

3. **Открытость и неравновесность системы**. Порождение информации возможно только в открытой системе, т.е. системе, имеющей приток и сток, обменивающейся со своим окружением. Неравновесность – важнейшее условие, в равновесной систем ничего не происходит. [6, 12]

4. **Случайность**. Если выбор не случаен, то он чем-то мотивирован, какой-то подсказкой, т.е. какой-то информацией. Следовательно, в полном смысле порождения информации нет, есть лишь её рецепция (восприятие) и преобразование.

5. **Запомненность выбора**. Если результат выбора не запомнен, то … о чем тогда можно говорить?

6. **Переход от хаоса к порядку**. Здесь подчеркнута важнейшая т.н. творческая роль хаоса - без хаоса нет творчества. Способность к творчеству у человека неразрывно связана со способностью к "погружению в хаос" – переходу к хаотической динамике нейронной активности коры головного мозга. [13-16]

Рассмотрим вопрос количественной оценки информации

В настоящее время популярен т.н. "энтропийный подход" к оценке количества информации. Основы этого подхода заложены в классических работах Ральфа Хартли [17] и Клода Шеннона [18]. Хартли в 1928 г. предположил, что информация допускает количественную оценку. Завершенный и полный вид этой теории придал в 1948 году Клод Шеннон.

Р. Хартли процесс получения информации рассматривал как выбор одного сообщения из конечного наперёд заданного множества из N равновероятных сообщений, а количество информации I, содержащееся в выбранном сообщении, определял как двоичный логарифм N. т.е.:

$I = \log_2 N$.

Допустим, нужно угадать одно число из набора чисел от 1 до 8 (Рис.1). По формуле Хартли можно вычислить, какое количество информации для этого требуется: $I = \log_2 8 = 3$. Таким образом, сообщение о верно угаданном числе содержит количество информации, равное 3 единицам информации.

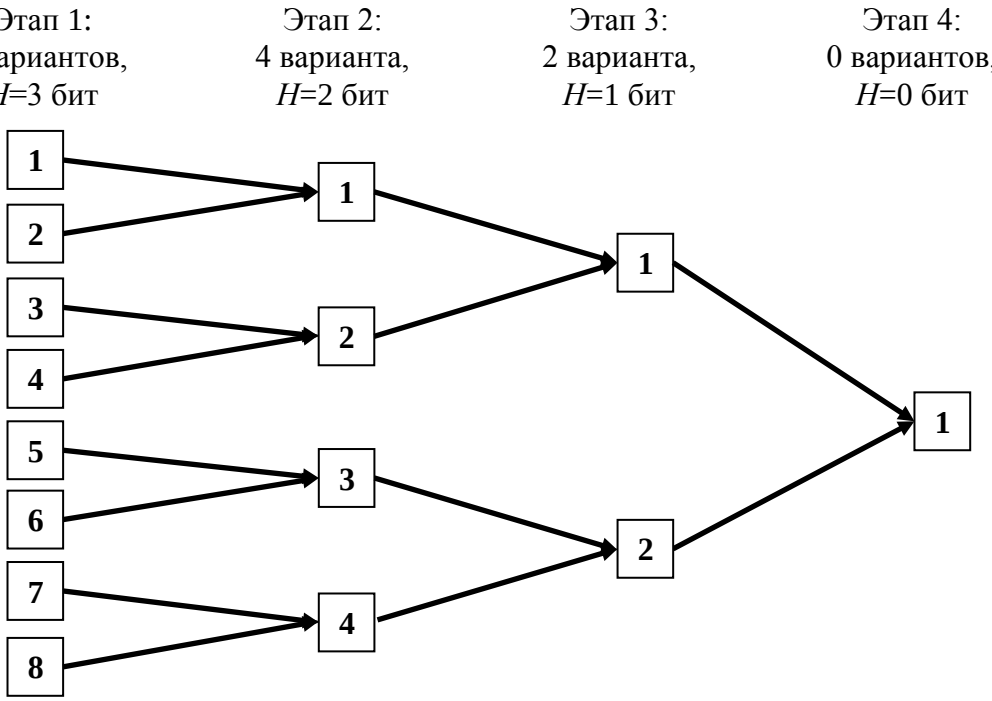


Рис.1. Иллюстрация подхода Хартли – уменьшение неопределенности путем последовательного принятия решений по выбору варианта

Обратим внимание, что здесь речь идет о выборе одного из сообщений. Т.е. рассматривается не вопрос порождения информации, а вопрос передачи и приема сообщений – уже существующей информации. Это важный момент, который часто упускается из виду.

Но данный процесс получения информации (сообщений) рассматривает равновероятное число конечных элементов N. Например, являются ли равновероятными сообщения "первой выйдет женщина" и "первым выйдет мужчина"? Понятно, что в общем случае однозначно ответить на этот вопрос нельзя. Все зависит от того, откуда выйдет.

Для задач такого рода Клод Шеннон предложил в 1948 г. [18] другую формулу определения количества информации, учитывающую возможную неодинаковую вероятность сообщений в наборе.

$$I = -\sum_{i}^{n} p_i \log_2 p_i , \qquad (1.1)$$

где p_i - вероятность появления i - ого сообщения, n - общее число сообщений.

Если все варианты равновероятны, то есть $p_i = 1/n$, то мы получаем классическую формулу Хартли.

Отметим, что Шеннон под информацией понимал **сообщение**, уменьшающее неопределенность (энтропию) у получателя сообщения. Информация – снятая неопределенность. Точнее, получение информации – необходимое условие для снятия неопределенности. Неопределенность возникает при выборе. Неопределённость снимается уменьшением количества рассматриваемых вариантов (уменьшением разнообразия) – выбором одного соответствующего ситуации варианта из возможных, что даёт возможность принимать обоснованные решения и действовать. В этом заключается управляющая роль информации. Формула Шеннона отражает количество полученной информации, но не её ценность.

Для того, чтобы наглядно увидеть разницу между приемом сообщений, рассматриваемым Хартли и Шенноном, и информацией как феноменом, приведем простой пример – количество информации в сообщении, определяемое формулой Шеннона, не зависит от того или иного сочетания букв: можно сделать сообщение бессмысленным, просто переставив буквы. В этом случае ценность информации исчезнет, она превратится в шум, но количество информации, определяемой формулой Шеннона, формально останется прежним. Например, энтропия слова «пирожок», имеющего всем понятный и приятный смысл, согласно (1.1)

$$I_{ПИРОЖОК} = -N\sum_{i=0}^{M} p_i \log_2 p_i = 9.656 ,$$

а для бессмысленного «жиропок» та же формула (1.1) дает энтропию

$$I_{ЖИРОПОК} = -N\sum_{i=0}^{M} p_i \log_2 p_i = 9.656 ,$$

т.е. такую же оценку информационной емкости, как и для понятного слова.

Таким образом, энтропийный подход, применимый к теории передачи сообщений по линиям связи (теория связи), не учитывает различие между информацией и шумом.

Корреляционная мера информации. Альтернативой энтропийного подхода к оценке количества информации может служить подход с позиции корреляционной меры. Ценность этого подхода в том, что он учитывает такой важнейший атрибут информации, как её внутреннюю структурированность, т.е. связность её элементов. Подход, как следует из названия, основан на использовании функции корреляции.

Функция взаимной корреляции двух функций $f_1(x)$ и $f_2(x)$ определяется следующим образом:

$$C_{f_1 f_2}(\Delta) = f_1(x) \otimes f_2(x) = \int_{-\infty}^{+\infty} f_1(x+\Delta) f_2^*(x) dx . \qquad (1.2)$$

Если $f_1(x) = f_2(x)$, то функция корреляции называется функцией автокорреляции.

Аргументом функция корреляции является Δ сдвиг первой функции $f_1(x)$ относительно второй $f_2(x)$. Если $f_1(x) = f_2(x)$, то при $\Delta = 0$ имеем максимальное значение функции корреляции. По мере сдвига одной функции относительно другой значение функции корреляции убывает относительно максимального значения. Скорость этого убывания показывает взаимную связь значений функций в точках, отстоящих друг от друга на величину сдвига. Если эти значения никак друг с другом не связаны (не коррелированны), то функция корреляции равна нулю.

Таким образом, скорость убывания автокорреляционной функции может служить мерой внутренней связности функции, её структурированности. В качестве такой оценки обычно используют параметр, называемый длиной корреляции l_C или радиусом корреляции r_C – полуширину глобального максимума функции автокорреляции по заданному уровню от максимального значения. Если рассматривать $f_1(x)$ как некоторый фрагмент информации длиной $2L$, то оценки информационной емкости $f_1(x)$ можно использовать характеристику, называемую обобщенной частотой

$$\Omega = L / l_C . \qquad (1.3)$$

Микро- и макро-информация

Ранее мы упоминали важность запоминания. В этой связи имеет смысл разграничить микро- и макро-информацию.

- **Макроиформация** – **запомненный** выбор одного варианта из нескольких возможных; *запоминание* означает, что сделанный выбор

сохраняется в течение времени, которое больше, чем характерное время использования данной информации.

- **Микроинформация** – **незапомненный** выбор одного варианта из нескольких возможных. Это означает, что данный выбор существует в течение не более пикосекунд, а далее забывается.

Граница между макро- и микроинформацией определяется способностью запоминания на длительное время.

Запоминание – приведение системы в устойчивое состояние. Этой способностью обладают только макросистемы – системы, состоящие из многих атомов (макромолекулы (10^{-9} м)). Один атом запомнить ничего не может – у него только одно устойчивое состояние.

Термодинамика работает с идеальным газом, состояние может быть запомнено только на время $t \sim 10^{-13}$ с. Поэтому энтропийная оценка количества информации (1.1) правомочна как оценка не реальной информации, а как оценка микроинформационной емкости или «емкости информационной тары» [9]. Корреляционная мера (1.3), в отличие от энтропийной, учитывает внутреннюю связность, т.е. оценивает не микро-, а макро-информацию.

Ценность информации

- Ценность информации зависит от цели, которую преследует рецептор (получатель информации);
- Чем в большей мере информация помогает достижению цели, тем более ценной она считается [7].

Количественное определение ценности информации

1. Если цель наверняка достижима и притом несколькими путями, то возможно определение ценности (V) по уменьшению материальных или временных затрат, благодаря использованию информации [19].

2. Если условие достижения цели не является обязательным, то могут быть использованы, например, следующие количественные оценки ценности информации:

а) $V = \log_2 \frac{P}{p};\ -\infty < V < V_{\min}$

согласно [20],[23] , и

б) $V = \frac{P - p}{1 - p};\ 0 < V < 1$

согласно [10] , где P – вероятность достижения цели при наличии информации, p – вероятность достижения цели при отсутствии информации.

Ценность информации можно понимать, как количество ценной информации.

Важно помнить, что **ценность информации - понятие субъективное.**

Условная и безусловная информация

•Примером условной информации может являться информация о языке, на котором мы говорим (алфавит, словарь, грамматика). Условность подчеркивается тем, что в другом социуме существует другой язык, который имеет свой алфавит, свой набор правил и прочее. И он может быть не хуже и не лучше нашего.

•Безусловной называется информация о реально происходящих событиях. Земля круглая и вращается вокруг солнца – пример безусловной информации.

Литература к теме 1.

1. Философская энциклопедия. В 5-х т. — М.: Советская энциклопедия. Под редакцией Ф. В. Константинова. 1960—1970.
2. Философский энциклопедический словарь. Электронный ресурс http://slovari.bibliofond.ru/enc_philosophy_word/%D7%C8%D1%CB%CE/
3. *Кастлер Г.* Возникновение биологической организации. – М.: Мир, 1967.
4. И.В. Мелик-Гайказян. «Информационные процессы и реальность». М.: 1997.
5. *Ландауэр Р.* Необратимость и выделение тепла в процессе вычислений // В кн. Квантовый компьютер и квантовые вычисления (Библиотека «Квантовый компьютер»). Под.ред. Садовничего В.А. – Ижевск, 1999. Т.2. С.9 – 32.
6. *Кадомцев Б.Б.* Динамика и информация. Изд.2-е. – М.: Ред. журн. УФН, 1999. – 400 С.
7. *Чернавский Д.С.* Синергетика и информация. Динамическая теория информации. Изд.3-е, доп. – М.: URSS, 2009. – 300 С. ISBN 978-5-397-00207-3.
8. *Климонтович Ю.Л.* Введение в физику открытых систем – М.: Янус-К, 2002. – 284 С. ISBN 5-8037-0101-7.
9. *Корогодин В.И.* Информация и феномен жизни. Пущино.: АН СССР, 1991. 202 С.
10. *Корогодин В.И.* Информация и феномен информации. Пущино: 2007. 197 С.
11. *Пригожин И.Р., Стенгерс И.* Время, хаос, квант. К решению парадокса времени. Пер. с англ./ Под.ред. Аршинова В.И. Изд.6-е. - М.: КомКнига, 2005. 232 С. ISBN 5-484-00180-3.

12. *Кальоти Дж.* От восприятия к мысли: О динамике неоднозначного и нарушениях симметрии в науке и искусстве. – М.: Мир, 1998. – 221С. ISBN 5-03-003306-8.
13. *Ижикевич Е.М., Малинецкий Г.Г.* О возможной роли хаоса в нейросистемах // ДАН. 1992. Т. 326. С.627 – 632.
14. *Фриман Дж.У.* Динамика мозга в восприятии и сознании: творческая роль хаоса // В сб. «Синергетика и психология». Вып.3. "Когнитивные процессы". – М.: Когито-Центр, 2004. С.13-28.
15. *Князева Е.Н.* Методы нелинейной динамики в когнитивной науке // В сб. «Синергетика и психология». Вып.3. "Когнитивные процессы". – М.: Когито-Центр, 2004. С. 29 – 48.
16. *Комбс А.* Сознание: Хаотическое и странно-аттракторное // В сб. «Синергетика и психология». Вып.3. "Когнитивные процессы". – М.: Когито-Центр, 2004. С. 49 – 60.
17. *Hartley, R.V.L.* Transmission of Information // Bell System Technical Journal, July 1928, pp. 535–563. http://www.dotrose.com/etext/90_Miscellaneous/transmission_of_information_1928b.pdf
18. *Shannon C. E.* A Mathematical Theory of Communication // Bell System Technical Journal. — 1948. — Vol. 27. — P. 379—423. http://cm.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf Русский перевод: *Шеннон К. Э.* Математическая теория связи // Работы по теории информации и кибернетике / Пер. С. Карпова. — М.: ИИЛ, 1963. — 830 с.
19. *Стартонович Р.Л.* Теория информации. М.: "Сов. радио", 1975. 424 С.
20. *Бонгард М.М.* Проблемы узнавания. М.: Наука, 1967. - 321 с.
21. *Харкевич А.А.* О ценности информации // Проблемы кибернетики. 1960. В. 4.

2. Аналоговые и цифровые вычислители

Исторически существуют два больших класса представления и обработки информации: цифровые и аналоговые. В настоящее время под вычислениями и, соответственно, под вычислителем (компьютером) понимаются преимущественно цифровые методы. Наш курс касается именно цифровых методов, но необходимо кратко упомянуть и аналоговые. В последнее время появились новые классы вычислителей: квантовые, биологические (вычисления на ДНК), нейро-компьютеры.

Таблица 2.1. Сравнение аналоговых и цифровых методов

	Аналоговые	Цифровые
Представление данных	Аналоговое	Дискретное
Точность представления данных и вычислений	Определяется относительно максимального значения операнда	Определяется относительно шага дискретизации, поэтому выше, чем у аналоговых
Объем и скорость вычислений	+	
Гибкость, универсализм		+

Цифровые методы представления и обработки данных превосходят аналоговые в части точности, гибкости и универсализма, но уступают им в части требований к вычислительной мощности и объему памяти в силу большего объема вычислений. Отсюда следуют и различия в областях их применения – цифровые методы применяются там, где важны точность и универсализм, а аналоговые при создании специализированных процессоров, ориентированных на решение узкого круга задач одного класса (или только одной), отличающихся высокими требованиями к вычислительной мощности. Пример – выполнение в реальном времени интегральных преобразований типа свертки над массивами информации, имеющими двумерную область определения (изображениями). Аналоговые вычислители (АВМ) бывают как электронные, так и оптические.

Переход от аналоговой формы представления информации к дискретной (цифровой) основан на теореме Котельникова [1]

Академик Владимир Александрович Котельников (24.08 (06.09) 1908, г. Казань, Российская империя – 11.02.2005, Москва, Россия). Доказал теорему в 1933 г.

Теоретические основания цифровых методов обработки информации и вычислений. Теорема Котельникова (1933 г.)

Операция преобразования Фурье функции *f(x)*, удовлетворяющей следующим условиям:

1. Функция *f*(*x*) абсолютно интегрируема;
2. Функция *f*(*x*) непрерывна или имеет конечное число разрывов первого рода и конечное число максимумов и минимумов в любых конечных пределах;
3. Функция *f*(*x*) не имеет разрывов второго рода.

Фурье-образ функции *f*(*x*) определяется посредством следующего преобразования (преобразование Фурье функции *f*(*x*))

$$F(\nu) = \int_{-\infty}^{\infty} f(x)\exp(-j2\pi\nu x)dx, \qquad (1)$$

где ν – пространственная частота, ω – круговая пространственная частота, которая связана с пространственной частотой выражением ω = 2πν.

Обратное преобразование Фурье связывает Фурье-образ с его прообразом (самой функцией)

$$f(x) = \int_{-\infty}^{\infty} F(\nu)\exp(j2\pi\nu x)d\nu \qquad (2)$$

так как $F(\nu)d\nu = F(\omega)d\omega$,

то $F(\nu) = F(\omega)\dfrac{d\omega}{d\nu} = F(\omega)\dfrac{2\pi d\nu}{d\nu} = 2\pi F(\omega)$

отсюда $F(\omega) = \dfrac{1}{2\pi}\int_{-\infty}^{\infty} f(x)\exp(-j\omega x)dx$. (3)

Теорема Котельникова лежит в основе методов дискретизации непрерывных сигналов и, тем самым, в основе всех современных методов компьютерной обработки и передачи по каналам связи аудио и видео информации. Применительно к задаче ИИ она представляет интерес в силу того, что информация из внешнего мира поступает на сенсоры в виде непрерывных функций, а как сами сенсоры, так и обрабатывающие ее структуры (нейронные структуры), дискретны.

Пусть имеется сигнал *f*(*x*) со спектром *F*(ν), ограниченным максимальной частотой ν_{max}.

$$f(x) = \int_{-\nu_{max}}^{\nu_{max}} F(\nu)\exp j2\pi\nu x d\nu \qquad (4)$$

Поскольку пределы интегрирования ограничены пределами $-\nu_{min}, \nu_{max}$, то можно продолжить функцию (спектр) вне этих пределов посредством ее периодического повторения с периодом $2\nu_{max}$. Тогда эту новую функцию (спектр) можно представить рядом Фурье

$$F(\nu)=\frac{1}{2}\sum_{k=-\infty}^{k=+\infty}\dot{F}_k \exp j2\pi\frac{k\nu}{2\nu_{max}} \quad , \tag{5}$$

Где

$$\dot{F}_k=\frac{2}{2\nu_{max}}\int_{-\nu_{max}}^{\nu_{max}} F(\nu)\exp -j2\pi\frac{k\nu}{2\nu_{max}}d\nu \tag{6}$$

Подставив выражение (1.5) в формулу (1.4), получим

$$f(x)=\int_{-\nu_{max}}^{\nu_{max}}\left(\frac{1}{2}\sum_{k=-\infty}^{k=+\infty}\dot{F}_k \exp j2\pi\frac{k\nu}{2\nu_{max}}\right)\exp j2\pi\nu x d\nu$$

изменяя порядок интегрирования и суммирования получим

$$f(x)=\frac{1}{2}\sum_{k=-\infty}^{k=+\infty}\dot{F}_k\int_{-\nu max}^{\nu_{max}}\exp\left(j2\pi\nu\left(x+\frac{k}{2\nu_{max}}\right)\right)d\nu$$

нетрудно видеть, что при $x=-k/2\nu_{max}$

$$f\left(-\frac{k}{2\nu_{max}}\right)=\int_{-\nu_{max}}^{\nu_{max}} F(\nu)\exp -j2\pi\frac{k\nu}{2\nu_{max}}d\nu,$$

Откуда

$$\dot{F}_k=\frac{2}{2\nu_{max}}f\left(-\frac{k}{2\nu_{max}}\right),$$

а интеграл

$$\int_{-\nu_{max}}^{\nu_{max}}\exp\left(j2\pi\nu\left(x+\frac{k}{2\nu_{max}}\right)\right)d\nu=$$

$$=\frac{\exp\left(j2\pi\nu_{max}\left[x+\frac{k}{2\nu_{max}}\right]\right)-\exp\left(-j2\pi\nu_{max}\left[x+\frac{k}{2\nu_{max}}\right]\right)}{j2\pi\left[x+\frac{k}{2\nu_{max}}\right]}=,$$

$$=2\nu_{max}\mathrm{Sinc}\left(j2\pi\nu_{max}\left[x+\frac{k}{2\nu_{max}}\right]\right)$$

где $-\mathrm{Sinc}(x)$ обозначение Вудварда для функции вида $\frac{\mathrm{Sin}(x)}{x}$.

Тогда

$$f(x)=\sum_{k=-\infty}^{k=+\infty}f\left(-\frac{k}{2\nu_{max}}\right)\mathrm{Sinc}\left(2\pi\nu_{max}\left[x+\frac{k}{2\nu_{max}}\right]\right). \tag{7}$$

Поскольку k принимает значения от -∞ до +∞, то знаки при k можно поменять местами, а величину $1/2\nu_{max}$ обозначить Δx. Отсюда получим окончательное выражение для дискретного представления непрерывной функции с ограниченным спектром:

$$f(x)=\sum_{k=-\infty}^{k=+\infty} f(k\Delta x)\mathrm{Sinc}\left(2\pi\nu_{\max}\left[x-k\Delta x\right]\right). \qquad (8)$$

Таким образом непрерывная функция со спектром, ограниченным максимальной частотой ν_{max} может быть представлена посредством отсчетов, отстоящих друг от друга на расстояние $\Delta x=1/2\nu_{max}$. А поскольку любой реальный сигнал или процесс (функция одной координаты) или поле (функция двух координат) ограничены (во времени или пространстве), то и описывающая и функция также ограничена, следовательно, число отсчетов также конечно.

3. Цифровой компьютер (ЦК).

3.1. Определения

Цифровой компьютер – машина, которая может решать задачи, выполняя данные ей команды. Последовательность команд, описывающих решение определённой задачи, называется программой. [2]

ЭВМ – устройство, которое принимает данные, обрабатывает их в соответствии с хранимой программой, генерирует результаты и обычно состоит из блоков ввода/вывода, памяти, арифметики, логики и управления. [3]

ЭВМ – искусственная (инженерная) машина, предназначенная для вычислений на основе алгоритмов. [4]

Основное назначение ЦК – выполнение вычислений, формализованных в виде алгоритма (программы). Следовательно, именно свойства алгоритмов, вкупе с наличествующей элементной базой, и определяют принципы построения ЦК – его архитектуру.

АЛГОРИТМ – от algorithm; algorismus, происходит от латинской транслитерации имени средне-азиатского учёного 9 в. Мухаммеда бен Муса аль-Хорезми – программа, определяющая способ поведения (вычисления); система правил (предписаний) для эффективного решения задач. [5]

3.2. Многоуровневая компьютерная организация (иерархическая организация цифрового компьютера)

Суть проблемы: цифровому компьютеру доступны только команды на машинном языке, который мы будем обозначать как язык нулевого уровня – Я0, а пользователи формулируют задачи и методы их решения на совсем другом языке (язык уровня N – ЯN, например, Я1).

Можно выделить два способа решения этой проблемы:

1. Исполнение программы, написанной на Я1, посредством замены каждой команды из Я1 эквивалентным набором команд на Я0. Эта технология называется **трансляцией**. При трансляции вся программа на

Я1 преобразуется в программу на Я0, программа на Я1 отбрасывается, а новая программа на Я0 загружается в память и исполняется.

2. Создание на Я0 программы, получающей в качестве входных данных программу, написанную на Я1. Каждая команда Я1 обрабатывается последовательно исполнением соответствующего набора команд Я0. Это **интерпретация,** а программа на Я0 – **интерпретатор**. При интерпретации каждая команда на Я1 перекодируется на Я0 и сразу же исполняется. Новая программа на Я0, в отличие от трансляции, не создается.

Для каждого уровня языка ЯN можно ввести понятие виртуальной машины соответствующего уровня ЯN – MN.

Но тут возникает новая проблема: для эффективной трансляции и/или интерпретации языки Я0 и Я1 должны быть близки между собой. Но язык машинного уровня Я1 далек от естественного языка ЕЯ. Отсюда вытекает многоуровневая организация как средство решения этой проблемы (Рис.3.1). [2]

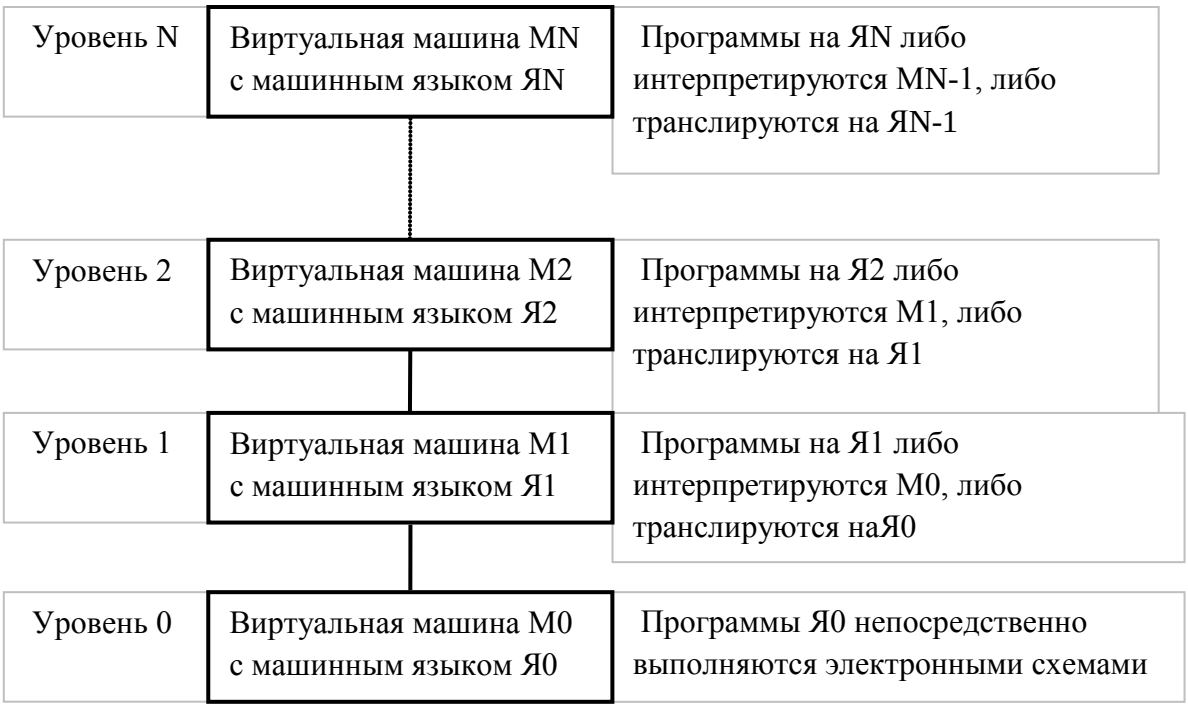


Рис.3.1. Принципиальная схема многоуровневой машины.

Язык и виртуальная машина взаимосвязаны. Каждая виртуальная машина поддерживает вполне определенный язык, команды которого она может выполнить, и наоборот.

Обратим внимание на определение «виртуальная». Можно создать реальную машину, поддерживающую язык высокого уровня, но вопрос упирается не в принципиальную возможность или невозможность её создания, а в целесообразность по критерию стоимость/эффективность.

На Рис.3.2 дана принципиальная схема «шестиуровневой» машины. Определение «шестиуровневая» взято в кавычки, так как на схеме семь уровней, самый нижний из которых – уровень элементной базы или физических устройств, включая физические принципы и механизмы их функционирования, программисты и «компьютерщики» обычно исключают из рассмотрения. Но именно он и определяет не только характеристики вычислителя, но и саму возможность его создания, т.е. является базовым уровнем. Но сейчас рассматривать его не будем.

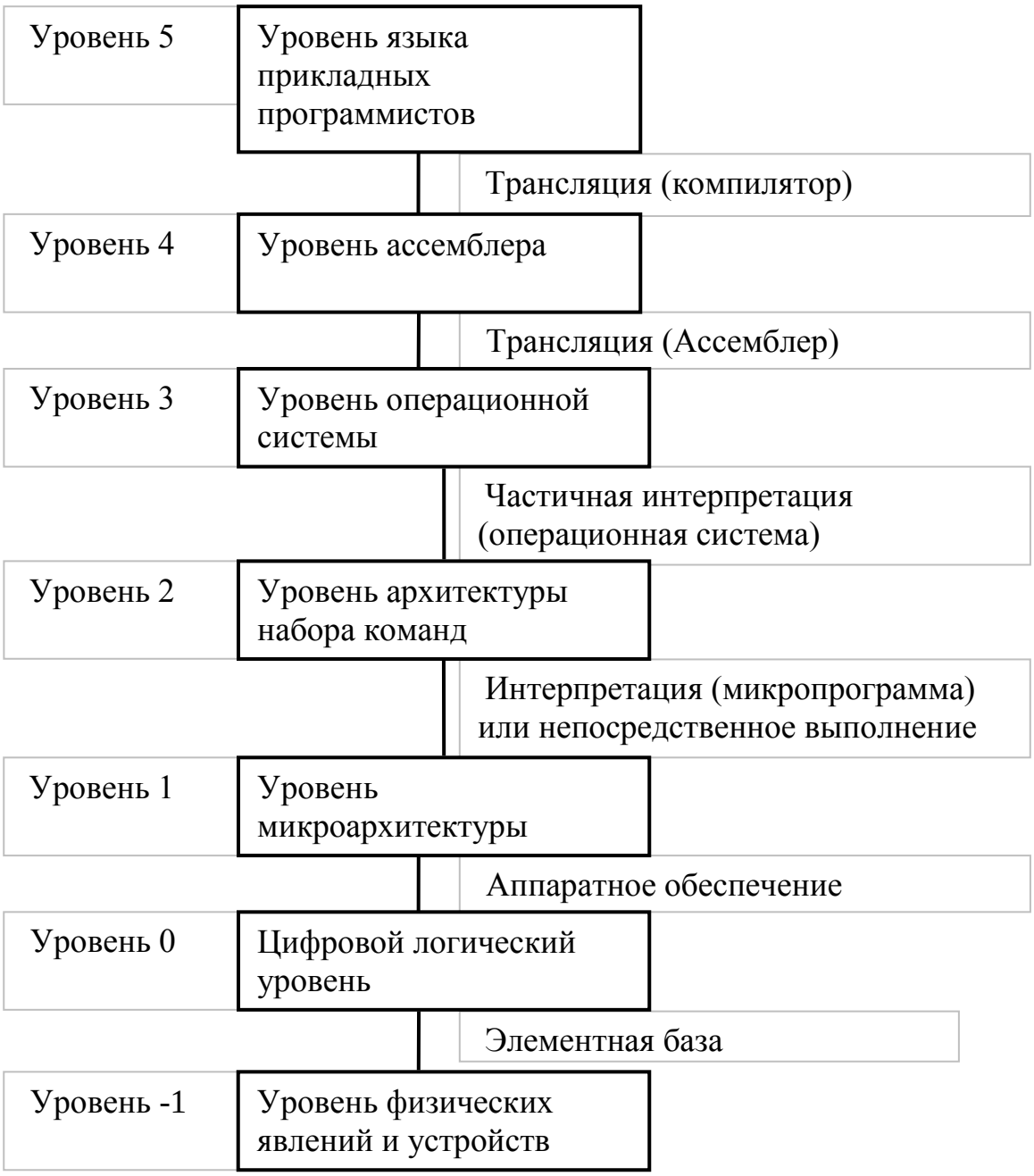


Рис.3.2. Принципиальная схема «шестиуровневой» машины. Под каждым уровнем со сдвигом вправо указан способ его поддержки.

<u>Уровень 0</u> – цифровой логический уровень. На этом уровне объекты называются вентилями. У каждого вентиля есть один или несколько

входов, на которые поступают бинарные сигналы (0 или 1), вентиль вычисляет простые функции этих сигналов, например, НЕ (отрицание), И (конъюнкция), ИЛИ (дизъюнкция). Несколько вентилей формируют 1 бит памяти, который может содержать 0 или 1. Биты памяти объединяют в группы по 16, 32 или 64 бита – это **регистры**. Каждый регистр может содержать одно число в двоичном формате.

Уровень 1 – уровень микроархитектуры. Образуется наборами регистров (8 или 32), которые формируют локальную память и **арифметико-логическое устройство** (**АЛУ**). АЛУ реализует арифметические операции.

Регистры вместе с АЛУ формируют **тракт данных. Базовая операция тракта данных** – выбирается один или два регистра (в зависимости от того, какая операция – одноместная (отрицание) или двухместная (сложение, вычитание, умножение) выполняется); над операндами выполняется операция и результат помещается в регистр результата операции.

Работа тракта данных может контролироваться особой программой – **микропрограммой** (преимущественно, на старых моделях), а может управляться напрямую аппаратными средствами (преимущественно в современных моделях).

Если тракт данных контролируется микропрограммой, то на этом уровне обычно находится программный интерпретатор и уровень может называться уровнем микропрограммирования. **Микропрограмма** в этом случае – **интерпретатор** команд на уровне 2. Она последовательно считывает команды из памяти и исполняет их, использую тракт данных.

Программы, написанные на машинном языке (Я1) могут непосредственно, т.е. без использования **интерпретаторов** и **трансляторов**) исполняться электронными устройствами М0. Эти электронные устройства, включая память и средства ввода-вывода, т.е. реальные материальные устройства, формируют **аппаратное обеспечение** вычислителя.

Программное обеспечение – модели, алгоритмы, программы. В первых компьютерах различие между программным и аппаратным обеспечением было очевидно. При этом важно иметь в виду, что **программное и аппаратное обеспечение компьютера логически эквивалентно** [2]. Поэтому выбор того, каким способом будет реализована та или иная операция – программно или аппаратно, каждый раз принимается исходя из анализа комплекса сопутствующих факторов по критерию стоимость/эффективность.

Определение. **Интерпретация** – покомандный (построчный) анализ программы, перевод в машинные коды и выполнение.

Достоинства интерпретации:

- бóльшая переносимость – интерпретируемая программа будет работать на любой платформе, имеющей соответствующий интерпретатор;

- как правило, более совершенные и наглядные средства диагностики ошибок в исходных кодах;
- упрощение отладки исходных кодов программ;
- меньшие размеры машинного кода по сравнению с кодом, генерируемым обычными компиляторами.

Недостатки интерпретации:

- для интерпретации необходима программа-интерпретатор, которая, впрочем, может быть очень компактной;
- интерпретируемая программа выполняется обычно медленнее в силу затрат ресурсов на промежуточный анализ исходного кода и планирование его выполнения;
- скорость работы интерпретируемых программ снижается из за отсутствия оптимизации итогового кода.

Определение. **Трансляция** – преобразование программы, написанной на одном языке, в программу на другом. Выполняется над программой в целом. Трансляция в машинный язык называется **компиляцией. Транслятор (компилятор)** – обычно более сложная программа, чем интерпретатор. За счет обработки программы в целом осуществляется анализ и оптимизация промежуточного кода. Трансляция может быть выполнена один раз с последующим выполнением транслированной программы на разных машинах.

Уровень 2 – уровень архитектуры набора команд. Этот уровень, в отличие от более низких, уже доступен пользователю, так как он публикуется в руководствах к компьютерам. Описываемые в этих руководствах наборы команд на самом деле не команды машинного уровня, так как в реальности они исполняются микропрограммами-интерпретаторами или аппаратным обеспечением.

Отметим, что первые ЭВМ (40-е годы XX века) имели только два уровня – те, что мы обозначили как уровни 0 и 2. Это была эпоха ламповых ЭВМ, отличавшихся низкой надежностью. Проблема низкой надежности цифрового логического уровня усугублялась тем, что развитие набора команд было возможно только за счет развития электронных схем, увеличения их сложности, т.е. снижения надежности вычислителя в целом. **Уровень 1** был введен в 1951 году Морисом Уилксом из Кембриджского университета – он предложил кардинально упростить аппаратное обеспечение за счет введения промежуточного уровня 1, предназначенного для интерпретации программ уровня 2 (архитектуры набора команд).

Эту интерпретация выполняет спеуиальная неизменяемая программа – **микропрограмма.** В результате, на аппаратное обеспечение (уровень 0) возложено исполнение только ограниченного набора команд микропрограммы, что и дает снижение требований к аппаратному обеспечению за счет ограниченности набора команд микропрограммы. Этот подход, известный также под названием **микропрограммирование,**

наглядно демонстрирует правомочность постулата о логической эквивалентности программного и аппаратного обеспечения.

Уровень 3 – уровень операционной системы. Этот уровень можно рассматривать как гибридный, поскольку большинство команд этого уровня наличествуют также и на предыдущем уровне – архитектуры набора команд. Особенности этого уровня:

- новый набор команд;
- другая организация памяти;
- возможность исполнения нескольких программ одновременно;
- ряд иных.

Тем самым на этом уровне обеспечивается большее разнообразие, чем на предыдущих.

Исполнение новых средств и команд уровня 3 обеспечивает работающий на уровне 2 интерпретатор, называющийся **операционной системой** (**ОС**). Те команды уровня 3, что идентичны командам уровня 2, ОС не задействуют и исполняются микропрограммой или аппаратно. Поэтому уровень ОС часто определяется как гибридный. [2]

Уровни 1 – 3 прикладными программистами обычно не используются. Эти уровни ориентированы на интерпретаторы и трансляторы, обеспечивающие работу на более высоких уровнях, они разрабатываются **системными программистами**. Уровни 2 – 3 всегда интерпретируются. **Прикладные программисты** работают на уровнях 4 – 5. Эти уровни обычно транслируются (транслирующая программа – **ассемблер**).

Таблица 3.1. Системные и прикладные уровни ЦК

Уровни	1-3	4-5
Использование	Системные программисты	Прикладные программисты
Механизмы поддержки более высоких уровней	Интерпретаторы	Трансляторы
Машинные языки	Цифровые	Символьные

Уровень 4 – символическая форма языков более низкого уровня. Символический язык транслируется в язык более низкого уровня (цифровой) **ассемблером**.

Определение. **Ассемблер** (assembler (англ.) – сборщик) – компилятор исходного текста программы, написанной на языке ассемблера (символическом), в программу на машинном языке (цифровом). Ассемблер специфичен для конкретной архитектуры, операционной системы и синтаксиса.

Язык ассемблера – это машинно-ориентированный язык низкого уровня, допускающий использование **мнемонических алфавитных кодов** операций (символов), присваивание символических адресов регистрам и

памяти, задание схем адресации, использование различных систем счисления, символических меток (имен) для строк программы.

Команды языка ассемблера соответствуют командам процессора, но представлены в более удобной для человека символьной форме – в виде мнемонических кодов. Команды языка ассемблера делятся на два типа: машинные и директивы. Директивы – это команды, не переводимые в машинные, а исполняемые непосредственно ассемблером.

Языки ассемблера занимают промежуточное положение между машинными кодами (исторически первым поколением языков программирования) и языками высокого уровня – третьим поколением языков программирования.

Уровень 5 – уровень языков высокого уровня. Язык высокого уровня – язык для прикладных программистов. Эти языки транслируются на низшие уровни **компиляторами**, но в некоторых случаях используются и интерпретаторы. Например, интерпретаторы могут применяться в конкретной прикладной области, например, для символической логики.

4. Архитектура и организация компьютера

Под **архитектурой ЦК** в общем обычно понимается набор типов данных, операций и характеристик каждого отдельно взятого уровня ЦК. Она связана с теми аспектами, что видимы программирующему пользователю соответствующего уровня. Например, ресурсы памяти – это аспект архитектуры. А технология реализации запоминающих устройств, физические принципы и механизмы их работы, хотя и являются базовыми, но к архитектуре компьютера в этом смысле не относятся. [2]

Но при анализе понятия **архитектура ЦК** нужно иметь в виду, что на сегодня разработка ЦК – это не независимая разработка "железа" (hardware) и программного обеспечения (software) по отдельности, но их разработка в едином комплексе. Понятие архитектуры актуально в контексте взаимодействия аппаратных и программных компонентов.

Можно выделить условный стандартный цикл работы ЦК:

1. выборка инструкции из основной памяти;
2. выборка операндов из оперативной памяти и/или регистров;
3. выполнение инструкции;
4. фиксация результата – запись в память, регистр, вывод.

На сегодня существует ряд трактовок и определений понятия архитектуры ЦК. Возможно, что термин «архитектура» применительно к ЦК впервые был использован корпорацией IBM в середине 60-х годов при разработке семейства IBM 360. Некоторые специалисты считают термины «архитектура» и «организация» синонимами [2]. Под термином «архитектура» часто понимается представление возможностей ЦК с точки зрения пользователя, разрабатывающего программу на машинно-

ориентированном языке. Соответственно, архитектура отражает те стороны структуры и функционирования ЦК, что существенны и видимы для такого программирующего пользователя в контексте разрабатываемых им программ.

Архитектура – совокупность характеристик ЦК, существенных с точки зрения пользователя [6]. Можно видеть, что ключевой момент здесь – способ использования ЦК.

Другая формулировка: **Архитектура** – совокупность общих принципов организации аппаратно-программных средств и их характеристик, определяющая функциональные возможности вычислителя при решении соответствующих классов задач.

Еще определение: **Архитектура** – логическое построение вычислителя, как оно представляется программисту, разрабатывающему программу на машинно-ориентированном языке [7].

Согласно [2]: "*Архитектура связана с аспектами, которые видны программисту. Например, сведения о том, сколько памяти можно использовать при написании программы – часть архитектуры, а аспекты разработки (например, какая технология используется при создании памяти) не является частью архитектуры. Изучение того, как разрабатываются те части компьютерной системы, которые видны программистам, называется изучением компьютерной архитектуры. Термины компьютерная архитектура и компьютерная организация означают, в сущности, одно и тоже…*".

Архитектура может трактоваться также как раскрытие способов взаимодействия центрального процессора (ЦП) с памятью, периферийными устройствами, окружением и пользователем (интерфейсный подход).

Альтернативный подход основан на противопоставлении понятий «архитектура» и «организация». В рамках этого подхода, **архитектура** определяет возможности ЦК, а **организация** – реализацию этих возможностей в конкретных моделях. Правомочность такого подхода видна при сравнении моделей, принадлежащих одному семейству – как правило, эти модели обладают одной архитектурой, но разной структурной организацией.

Согласно [8]: "*При описании компьютерных систем принято различать их структурную организацию и архитектуру. Хотя точное определение этим понятиям дать довольно трудно, среди специалистов существует общепринятое мнение о смысле этих понятий и различий между ними.*

Термин архитектура компьютерной системы (компьютера) относится к тем характеристикам системы, которые доступны извне, то есть со стороны программы или, с другой точки зрения, оказывает непосредственное влияние на логику выполнения программ.

Под термином структурная организация компьютерной системы подразумевается совокупность операционных блоков (устройств) и их

взаимосвязей, обеспечивающих реализацию спецификаций, заданных архитектурой компьютера.

В число характеристик архитектуры входят набор машинных команд, формат разрядной сетки для представления данных разных типов, механизм обращения к средствам ввода/вывода и метод адресации памяти.

Характеристики структурной организации включают скрытые от программиста детали аппаратной реализации системы: управляющие сигналы, аппаратный интерфейс между компьютером и периферийным оборудованием, технологию функционирования памяти".

Организация ЦК

- **структурная организация** определяет устройство ЦК - задает структуру на уровне устройств ЦК и организацию связей между этими устройствами на уровне аппаратных интерфейсов.
- **функциональная организация** определяет принципы функционирования ЦК, т. е. собственно вычислительные процессы.

Два уровня представления архитектуры ЦК:

- **программная архитектура** включает в себя аспекты, видимые программистам и, соответственно, программам
- **аппаратная архитектура** включает аспекты, невидимые для программиста.

В этом смысле понятие аппаратной архитектуры и структурной организации ЦК можно рассматривать как синонимы.

В связи с делением программистов на прикладных и системных, программную архитектуру также можно разделить на 2 вида: прикладную и системную.

Основными компонентами архитектуры ЦК принято считать следующие:

1. Вычислительные и логические возможности
    1. Система команд
    2. Формат команд - то, из чего состоит команда
    3. Способы адресации
    4. Назначение и состав регистров
2. Аппаратные средства
    1. Структура
    2. Организация памяти
    3. Организация ввода-вывода
    4. Принципы управления
3. Программное обеспечение
    1. ОС
    2. Языки программирования
    3. Прикладное ПО

Основные элементы архитектуры и структурной организации ЦК

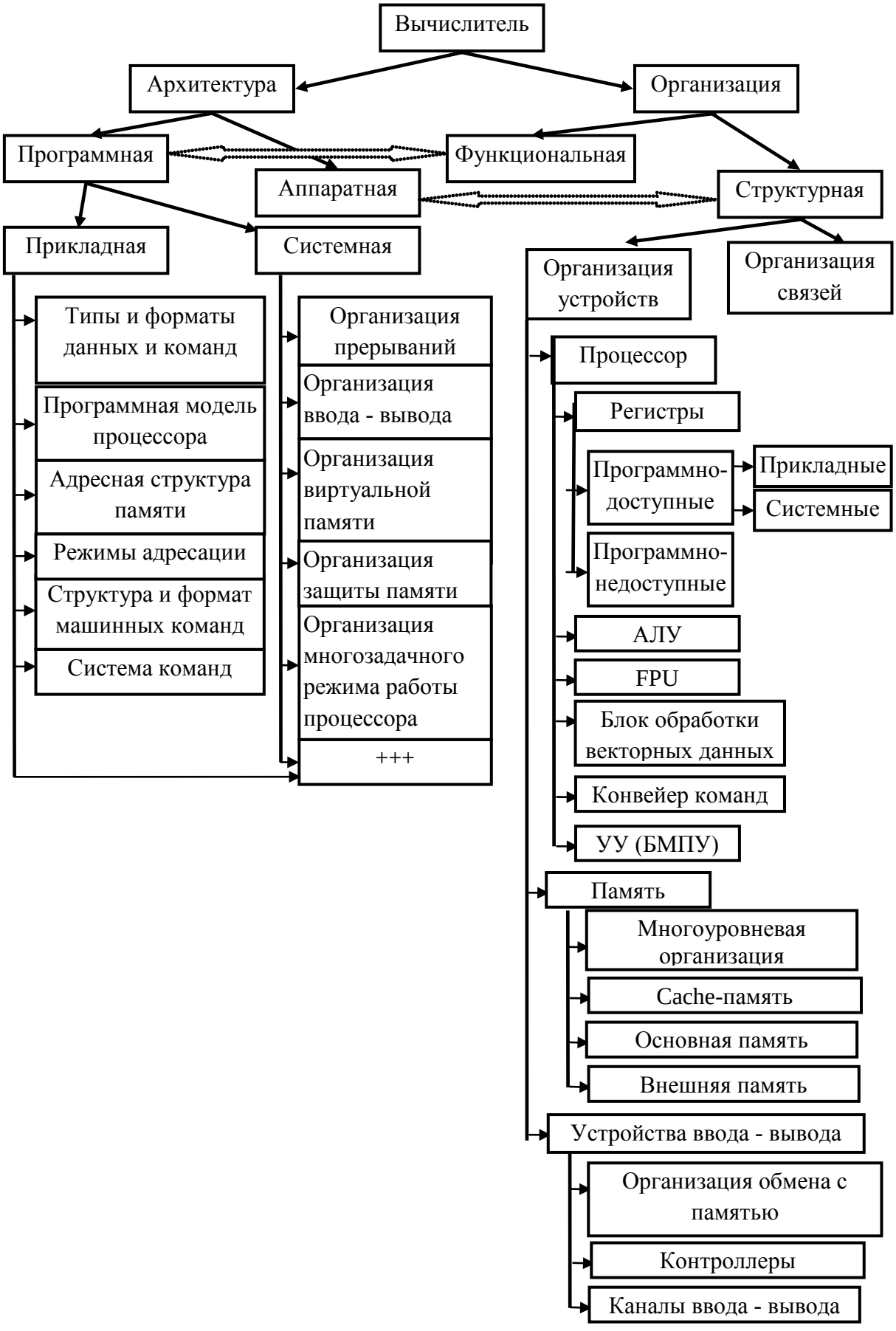


Рис.4.1. Основные элементы архитектуры и структурной организации ЦК

Гарвардская и Принстонская архитектуры ЦК.

Гарвардская архитектура (ГА) разработана Говардом Эйкеном. Реализована в 1944 г., вес компьютера «Марк I» был 5 тонн, он включал 700 км. проводов, операция умножения выполнялась за 3 секунды. Инструкции хранились на перфоленте, а данные обрабатывались релейными (электромеханическими) регистрами. Основное отличие – в **ГА** линии передачи команд и данных физически разделены.

Достоинство ГА – параллелизм передачи, чтения и обработки команд и данных, что повышает быстродействие, но усложняет реализацию по сравнению с фонНеймановской. Характеристики инструкций и данных (длина слова, временные диаграммы, структуры адресов) различны.

Основные отличительные признаки ГА:

- Память для инструкций и данных – разные физические устройства;
- Каналы передачи инструкций и данных – разные физические устройства;
- Инструкции и данные могут представляться в разных форматах.

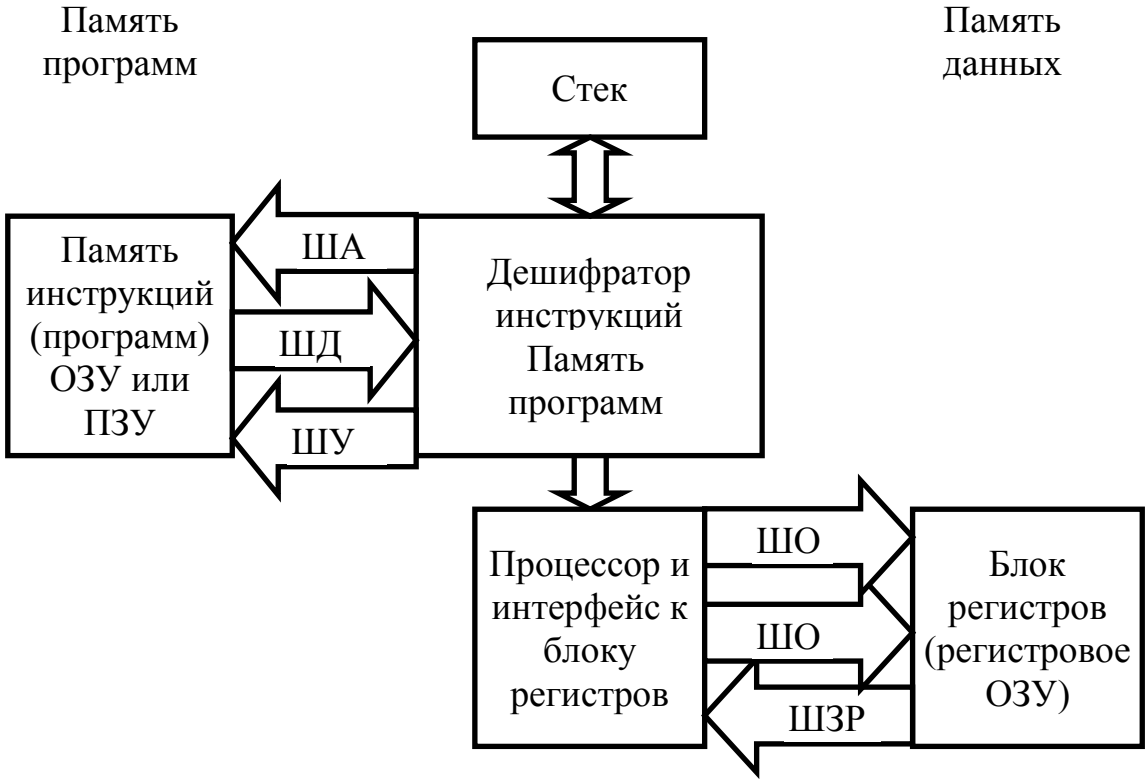


Рис. 4.2. Принципиальная схема Гарвардской архитектуры. Стек – структура данных, организованная по принципу «пришел последним – вышел первым» (примеры: стопка тарелок или ружейный магазин).

В ГА в принципе невозможна запись в память программ, что защищает от случайного или намеренного повреждения управляющей программы.

Основные области применения ГА – микроконтроллеры и сигнальные процессоры, к которым предъявляются требования высокой надежности.

Для повышения скорости работы применяется трехшинная архитектура операционного блока: две шины для считывания операндов, одна – для записи результата. Эти операции выполняются независимо и параллельно.

Принстонская архитектура (ПА) разработана фон Нейманом как попытка облегчить задачу программирования. Основная идея – представление программы в памяти вычислителя в цифровой форме, вместе с данными. Представление чисел в двоичной форме.

Основные атрибуты Принстонской архитектуры [9]:

- и инструкции, и данные представляются и обрабатываются в одном формате;
- и инструкции, и данные хранятся в одной (основной) памяти;
- линейная организация памяти – машинные слова расположены строго последовательно в виде одномерного массива в едином упорядоченном пространстве адресов, каждый адрес имеет последовательный номер 0, 1, 2 …;
- последовательное выполнение – инструкции извлекаются из памяти и выполняются последовательно до тех пор, пока не встретится специальная инструкция перехода (условного или безусловного);
- семантическая неразличимость кодов – во внутреннем машинном представлении не содержится никакой информации, позволяющей отделить инструкции от данных, равно как, и данные различных типов;
- электронная реализация – компьютер полностью электронное устройство;
- фиксированная организация – в процессе вычислений топология, архитектура и список команд не меняются;
- аккумулятор – особый регистр (у фон Неймана на 40 бит) внутри арифметико-логического устройства – типичная команда прибавляла слово из памяти к аккумулятору и сохраняла содержимое аккумулятора в памяти.

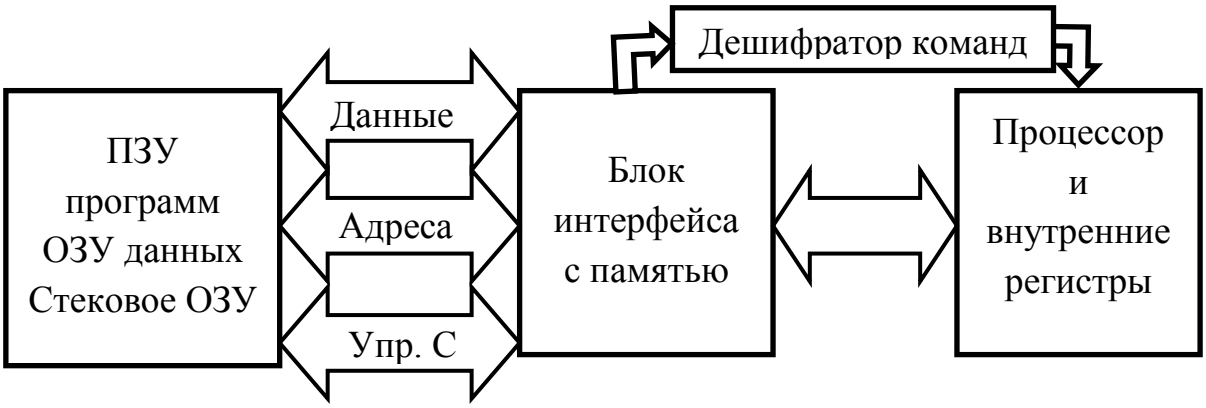


Рис.4. 2. Принципиальная схема Принстонской архитектуры (фон Неймановской).

Узкое место фон Неймановского компьютера – магистраль передачи данных между АЛУ и ОП.

Обратим внимание, что современные CISC-процессоры объединяют в себе принципы, заложенные в обе архитектуры: они обладают раздельной кэш-памятью 1-го уровня для инструкций и данных. Т.е. процессорное ядро формально Гарвардское, а программно – фон Неймановское

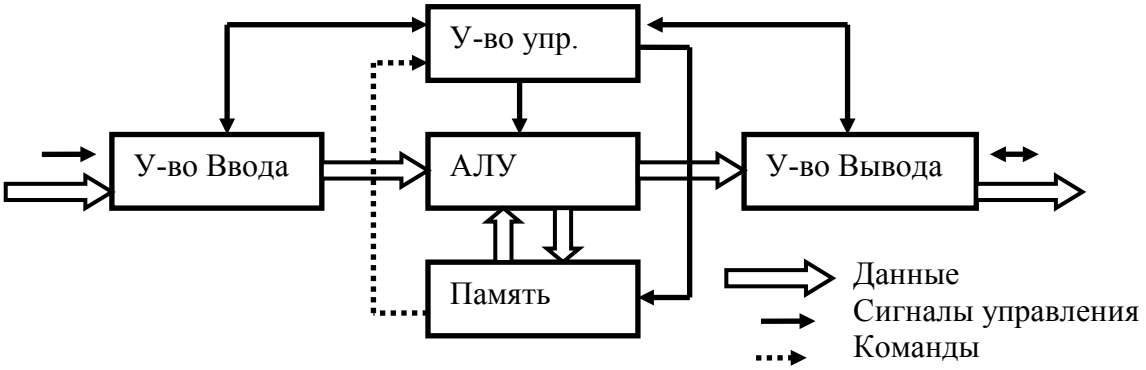


Рис. 4.4. Обобщенная структурная схема компьютера (классическая фон Неймановская или Принстонская).

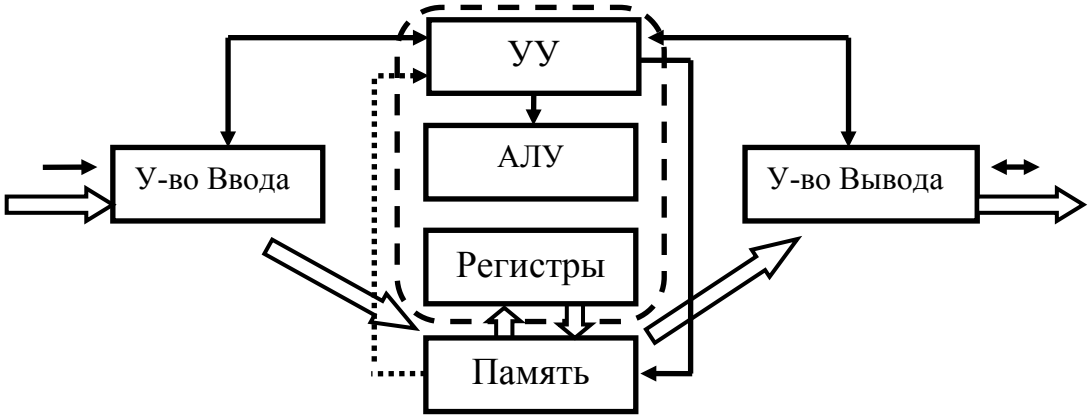


Рис.4.5. Обобщенная структурная схема компьютера с прямым доступом к памяти. Штриховой линией показан центральный процессор (ЦП)

Устройство управления (УУ) организует автоматическое выполнение программ и функционирование вычислителя. Основная задача УУ – преобразование команд считываемой из памяти программы в управляющие сигналы и их распределение по цепям управления.

Арифметико-логическое устройство (АЛУ) предназначено для выполнения арифметических и логических операций над поступающими данными.

Память физически организована как массив запоминающих элементов (ячеек), способных хранить некоторую единицу информации.

Устройство ввода решает задачу преобразования входящей информации в последовательность электрических сигналов, воспринимаемых ЭВМ.

Устройство вывода решает обратную задачу – преобразует последовательность электрических сигналов, формируемых ЭВМ, в форму, удобную для восприятия пользователем.

УУ и АЛУ вместе формируют центральный процессор (ЦП).

Концепция фон Неймана (ПА) предполагает одно АЛУ и одну основную память. В контексте развития параллельных методов и алгоритмов эта особенность Принстонской архитектуры является скорее минусом. Узкое место ПА – магистраль передачи данных между АЛУ и ОП.

Достоинства фон Неймановской (Принстонской) архитектуры:

Данные поступают в память из одного источника. Аналогично и при выводе.

Недостатки классической фон Неймановской архитектуры (с прямым доступом к памяти):

Информация попадает в память через АЛУ без изменений, занимая аппаратные ресурсы. Наличие в памяти 2х типов: данные и команды.

Необходима система прерываний и приоритетов, что усложняет архитектуру.

Для решения этих проблем в большинстве больших ЭВМ применяется идея буферизации данных, реализуемая в канале ввода-вывода (Main Frame), представленная на рис.4.6.

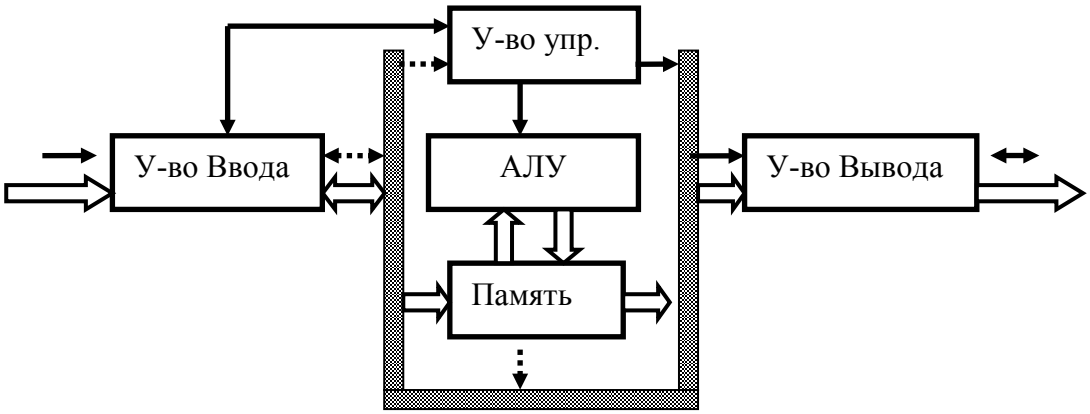


Рис. 4.6. Обобщенная структурная схема компьютера с каналом ввода – вывода (фон Неймановская архитектура с Main Frame).

Обратим внимание, что современные CISC-процессоры объединяют в себе принципы, заложенные в обе архитектуры: они обладают раздельной кэш-памятью 1-го уровня для инструкций и данных. Т.е. процессорное ядро формально Гарвардское, а программно – фон Неймановское

Принцип программного управления

Принцип программного управления заключается в том, что после загрузки в ЦК программы, вся работа ЦК управляется только загруженной программой, т.е. программа управляет выполнением самой себя – никакого внешнего управления не требуется.

Программа пишется программистом как последовательность команд. Последовательное выполнение команд реализуется следующим образом:

Первая команда сообщается ЦК, заносится в регистр команд и выполняется, к её адресу прибавляется её длина (в байтах) – это дает адрес второй команды.

Аналогично, по выполнении каждой последующей команды к её адресу прибавляется её длина, что определяет адрес следующей команды. Если этот процесс нарушается, выполняется процесс условного перехода.

Этот принцип выполняется на уровне как команд, так и микрокоманд.

Программа:								Микропрограмма:		
A1	K1	→	КОП	A1	A2	A3	→	Микрокоманда 1	→	Микрооперация 1
A1+A2	K2							Микрокоманда 1		Микрооперация 1
	K3							Микрокоманда 1		Микрооперация 1
	…	→	Регистр команд					…		…
	KN							Микрокоманда M		Микрооперация M

Команда имеет оперативную и адресную части.

В оперативной части команды находится **код операции** (КОП), он определяет какая именно операция выполняется. Занимает 8 бит.

Адресная часть может включать один, два, три и четыре адреса. В последнем случае A4 – адрес следующей команды.

Регистр команд – регистр, после помещения в который, тело команды начнет выполняться.

Команды дробятся на микрокоманды. Микрокоманды и микрооперации находятся в постоянной памяти. Каждой микрокоманде может соответствовать одна или несколько микроопераций

Микрооперация – элементарное действие внутри ЦК.

Типы, форматы и способы представления данных в ЦК.

Формат данных - внутреннее представление данных: разрядность и назначение битов. Например, для знаковых целых чисел крайний левый бит формата отводится для представления знаков. Для обозначения форматов стандартной длины принято использовать следующие наименования:

- Байт (B – Byte) – 8 бит
- Слово (W – Word) – 16 бит
- Двойное слово (DW – Double Word) – 32 бита
- Учетверенное слово (QW – Quadro Word) – 64 бита

Ключевой вопрос касательно данных, представляемых в ЦК, – наличие или отсутствие аппаратной поддержки для конкретного типа и формата данных. Под аппаратной поддержкой понимается наличие в системе команд ЦК некоторого множества машинных команд, предназначенных для обработки данных определенного типа, представленных в соответствующих форматах.

Литература.

1. *Котельников В. А.* О пропускной способности эфира и проволоки в электросвязи — Всесоюзный энергетический комитет. // Материалы к I Всесоюзному съезду по вопросам технической реконструкции дела связи и развития слаботочной промышленности, 1933. Репринт статьи в журнале УФН, 176:7 (2006), 762—770.
2. *Таненбаум Э., Остин Т.* Архитектура компьютера // 6-е издание. — СПб.: Питер, 2013. — 811 с., илл. — ISBN: 978-5-496-00337-7.
3. *Цилькер Б.Я., Орлов С.А.* Организация ЭВМ и систем // Учебник для вузов. — СПб.: Питер, 2004. — 668 с.: ил. ISBN 5-94723-759-8
4. *С. А. Майоров, Г. И. Новиков* Принципы организации цифровых машин // Машиностроение, 1974. 434 С.
5. *http://dic.academic.ru/dic.nsf/enc_philosophy/4131/АЛГОРИТМ* электронный ресурс
6. *Карцев М.А.* Архитектура цифровых вычислительных машин // М.: Наука, 1978. – 295 С.
7. http://vssit.ucoz.ru/index/0-4 электронный ресурс
8. У. *Столлингс.* Структурная организация и архитектура компьютерных систем, 5-е изд. М.: Вильямс, 2002, 896 с.
9. Чекменев С.Е. Архитектура вычислительных систем. Конспект лекций.

5. Элементы и узлы цифрового компьютера

5.1. Алгебра логики. Логические вентили

Логика как наука о рассуждениях (λογος – рассуждение) в классическом виде зародилась в древней Греции и первоначально понималась как атрибут психической сферы человека. Развитие Европейской культурной традиции, ориентированной на объективизацию науки, привело к появлению в середине XIX века математической формализации логики – Булевой алгебры, понимаемой как алгебра логики.

В Булевой логике, формализовавшей Аристотелеву логику, операнды и результаты операций (высказывания в логике высказываний) принимали только два значения: 0 – ложь и 1 – истина.

Основные логические операции:

1. унарная (выполняется над одним операндом) операция отрицания, обозначается равноправными символами N, ¬ или чертой над операндом;
2. бинарная операция конъюнкции («И» или логическое умножение), обозначается символом ∧;
3. бинарная операция дизъюнкции («ИЛИ» или логическое сложение), обозначается символом ∨.

На основе этих операций могут быть сконструированы и другие, уже более сложные логические операции.

В начале XX века появились многозначные логики, в которых значений истинности было уже больше. А во второй половине XX века появилась теория нечетких множеств Лотфи Заде и, на её основе, широкий класс нечетких логик, в которых значения истинности могут описываться нечеткими множествами. Элементная база современных массовых цифровых компьютеров реализует двузначную, т.е. Булеву логику, поэтому в курсе мы ограничимся только двузначной или строгой логикой.

Устройства, выполняющие отдельные логические операции, называются вентилями. Ниже даны примеры условных графических обозначений (УГО) логических вентилей на схемах по ГОСТ 2.734-91 ЕСКД. Обозначения условные графические в схемах. Элементы цифровой техники [1] и US ANSI 91-1984. УГО по системе IEC 60617-12:1997 похожи на обозначения по ГОСТ, но кружок, означающий отрицание, не налагается на границу прямоугольника, обозначающего сам вентиль, а выдвинут вне его – в состав выхода, как в обозначениях по системе ANSI 91-1984.

Таблица 5.1. Условно-графические обозначения основных логических вентилей, упоминаемых в данном курсе

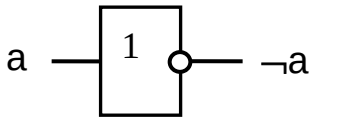
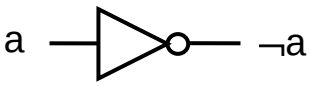
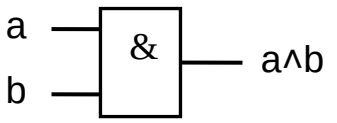
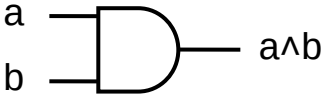
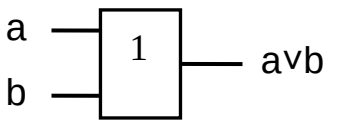
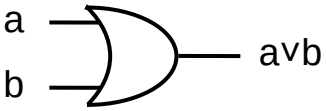
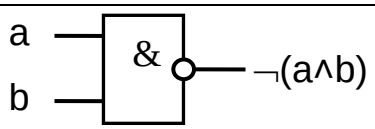
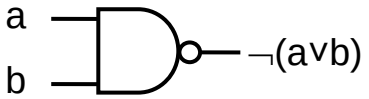
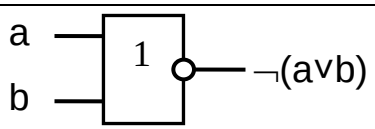
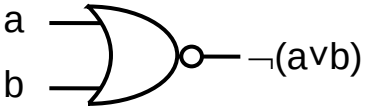
Логический вентиль	ГОСТ 2.734-91	US ANSI 91-1984
Инвертор (НЕ)		
Конъюнктор (И)		
Дизъюнктор (ИЛИ)		
Элемент Шеффера (И-НЕ) (Штрих Шеффера)		
Элемент Пирса (ИЛИ-НЕ) (Стрелка Пирса)		

Таблица 5.2. Таблицы истинности вентилей Таб.5.1 в Булевой логике.

Таблица истинности схемы «НЕ»	
a	¬a
0	1
1	0

Таблица истинности схемы «И»			Таблица истинности схемы «ИЛИ»		
a	b	a^b	a	b	a∨b
0	0	0	0	0	0
0	1	0	0	1	1
1	0	0	1	0	1
1	1	1	1	1	1

Таблица истинности схемы «И-НЕ»			Таблица истинности схемы «ИЛИ-НЕ»		
a	b	¬ab	a	b	¬(a∨b)
0	0	1	0	0	1
0	1	1	0	1	0
1	0	1	1	0	0
1	1	0	1	1	0

5.2. Триггер

Аппаратно, т.е. «в железе» логические вентили реализуются триггерами.

Триггер – простейшее **последовательностное** устройство, которое может длительно находиться в одном из нескольких устойчивых состояний и переходить из одного в другое под воздействием входных сигналов. **Последовательностными** называют такие логические устройства, выходные сигналы которых определяются не только сигналами на их входах, но и предысторией их работы, то есть состоянием элементов памяти. Триггер — один из базовых элементов цифровой техники.

Триггерные схемы классифицируют по следующим признакам:

- способу приёма логических сигналов;
- функциональным возможностям;
- принципу построения;
- числу устойчивых состояний (обычно устойчивых состояний два, реже – больше);
- числу уровней – два уровня (высокий, низкий) в двухуровневых элементах, три уровня (положительный, ноль, отрицательный) в трёхуровневых элементах.

По способу приема сигналов различают асинхронные, синхронные и смешанные триггерные схемы, статические и динамические.

Асинхронный триггер изменяет своё состояние непосредственно в момент появления соответствующего информационного сигнала.

Синхронные триггеры реагируют на информационные сигналы только при наличии соответствующего сигнала на так называемом входе синхронизации С (от англ. clock). Этот вход также обозначают терминами «строб», «такт». Синхронные триггеры в свою очередь подразделяют на триггеры со статическим (статические) и динамическим (динамические) управлением по входу синхронизации С.

Статические триггеры воспринимают информационные сигналы при подаче на вход С логической единицы (прямой вход) или логического нуля (инверсный вход).

Динамические триггеры воспринимают информационные сигналы при изменении (перепаде) сигнала на входе С от 0 к 1 (прямой динамический С-вход) или от 1 к 0 (инверсный динамический С-вход).

Статические триггеры в свою очередь подразделяют на одноступенчатые (однотактные) и двух-ступенчатые (двухтактные).

В одноступенчатом триггере имеется одна ступень запоминания информации, а в двухступенчатом — две такие ступени. Вначале информация записывается в первую ступень, а затем переписывается во вторую и появляется на выходе. Двухступенчатый триггер обозначают ТТ.

По структурному построению – однотактные (триггеры защёлки), двухтактные и триггеры с динамическим управлением. По способу реакции на помехи — прозрачные и непрозрачные. Непрозрачные, в свою очередь, делятся на проницаемые и непроницаемые. По функциональному назначению – RS, D, JK, T, RR, SS, EE, DV.

При изготовлении триггеров применяются преимущественно полупроводниковые приборы (обычно полевые транзисторы), в прошлом – электронные лампы, реле. В настоящее время в основном разрабатываются программируемые логические интегральные схемы (ПЛИС).

Триггеры используются в вычислительной технике как основной элемент для построения более сложных компонентов вычислительных систем: процессоров, регистров, счётчиков, ОЗУ.

По функциональным возможностям триггеры разделяют на следующие классы:

- с раздельной установкой состояния 0 и 1 (RS-триггеры). Если триггер является синхронным – добавляется вход синхронизации С.;
- универсальные (JK-триггеры);
- с приёмом информации по одному входу D (D-триггеры, или триггеры задержки);
- со счётным входом Т (Т-триггеры).

Каждый тип триггера имеет собственную таблицу работы (таблицу истинности). Выходное состояние триггера обычно обозначают буквой Q. Индекс возле буквы означает состояние до подачи сигнала (t) или после подачи сигнала (t+1).

Если триггер синхронный, то существует также дополнительный вход синхронизации. Для того, чтобы такой триггер учёл информацию на

синхронных входах, на входе синхронизации необходимо сформировать активный фронт (обычно положительный).

Триггеры обычно реализуются на основе логических схем «И-НЕ» или «ИЛИ-НЕ». Ниже даны основные варианты реализации триггеров с помощью рассмотренных выше логических вентилей и их УГО.

RS-триггер (от английских *set* – установка, и *reset* – восстановление, сброс, т.е. триггер с установочными входами). Это триггер с раздельной установкой состояния 0 и 1, имеет два симметричных входа S и R и два симметричных выхода Q и ¬Q, причем выходной сигнал ¬ Q является логическим отрицанием сигнала Q . Если триггер является синхронным, то добавляется вход синхронизации C. Пример УГО RS триггера, его реализация на элементах «2ИЛИ-НЕ» и таблица истинности даны ниже.

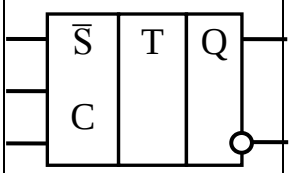


Рис.5.1. УГО RS триггера (синхронного)

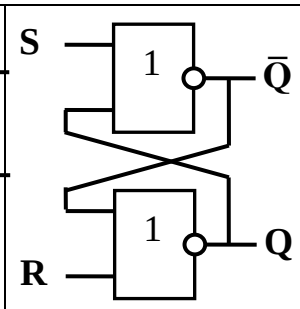


Рис.5.2 RS триггер на элементах «2ИЛИ-НЕ»

S	**R**	**Q**	**¬Q**
0	0	Хранение бита	
1	0	1	0
0	1	0	1
1	1	Запрещенное состояние! Нарушение закона непротиворечивости и неопределенность состояния.	

NB! В литературе встречаются разные варианты взаимного расположения на схемах входов и выходов триггеров! Всегда необходимо обращать внимание на обозначения.

NB! Триггер может быть реализован и на элементах «2И-НЕ».

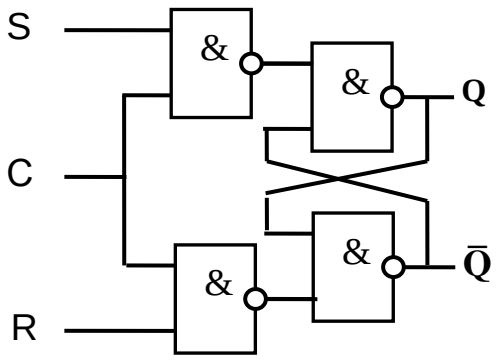


Рис.5.3 RS триггер (синхронный) на элементах «2И-НЕ»

S-триггер – модификация RS-триггера, выходы которого при комбинации сигналов R=S=1 принимают единичное состояние, а при всех остальных комбинациях входных сигналов функционирует как RS-триггер. S - триггер – приоритетный триггер, т.к. S вход имеет приоритет перед R входом.

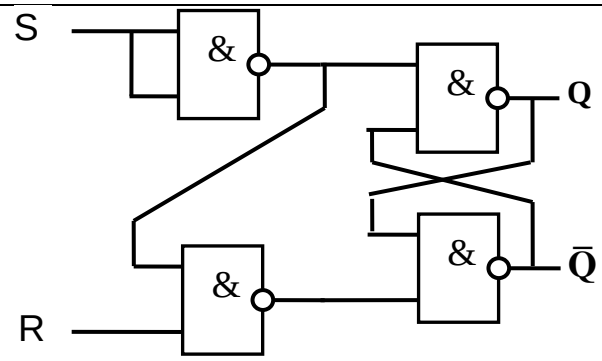


Рис.5.4 S триггер на элементах «И-НЕ»

Табл.5.2 Таблица истинности S-триггера

S	**R**	**Q**	**¬Q**
0	0	Хранение бита	
1	0	1	0
0	1	0	1
1	1	1	1

Состояние выхода S-триггера на n+1 шаге может быть описано следующим образом:

$$Q^{n+1} = S^n + \bar{R}^n Q^n = S^n\left(1 + Q^n\right) = S^n + S^n Q^n + \bar{R}^n Q^n = \overline{\overline{S^n\left(\overline{S^n R^n}\right)Q^n}} \quad . (5.1)$$

R-триггер – модификация RS-триггера, выходы которого при комбинации сигналов R=S=1 принимают нулевое состояние, а при всех остальных комбинациях входных сигналов функционирует как RS-триггер. R - триггер – приоритетный триггер, т.к. R вход имеет приоритет перед S входом.

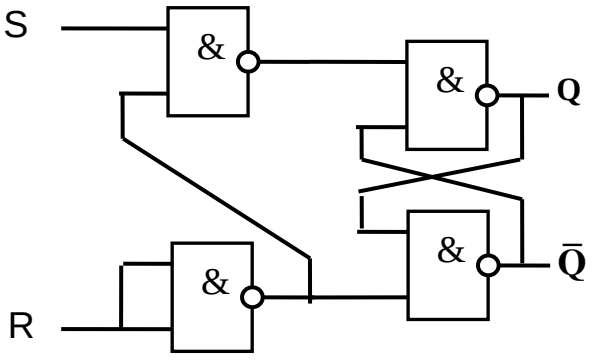


Рис.5.5 R триггер на элементах «И-НЕ»

Табл.5.3 Таблица истинности R-триггера

S	**R**	**Q**	**¬Q**
0	0	Хранение бита	
1	0	1	0
0	1	0	1
1	1	0	0

Состояние выхода R-триггера на n+1 шаге может быть описано следующим образом:

$$Q^{n+1} = \bar{R}^n S^n + \bar{R}^n Q^n = \overline{\overline{\left(R^n S^n\right)\left(R^n Q^n\right)}} \; / \qquad (5.2)$$

E-триггер – модификация RS-триггера, выходы которого при комбинации сигналов R=S=1 не изменяют свое состояние, а при всех остальных комбинациях входных сигналов функционирует как RS-триггер. E - триггер – приоритетный триггер, т.к. тот вход, на котором сигнал появился первым, имеет приоритет перед другим входом.

Состояние выхода E-триггера на n+1 шаге может быть описано следующим образом:

$$Q^{n+1} = S^n Q^n + S^n \bar{R}^n + \bar{R}^n Q^n = \overline{\overline{\left(S^n \overline{R^n}\right)\left(\overline{S^n} R^n\right)} Q^n} \quad . \qquad (5.3)$$

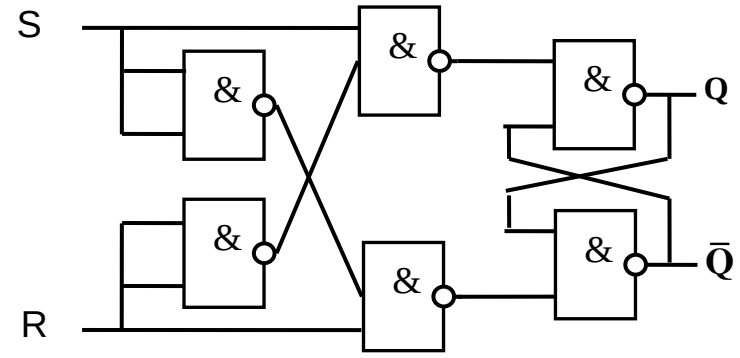


Рис.5.6 Е триггер на элементах «И-НЕ»

Табл.5.4 Таблица истинности Е-триггера

S	**R**	**Q**	**¬Q**
0	0	Хранение бита	
1	0	1	0
0	1	0	1
1	1	Хранение бита	

D-триггер (от англ. delay – задержка). В RS-триггерах для записи логического нуля и логической единицы требуются разные входы, что не всегда удобно. При записи и хранении данных один бит может принимать значение, как нуля, так и единицы. Для его передачи достаточно одного провода. Поскольку сигналы установки и сброса триггера не могут появляться одновременно, то можно объединить эти входы при помощи инвертора. D- триггер способен запоминать по синхросигналу и хранить один бит двоичной информации.

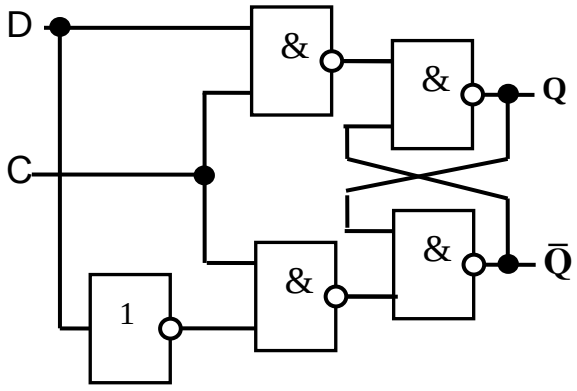


Рис.5.7 D триггер на элементах «И-НЕ»

Табл.5.5 Таблица истинности синхронного D-триггера

C	**D**	**Q(t)**	**Q(t+1)**
0	Произвольное	0	0
0	Произвольное	1	1
1	0	Произвольное	0
1	1	Произвольное	1

В Таблице 5.5. первым двум строкам соответствует режим хранения информации, а двум последним – режим записи информации.

D-триггеры, работающие по фронту. Выше приведен D-триггер, работающий по потенциалу. Потенциал, высокий или низкий, может сохраняться длительное время. В этой связи интерес представляют D-триггеры, работающие по фронту, так как фронт сигнала синхронизации всегда ограничен во времени. В идеале длительность фронта равна нулю. Поэтому в триггере, запоминающем входную информацию по фронту не нужно предъявлять требования к длительности тактового сигнала. Триггер,

запоминающий входную информацию по фронту сигнала синхронизации, может быть построен из двух триггеров, работающих по потенциалу.

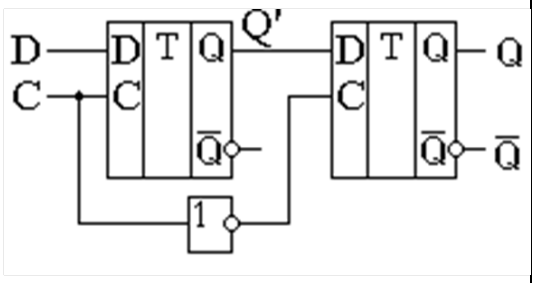


Рис.5.8 D-триггер, работающий по фронту

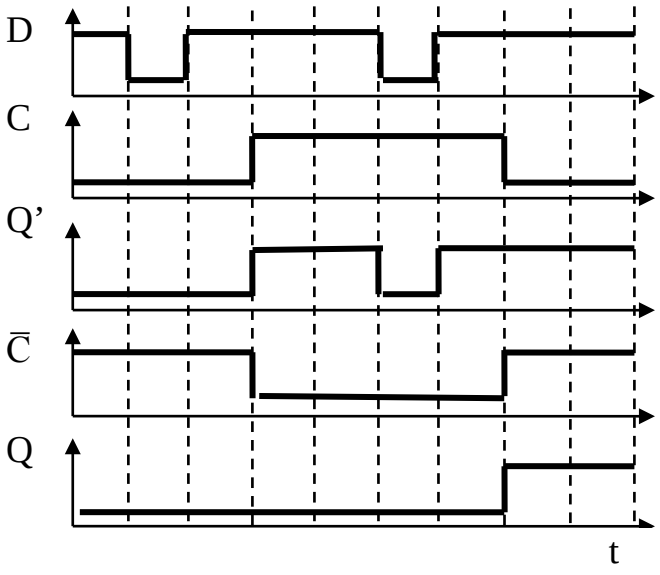


Рис.5.9 Временные диаграмм D-триггера, работающего по фронту

DV – триггер – это D-триггер, имеющий дополнительный управляющий вход V. При V=1 DV-триггер работает по правилам D-триггера. При сигнале V=0 триггер сохраняет свое состояние неизменным – «защелкивается» и хранит информацию независимо от состояния входа D. Наличие дополнительного входа V расширяет функциональные возможности D-триггера – информация на выходах в нужные моменты может быть сохранена в течении необходимого числа тактов.

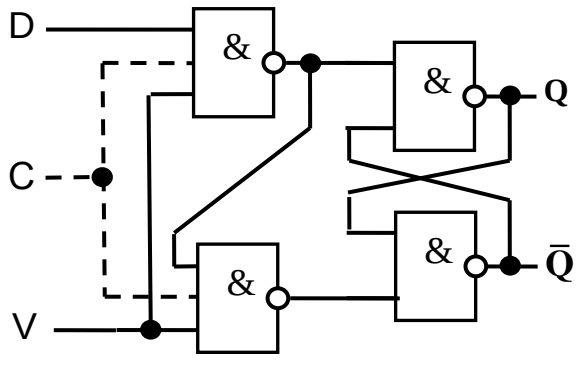


Рис.5.9 DV-триггер на элементах «И-НЕ» (к D-триггеру добавлен вход V)

Табл.5.6 Таблица истинности синхронного DV-триггера

C	D	V	Q(t+1)
0	?	0	Q(t)
0	?	0	Q(t)
1	?	0	Q(t)
1	?	0	Q(t)
1	0	1	0
1	1	1	1

Т-триггеры – счетные. У Т-триггера только один вход. (У TV – два входа, добавлен управляющий вход, как у DV-триггера.)

После поступления на вход Т импульса, состояние триггера меняется на прямо противоположное. При поступлении второго импульса Т-триггер

сбрасывается в исходное состояние. Таким образом, Т-триггер счетчик весьма своеобразный – считать он умеет только до одного.

Используются Т-триггеры для построения схем различных счётчиков. Т-триггеры можно реализовать только на базе двухступенчатых триггеров, подобных D триггеру. Использование двух триггеров позволяет избежать неопределенного состояния схемы при разрешающем потенциале на входе синхронизации С.

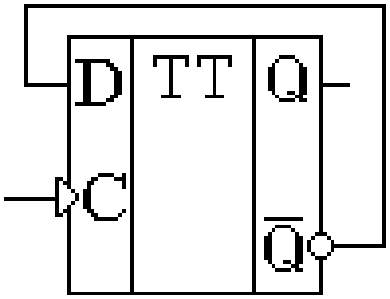


Рис.5.10 Т-триггер на основе D-триггера

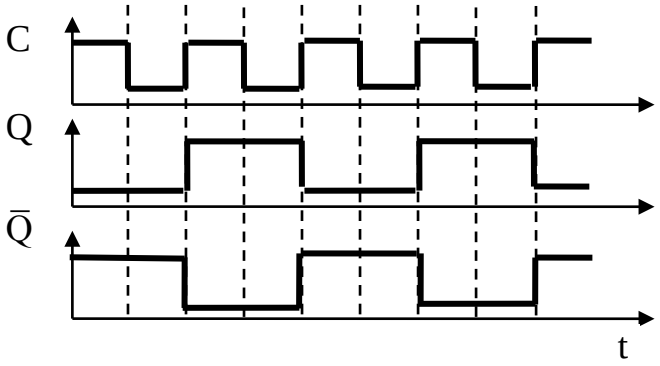


Рис.5.11 Временные диаграммы Т-триггера, построенного на основе D-триггера, работающего по заднему фронту сигнала

JK-триггеры – универсальные (Jump-Kill). В RS триггере есть запрещённые комбинации входов – одновременная подача единичных сигналов на входы R и S. В JK триггере запрещённое состояние исключено.

Для этого его схема изменена так, чтобы при подаче двух единиц JK триггер превратился в счетный (Т-триггер). Тогда, при подаче на вход С импульсов синхронизации этот триггер должен изменить своё состояние на противоположное. Для реализации счетного режима в схеме (Рис.5.12) введена перекрестная обратная связь с выходов второго триггера на входы R и S первого триггера. Обратная связь на R и S входах первого триггера гарантирует невозможность запрещено комбинации, а её перекрестность дает новый режим работы — счетный. В результате, при подаче на входы J и K логической единицы, JK-триггер переходит в счетный режим, подобно Т триггеру. Таблица истинности JK триггера практически совпадает с таблицей истинности синхронного RS-триггера.

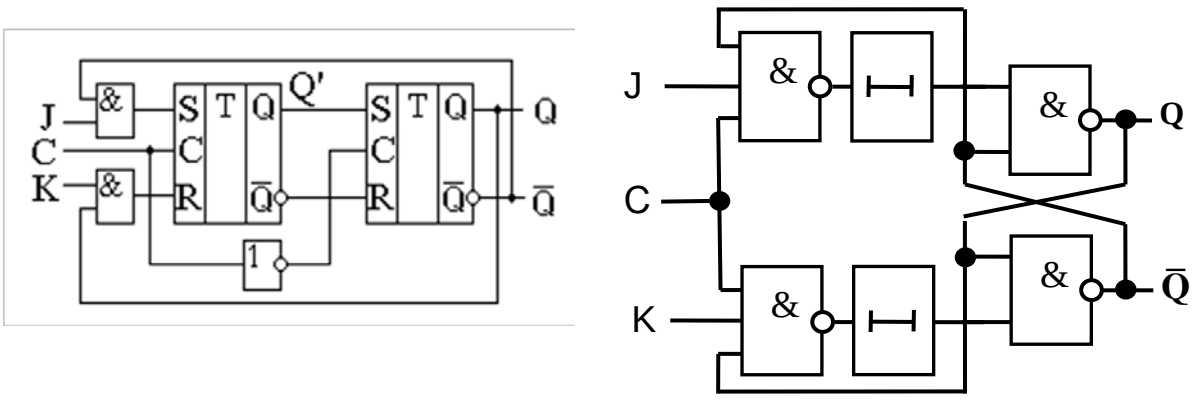


Рис.5.12 JK-триггер

Регистр – это последовательностное логическое устройство, используемое для хранения n-разрядных двоичных чисел и выполнения преобразований над ними. Регистр представляет собой упорядоченную последовательность триггеров, число которых соответствует числу разрядов в машинном слове. С каждым регистром обычно связано комбинационное цифровое устройство, с помощью которого обеспечивается выполнение требуемых операций над словами. Фактически любое цифровое устройство можно представить в виде совокупности регистров, соединённых друг с другом при помощи комбинационных цифровых устройств. Регистры строятся обычно на основе D-триггеров (с приемом информации по одному входу).

Основные операции, выполняемые регистром:

- приём слова в регистр;
- передача слова из регистра;
- поразрядные логические операции;
- сдвиг слова влево или вправо на заданное число разрядов;
- преобразование последовательного кода слова в параллельный и *v.v.*;
- установка регистра в начальное состояние (сброс).

Классифиция регистров вначале обычно проводится в два класса:

- накопительные (регистры памяти, хранения);
- сдвигающие, используемые для вычислений.

В свою очередь, сдвигающие регистры классифицируются:

- по способу ввода-вывода информации:
  - параллельные – запись информации производится одновременно по всем входам, аналогично и считывание идет параллельно со всех выходов;
  - последовательные – запись и считывание информации происходит в первый триггер, а информация из этого триггера перезаписывается в следующий, аналогичная процедура применяется и к остальным триггерам, формирующим регистр;
  - комбинированные;
- по направлению передачи информации:
  - однонаправленные
  - реверсивные.
- по основанию системы счисления;
  - двоичные
  - троичные
  - десятичные.

Параллельные или статические регистры. В параллельных регистрах прием и выдача слов производятся по всем разрядам одновременно. Используются для хранения слов, подлежащих поразрядным логическим преобразованиям. Схемы разрядов в этих регистрах не обмениваются данными между собой. Общие для разрядов обычно цепи управления: тактирования, сброса/установки, разрешения выхода или приема.

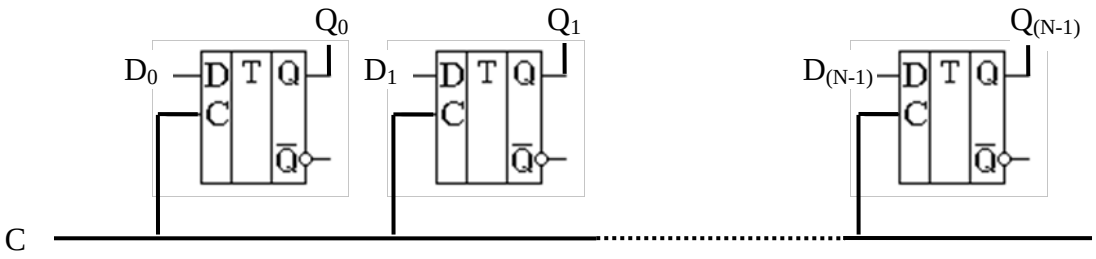


Рис.5.13. Параллельный регистр

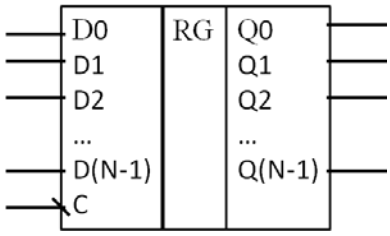


Рис.5.14. УГО параллельного регистра

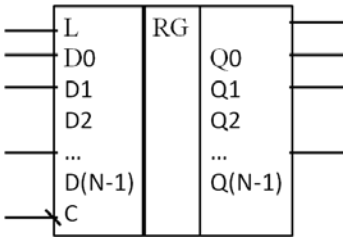


Рис.5.15. УГО параллельного регистра со входом разрешения записи

Устройства (в том числе – регистры), в которых для записи входного параллельного кода D_i используется сигнал разрешения записи L, а тактовый сигнал C не используется, называются устройствами с **асинхронной параллельной записью кода** (рис.5.14).

Устройствами с **синхронной параллельной записью кода** называются такие устройства, в том числе регистры (рис.5.15), в которых для записи входного параллельного кода D_i необходимы:

1. сигнал разрешения записи L и
2. перепад синхросигнала на тактовом входе C.

Последовательные (сдвигающие) регистры представляют собою цепочку разрядных схем, связанных цепями переноса.

В однотактных регистрах слово сдвигается при поступлении синхросигнала. Вход и выход – последовательные. Делятся на:

1. Со сдвигом вправо, обозначаются DSR – Data Serial Right;
2. Со сдвигом влево, обозначаются DSL – Data Serial Left;
3. Реверсивные регистры (сдвиг вправо или влево);
4. Последовательно-параллельные регистры – в них параллельный или последовательный режим записи-считывания информации определяется сигналом на специальном входе F.

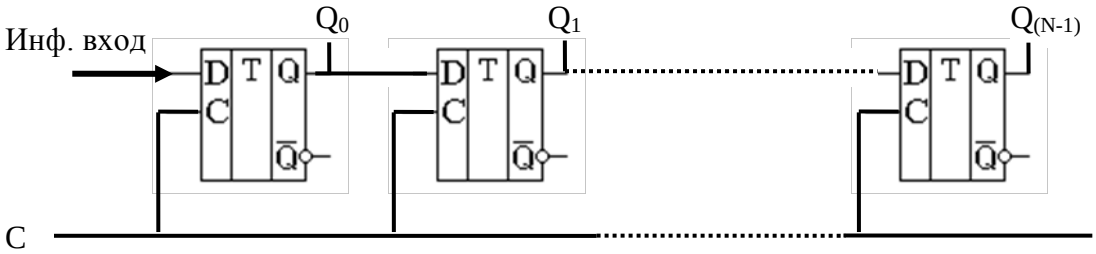


Рис.5.16. Последовательный регистр со сдвигом вправо (DSR)

Шифратор (CD – coder) преобразует требуемый десятичный номер активного элемента в двоичный код. Выделяют шифраторы:

- **приоритетные**, выполняющие преобразование максимального десятичного номера активного входа в двоичный эквивалент этого номера;
- **неприоритетные**, выполняющие преобразование десятичного номера активного входа в двоичный эквивалент этого номера).

Неприоритетный шифратор преобразует из унитарного кода (прямого или инверсного) в натуральный двоичный код (n-разрядный двоичный код).

Применение шифраторов:

1. сжатие информации (благодаря тому, что число разрядов натурального двоичного кода меньше числа разрядов унитарного кода);
2. ввод данных в ЦК с клавиатуры: ввод числовых данных с клавиатуры выполняется нажатием одной из десяти клавиш, что кодируется в унитарном коде, а ввод этих данных в процессор производится уже в двоичном коде, формируемым шифратором – при нажатии клавиши на одном из входов шифратора появляется логическая 1 и на выходах устанавливается двоичный код, соответствующий символу данной клавиши.

Приоритетный шифратор – на его вход может быть подан **произвольный двоичный код. ПДК** может содержать произвольное число единиц (или нулей для инверсного), расположенных в произвольном порядке. На выходе приоритетного шифратора формируется натуральный двоичный код, определяющий номер позиции приоритетной единицы, т.е. единицы, стоящей в самом старшем разряде.

Приоритетные шифраторы могут использоваться в качестве обычных шифраторов, поскольку унитарный код является частным случаем произвольного. Применяются для преобразователя двоичных чисел из формата с фиксированной запятой в формат с плавающей запятой (порядок определяется по положению старшей единицы), и в аналого-цифровых преобразователях параллельного типа.

Логика работы шифратора 4/2 (Рис.5.17 – 5.19) описывается так:

$$y_0 = x_1 \vee x_3 = \overline{x_0 \vee x_3}$$

$$y_1 = x_2 \vee x_3 = \overline{x_0 \vee x_1}$$

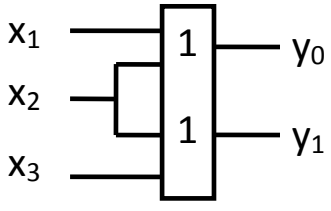


Рис.5.17. Построение шифратора 4/2 на элементах «ИЛИ»

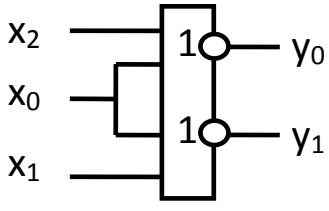


Рис.5.18. Построение шифратора 4/2 на элементах «ИЛИ-НЕ»

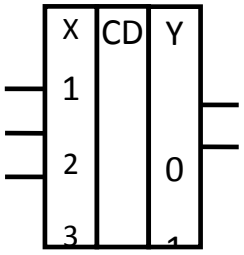


Рис.5.19. УГО шифратора 4/2

Дешифратор (DC – decoder) **–** это элемент, функционально обратный **шифратору**, т.е. комбинационное устройство, преобразующее n-разрядный двоичный код в логический сигнал, появляющийся на том выходе, десятичный номер которого соответствует двоичному коду.

Дешифратор работает следующим образом: двоичное слово $x_{N-1}x_{N-2}...x_0$ поразрядно подается на N входов, тогда на выходе формируется такой код разрядности меньшей или равной 2^N, что разряд, номер которого равен входному слову, принимает значение единица, а все остальные разряды равны нулю. Можно видеть, что максимально возможная разрядность выходного слова равна 2^N. Такой дешифратор называется полным. Если часть входных наборов не используется, то число выходов меньше 2^N, и дешифратор называется неполным.

Часто дешифраторы дополняются входом разрешения работы E (от enable). Если на E-вход поступает единица, то дешифратор функционирует, иначе на выходе дешифратора вырабатывается логический ноль вне зависимости от входных сигналов.

Вариант – дешифратор со входом V (veto). Вход V позволяет не только запрещать работу устройства, но и наращивать его разрядность. Когда на этот вход подается логическая единица, независимо от состояния информационных входов на выходе образуются нули.

Если на входе *V* дешифратора логический ноль, то на выходе, номер которого соответствует двоичному коду адреса, обозначенного на входе, формируется логическая единица, а на других выходах ноли.

Существуют дешифраторы с инверсными выходами, у такого дешифратора выбранный разряд показан нулем.

Логика работы дешифратора описывается системой конъюнкций, пример дан ниже.

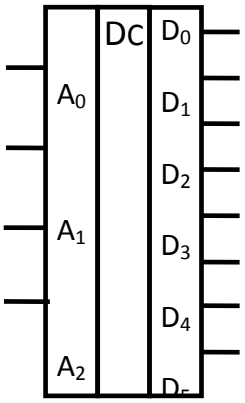


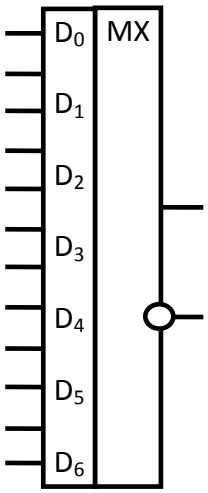
Рис.5.20. УГО дешифратора 3/8

$$D_0 = \overline{V} \cdot \overline{A_2} \cdot \overline{A_1} \cdot \overline{A_0}$$
$$D_1 = \overline{V} \cdot \overline{A_2} \cdot \overline{A_1} \cdot A_0$$
$$D_2 = \overline{V} \cdot \overline{A_2} \cdot A_1 \cdot \overline{A_0}$$
$$D_3 = \overline{V} \cdot \overline{A_2} \cdot A_1 \cdot A_0$$
$$D_4 = \overline{V} \cdot A_2 \cdot \overline{A_1} \cdot \overline{A_0}$$
$$D_5 = \overline{V} \cdot A_2 \cdot \overline{A_1} \cdot A_0$$
$$D_6 = \overline{V} \cdot A_2 \cdot A_1 \cdot \overline{A_0}$$
$$D_7 = \overline{V} \cdot A_2 \cdot A_1 \cdot A_0$$

Логика работы дешифратора 3/8

Мультиплексор (multiplexer – коммутатор, multi – много, plex – сеть) передает сигнал с одного из входов, указанного в адресе, на общий выход, т.е. выполняет роль селектора каналов. Применяются как универсальные

переключатели и устройства, реализующие произвольные логические функции. Могут строиться на основе дешифраторов.



$$y = V \cdot \begin{pmatrix} D_0 \cdot \overline{A_2} \cdot \overline{A_1} \cdot \overline{A_0} \vee D_1 \cdot \overline{A_2} \cdot \overline{A_1} \cdot A_0 \vee \\ D_2 \cdot \overline{A_2} \cdot A_1 \cdot \overline{A_0} \vee D_3 \cdot \overline{A_2} \cdot A_1 \cdot A_0 \vee \\ D_4 \cdot A_2 \cdot \overline{A_1} \cdot \overline{A_0} \vee D_5 \cdot A_2 \cdot \overline{A_1} \cdot A_0 \vee \\ D_6 \cdot A_2 \cdot A_1 \cdot \overline{A_0} \vee D_7 \cdot A_2 \cdot A_1 \cdot A_0 \end{pmatrix}$$

Рис.5.21. УГО мультиплексора Логика работы мультиплексора

Демультиплексор переключает сигнал с единственного входа на один из выходов – тот, который указан в адресе. Демультиплексор состоит из дешифратора, управляемого адресом, и конъюнкторов, управляемых информационным сигналом и выходами дешифратора.

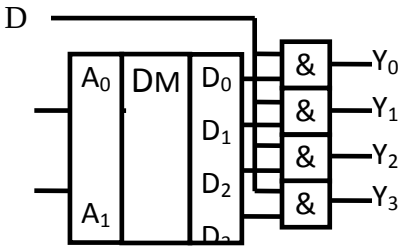


Рис.5.22. Демультиплексор

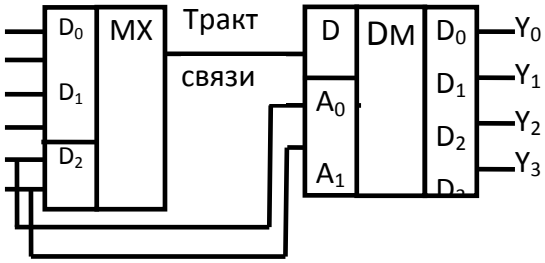


Рис.5.23. Мультиплексированная линия связи

Счётчик – это устройство, на выходах которого формируется двоичный (двоично-десятичный) код, определяемый числом поступивших импульсов. Счётчики могут строиться на Т-триггерах. Основной параметр счётчика – модуль счёта, он определяется как максимальное число единичных сигналов, которое может быть сосчитано счётчиком. Счётчики на схемах обозначают аббревиатурой СТ (от англ. counter).

Таблица 5.7. Классификация счетчиков

по модулю счёта	по направлению счёта	по способу формирования внутренних связей
двоично-десятичные; двоичные; с произвольным постоянным модулем счёта; с переменным	суммирующие; вычитающие; реверсивные;	с последовательным переносом; параллельным переносом; комбинированным переносом; кольцевые.

модулем счёта		

Преобразователи кодов (ПК) делятся на:

весовые – преобразуют информацию из одной системы счисления в другую;

невесовые – преобразуют информацию для дальнейшего отображения.

Пример[3]: **преобразователь двоично-десятичного кода** в код для семисегментных светодиодных индикаторов. На рисунке 5.24 [3] показан фрагмент подключения одного сегмента к выходу схемы с общим эмиттером и приведены начертания первых пяти цифр.

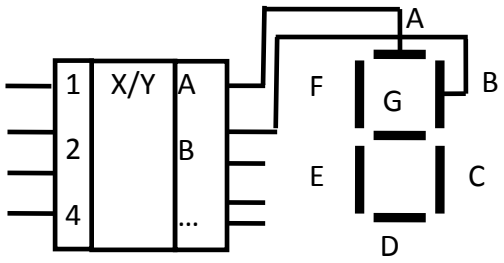


Рис.5.24. Преобразователь двоично-десятичного кода в код для семисегментных светодиодных индикаторов [3]

Литература

1. http://www.znaytovar.ru/gost/2/gost_274391_eskd_oboznacheniya.html
2. *Таненбаум Э., Остин Т.* Архитектура компьютера // 6-е издание. — СПб.: Питер, 2013. — 811 с., илл. — ISBN: 978-5-496-00337-7.
3. *Китаев Ю.В.* Цифровые и микропроцессорные устройства. Конспект по курсу «Электроника и МП«http://de.ifmo.ru/--books/electron/
4. *Шауцкова Л.З.* Информатика. http://book.kbsu.ru
5. *Угрюмов Е. П.* Цифровая схемотехника. СПб, БХВ-Петербург, 2004.
6. *Шило В. Л.* Популярные цифровые микросхемы. М, Радио и связь, 1987.
7. *Дж. Ф. Уэкерли* Проектирование цифровых устройств. М, Постмаркет, 2002.

6. Центральный процессор.

6.1. Программная модель (регистровая структура) процессора

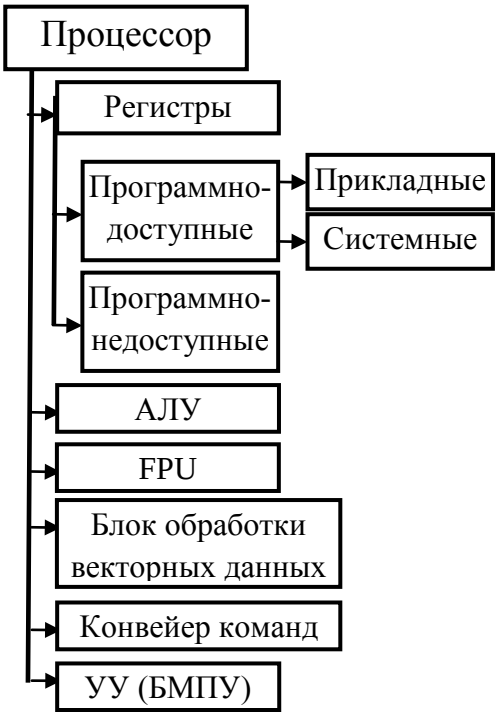


Рис.6.1. Фрагмент схемы рис.4.1, касающийся процессора

Под регистровой структурой процессора понимается набор программно доступных регистров. Поэтому часто используется термин «программная модель процессора». Программная доступность регистров означает, что со стороны программы, т.е. с использованием некоторых машинных команд, может осуществляться обращение к этим регистрам – как по чтению, так и по записи.

Регистровая память – внутренняя память процессора, предназначенная для хранения операндов, адресов и результатов операций.

Включает:

1. Программно доступные регистры:
   1.1.Прикладные;
   1.2.Системные;
2. Программно недоступные регистры (например, регистр команд).

Системные регистры включают в себя следующие типы:

- CR – Control Registers (управляющие регистры);
- Регистры управления памятью (Регистры системных адресов):
  - GDTR – Global Descriptor Table Register (регистр таблицы глобальных дескрипторов);
  - LDTR – Local Descriptor Table Register (регистр таблицы локальных дескрипторов);

    - IDTR – Interrupt Descriptor Table Register (регистр таблицы дескрипторов прерываний);
    - TR – Task Register (регистр задачи);
- DR – Debug Registers (Регистры отладки);
- TR – Test Registers (Регистры проверки);
- GPR – General Purpose Registers – предназначены для хранения как операндов, так и адресов.

Системные по критерию своей функциональной специализации регистры делятся на:

- Специализированные регистры;
- Регистры с функциональной специализацией;
- Универсальные регистры.

Функциональная специализация регистра позволяет сократить длину длину машинного кода данной операции за счет неявной адресации (операнд или адрес подразумевается по умолчанию). В «свободное от основной работы время» такой регистр может быть использован для выполнения других операций, но тут уже требуется явная адресация.

Сегментные регистры. Предназначены для сегментирования памяти. Содержат начальные (базовые) адреса 4-х сегментов памяти по наименованию регистров:

- Code Segment (сегмент кода);
- Stack Segment (сегмент стека);
- Data Segment (сегменты данных);
- Extra Segment (дополнительный сегмент);

Физический адрес формируется как сумма базового адреса сегмента и внутрисегментного смещения (Offset). При суммировании компонент первая составляющая сдвигается влево на 4 разряда. Получается двадцатиразрядная сумма – это и есть физический адрес. Он выставляется на внешнюю шину адреса.

Если используется мультиплексированная шина адрес/данные, то адрес и данные передаются по одним и тем же линиям, но в разные моменты времени.

В сегментных регистрах находятся старшие 16–ти разрядные компоненты 20–ти разрядных базовых адресов сегментов. В соответствии с этим подходом, границы сегментов физической памяти выравниваются на 16–ти байтную границу (4 младших нуля в адресе), которая называется **границей параграфа.** Выбор внутрисегментного смещения определяется видом обращения к памяти.

Регистр IP (Instruction Pointer) или PC (Program Counter). Термин «программный счетчик» PC неудачен, так как на самом деле этот регистр ничего не считает, а указывает какая команда должна выполняться следующей. Поэтому лучше использовать термин «указатель инструкций» и аббревиатуру IP.

Содержимое IP используется процессором (фон Неймановской архитектуры) при выборке очередной команды из памяти. В момент выполнения команды, содержимое IP определяет адрес следующей команды. Термин “следующая”, означает последовательность команд не столько в смысле их выполнения, сколько в смысле их положения в памяти. Например, при выполнении команд перехода, вызовов, возвратов, содержимое IP изменяется на значение адреса перехода, вызова или возврата.

Регистр FR (Flag Register). Флаги это особые отметки, делятся на:

1. Управления – они используются для управления режимом работы процессора:
   a. IF – Interrupt Flag;
   b. TF – Trace Flag;
   c. DF – Direction Flag.
2. Арифметические – они фиксируют признаки результата. Их содержимое используется при выполнении команд условных переходов.

6.2. Центральный процессор (тракт данных)

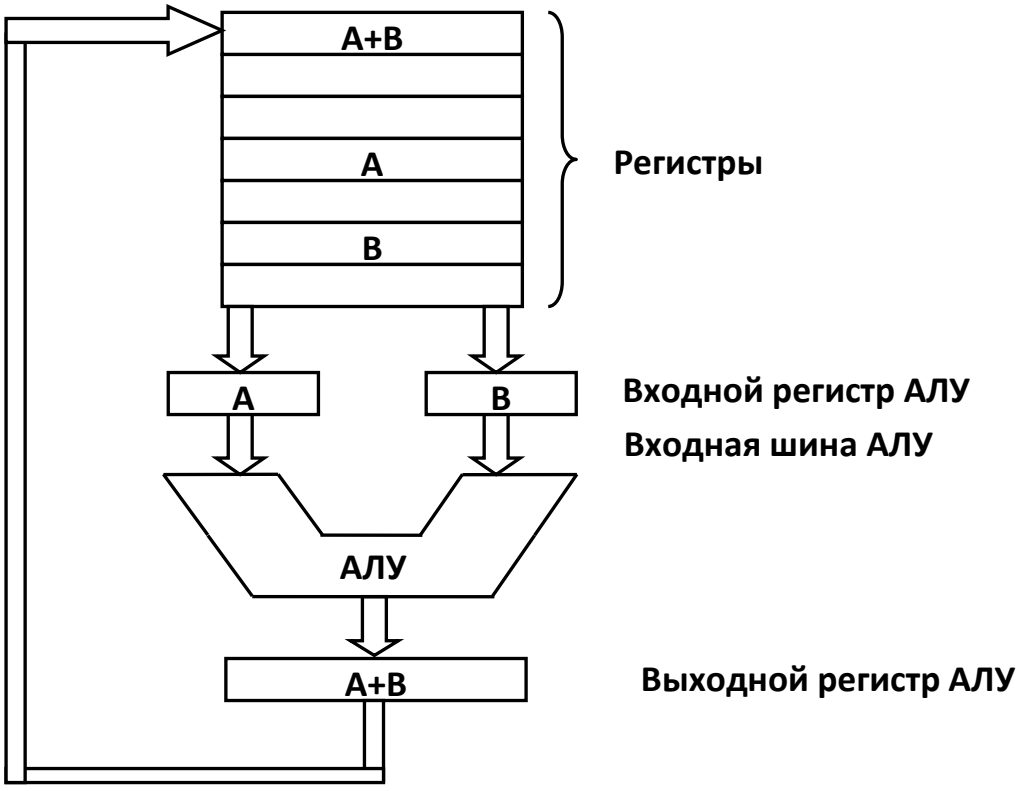


Рис.6.2. Принципиальная схема тракта данных (устройство управления не показано)

Основные компоненты центрального процессора (см. рис.4.5):

- Устройство управления – его задача вызов (выборка) команд из памяти, их определение и выполнение;

- Регистры – осуществляют временное хранение операндов и состояний процессора;
- АЛУ – выполняет операции над операндами и помещает результат в выходной регистр;
- Шины.

Команды делятся на два класса:

1. Регистр – память. Вызывают слова из памяти и помещают в регистр. И наоборот – из регистров в память.

2. Регистр – регистр. Вызывают операнды из регистров, помещают во входные регистры АЛУ , выполняют над ними операции и помещают результат обратно в регистр. Это – цикл тракта данных.

Цикл выполнения команд ЦП

1. Вызов команды из памяти и перенос её в регистр команд;
2. Изменение положения регистра IP (указателя инструкций или счетчика команд) для указания следующей команды;
3. Определение типа вызванной команды;
4. Если команда типа «память – регистр», то определение адреса слова в памяти;
5. Перенос слова (или набора слов) из памяти в регистр ЦП;
6. Выполнение команды;
7. Переход к шагу 1 для выполнения следующей команды.

Цикл выполнения команд ЦП может выполняться как аппаратно, так и программно (интерпретатором, 1951 г. – появление интерпретатора). Следовательно, чем проще «железо», тем сложнее «софт» и наоборот.

Достоинства простых (в смысле «железа») ЦК с интерпретаторами (т.е. сложных команд):

1. Возможность исправления неправильно реализованных команд «на месте» и компенсации ошибок «железа» на уровне «софта»;
2. Возможность добавления новых команд при минимальных затратах и без переделки «железа»;
3. Возможность разработки, проверки и документирования сложных команд (обеспечивается структурированной организацией).

6.3. RISC и CISC архитектуры.

В связи с логической эквивалентностью аппаратного и программного обеспечения упомянем проблему семантического разрыва. В контексте изучаемой нами темы под семантическим разрывом понимается различие принципов, положенных в основу программного и аппаратного обеспечения. Это проблема возникает на самых разных уровнях: на уровне архитектуры компьютера – между средствами формулировки операций и операндов на языках высокого уровня и аппаратных средств, на уровне прикладных программ и языковыми средствами, на уровне операционной

системы, на уровне архитектуры процессора – между аппаратными средствами и тем, на что они опираются.

Было замечено, что примерно 20% команд используются в 80% случаев, и наоборот (феномен 20/80). Поэтому возникла идея некоторые, наиболее часто используемые команды «зашить» на аппаратном уровне – тогда они будут выполняться быстрее за счет отказа от использования интерпретатора. Так появились две архитектуры:

RISC – Reduced Instruction Set Computer (1982)

CISC – Complex Instruction Set Computer

RISC – небольшое число простых команд, каждая из которых выполняется за один цикл тракта данных. RISC команды выполняются быстрее, чем CISC за счет того, что они не интерпретируются.

Но RISC компьютеры несовместимы с другими.

Примеры:

RISC – ARM (Acorn RISC Machine, затем Advanced RISC Machine), AVR (A(lf-Egil Bogen) and V(egard Wollan) RISC processor)

CISC – Intel (x86 до 486)

Intel 486 и более поздние модели содержат RISC ядро, выполняющее ограниченный набор самых простых и распространенных команд.

Принципы RISC архитектуры:

1. Все команды должны выполняться только аппаратно (выигрыш во времени за счет отказа от интерпретации);
2. Следует стремиться к максимальному числу запуска команд в единицу времени (MIPS) и, следовательно, параллелизму их выполнения;
3. Следует стремиться к уменьшению числа форматов команд (к их единообразию) для упрощения их декодирования (декодирование - ресурсозатратно!);
4. К памяти обращаются только команды загрузки и сохранения (большое время обращения); все остальные перемещают операнды только между регистрами;
5. Поэтому регистров должно много.

Рис.6.3. Сравнение CISC и RISC архитектур

Основное достоинство CISC архитектуры – её универсальность. Области применения – универсальные компьютеры без особых требований к энергопотреблению.

Основное достоинство RISC архитектуры – низкое энергопотребление. Области применения – те области, где критичны энергопотребление и размеры. Например, мобильные устройства, бортовые специализированные процессоры.

6.4. Методы обеспечения параллелизма на уровне команд

Можно выделить две группы методов обеспечения параллелизма: на уровне команд и на уровне процессора.

Параллелизм на уровне команд	Параллелизм на уровне процессора
запуск большого количества команд в единицу времени: • Конвейер команд; • Суперскалярные архитектуры (повышение производительности в разы, до 10 раз).	несколько процессоров, работающих «в параллель».

Конвейер команд – основная идея заключается в увеличении числа шагов выполнения команды. Первый «подход к теме» - 1959 г., IBM – «выборка с упреждением», буфер выборки с упреждением. Команды заранее вызываются из памяти и хранятся в специальном наборе регистров - буфере. Команда выполняется за два шага:

1. Выборка команды;
2. Выполнение команды.

В конвейере команда обрабатывается за большее количество шагов, каждый из которых реализуется своим аппаратным обеспечением; эти аппаратные компоненты могут работать параллельно.

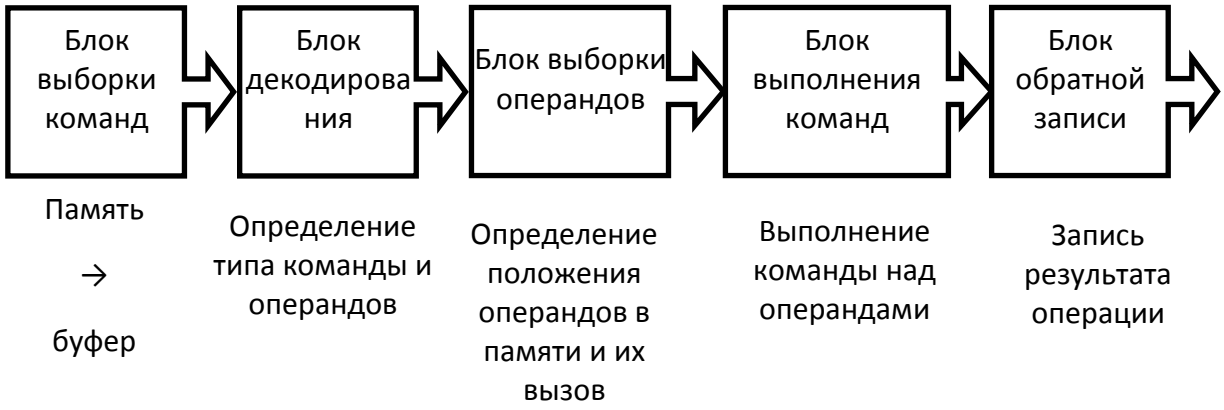


Рис.6.4. Конвейер команд

Дальнейшее развитие этой идеи – появление двух конвейеров: главный конвейер (u-конвейер) выполняет любые команды, а вспомогательный конвейер (v- конвейер) – только простые.

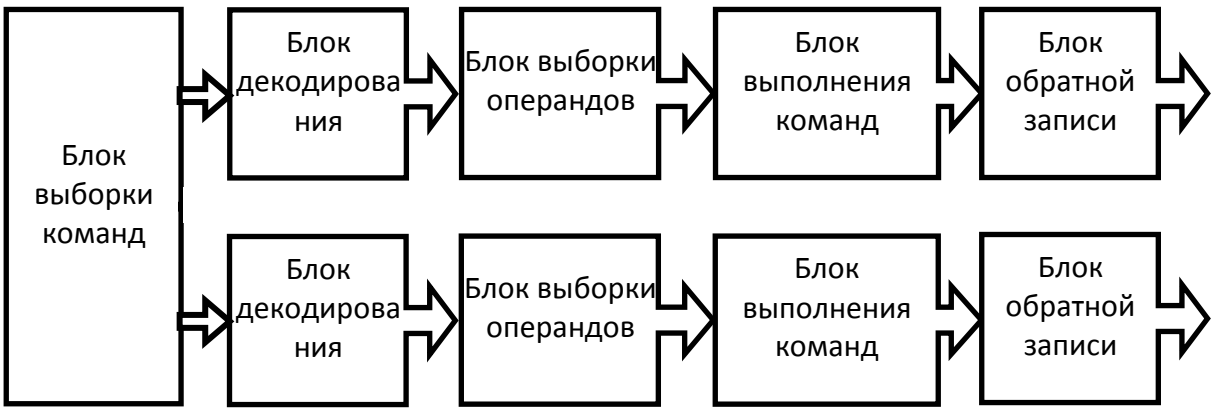


Рис.6.5. Развитие конвейера команд

Но в такой архитектуре возможны конфликты за ресурсы. Поэтому необходима проверка совместимости команд для их параллельного выполнения.

Увеличение числа конвейеров ведет к усложнению аппаратного обеспечения. Дальнейшее развитие – суперскалярная архитектура – один конвейер с большим числом функциональных блоков.

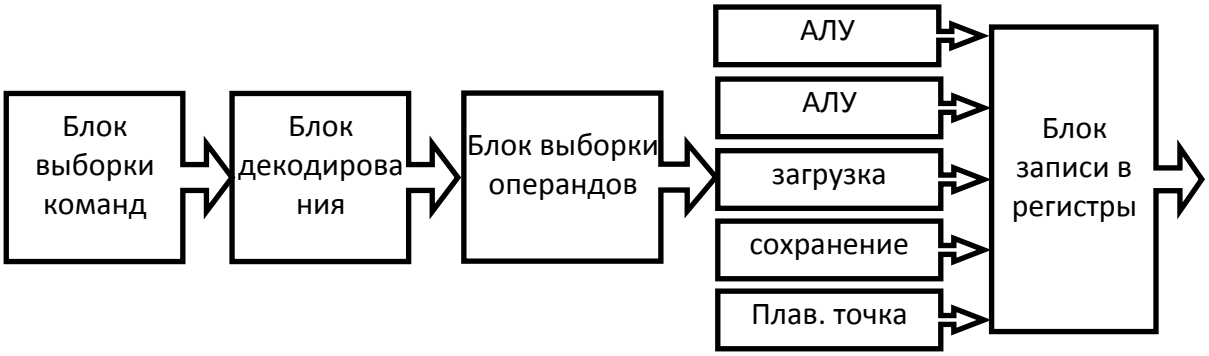


Рис.6.6. Суперскалярная архитектура

6.5. Структура и форматы машинных команд

Структура задает поля (основные части) машинной команды. Например:

- поле кода операции (КОП или OpC);
- адрес базового или индексного регистров (Base, Index);
- смещение (Disp);
- непосредственный операнд (IMM);

Формат определяет разрядность как всей команды, так и её полей.

Операционная часть (код операции – КОП или OpC) – задает тип выполняемой операции;

Адресная часть: указывает адреса операндов, результата и следующей команды. По количеству адресов команды делятся на:

Четырехадресная (код операции и адреса:2-х операндов, результата и следующей команды);

Трехадресная (код операции и адреса: 2-х операндов и результата);

Двухадресная (код операции и адреса:2-х операндов, а результат записывается по адресу второго операнда);

Одноадресная (код операции и адрес одного операнда, второй операнд и результат хранятся в аккумуляторе – регистре результата);

Безадресная (код операции и расширение кода операции – в компьютерах со стековой организации памяти).

Виды адресования:

- **Прямое** – указание адреса операнда, результата или перехода.
- **Косвенное** – указание адреса ячейки памяти, в которой сохраняется требуемый адрес. Упрощает организацию циклов, работу с подпрограммами и позволяет расширить адресное пространство.

Для указания вида адресации используется бит с адресом 11 (БА). При прямой адресации он устанавливается в 0, при косвенной – в 1.

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="4"></td><td></td><td colspan="3"></td><td colspan="4"></td><td colspan="4"></td></tr>
<tr><td colspan="4">КОП</td><td>БА</td><td colspan="12">Поля адресов</td></tr>
<tr><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td></tr>
<tr><td colspan="4">КОП</td><td colspan="12">Расширение кода операции</td></tr>
<tr><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td></tr>
<tr><td colspan="4">КОП</td><td colspan="4">Признак ввода-вывода</td><td colspan="12">Адреса устройств ввода - вывода</td></tr>
</table>

Рис.6.7. Форматы команд: верхняя строка – адресная, средняя – безадресная, нижняя – ввода – вывода

6.6. Структура процессора и выполнение команд

На рис. 6.8. дана регистровая структура процессора. На схеме показаны:

РА - Регистр адреса. Хранит адрес ячейки основной памяти вплоть до завершения операции с этой ячейкой.

УкС - Указатель стека – регистр, где хранится адрес вершины стека. Стек **–** участок основной памяти, обычно в области наибольших адресов. Заполнение стека происходит в сторону меньших адресов. **Вершина стека** — ячейка последней записи. При занесении в стек содержимое **УкС** сначала уменьшается на единицу, затем используется как адрес записи. Эта ячейка становится новой вершиной стека. Считывание из стека происходит из ячейки, на которую указывает текущий адрес в **УкС**, после чего содержимое **УкС** увеличивается на единицу. Вершина стека опускается, а считанное слово считается удаленным из стека. **НО!** Физически считанное слово остается в ячейке, реально оно заменяется при следующей записи **СК – Счетчик команд.** Адрес очередной команды вычисляется как сумма длины выполняемой команды, представленной числом занимаемых ею ячеек, к адресу ячейки, из которой была считана текущая команда.

РК – Регистр команды. Счетчик команд определяет адрес команды в памяти, но не дает информации о ней самой. Для выполнения команды она

извлекается из памяти и помещается в **РК.** Этот этап называется **выборкой команды**. Процессор видит только команды, находящиеся в **РК**. В **РК** команда хранится в течение всего времени её выполнения.

.

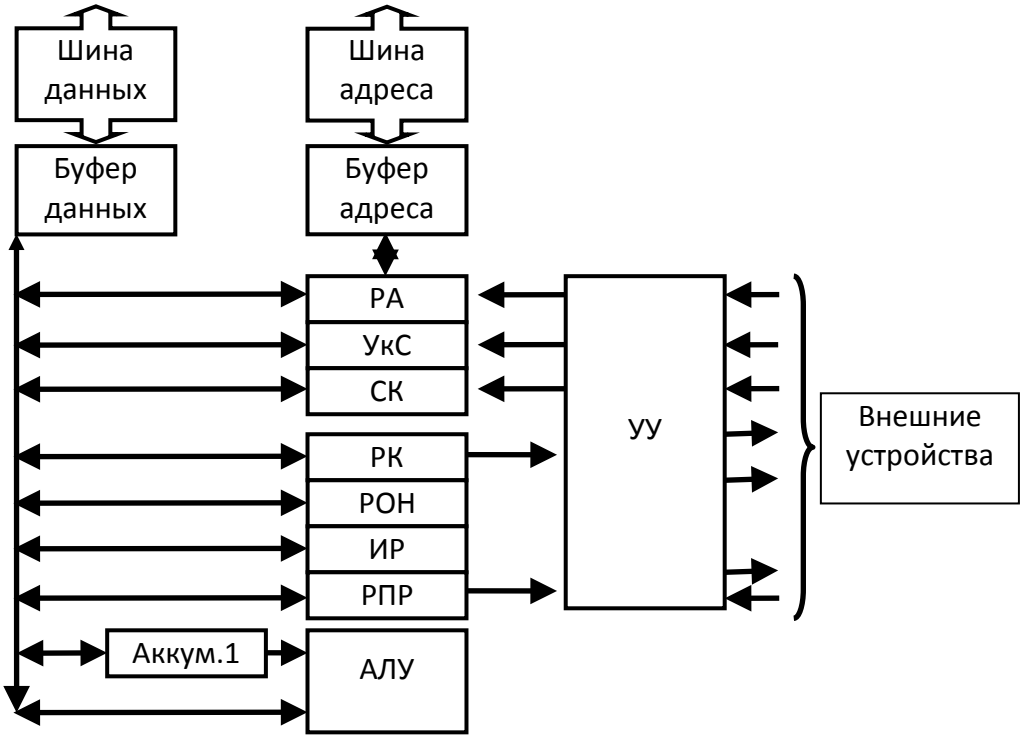


Рис.6.8. Структура центрального процессора

РОН – Регистры общего назначения предназначены для временного хранения операндов и результатов вычислений. Это самый быстрый, но и минимальный по емкости тип памяти, который иногда называют сверхоперативным запоминающи устройством — **СОЗУ**. В **CISC** количество **РОН** обычно невелико, в **RISC** их число больше, чем в **CISC.**

ИР – Индексные регистры служат для формирования адресов операндов при реализации циклических участков программ.

РПР – **Регистр признака результата** для фиксации и хранения признака, характеризующего результат последней выполненной операции:

- о равенстве результата нулю;
- о знаке результата;
- о возникновении переноса из старшего разряда;
- переполнении разрядной сетки и т. д.

Содержимое **РПР** обычно используется **УУ** для реализации условных переходов по результатам операций АЛУ. Под каждый из возможных признаков отводится один разряд РПР.

Буфер данных компенсирует разницу в быстродействии запоминающих устройств и устройств, выступающих в роли источников и потребителей хранимой информации. В **буфер данных** при чтении заносится содержимое ячейки ОП, а при записи — помещается информация, подлежащая сохранению в ячейке ОП.

Буфер адреса позволяет компенсировать различия в быстродействии оперативной памяти и других устройств ЦК.

Аккумулятор - специально выделенный регистр, его функции:

- в него предварительно загружается один из операндов;
- может хранить результат предыдущей команды;
- в него заносится результат очередной операции;
- производит операции ввода и вывода.

Аккумуляторов может быть несколько. Аккумулятор можно отнести как к АЛУ, так и к УУ, его можно рассматривать как один из регистров общего назначения.

Рассмотрим организацию работы центрального процессора при выполнении четырехадресной, одноадресной и безадресной команд.

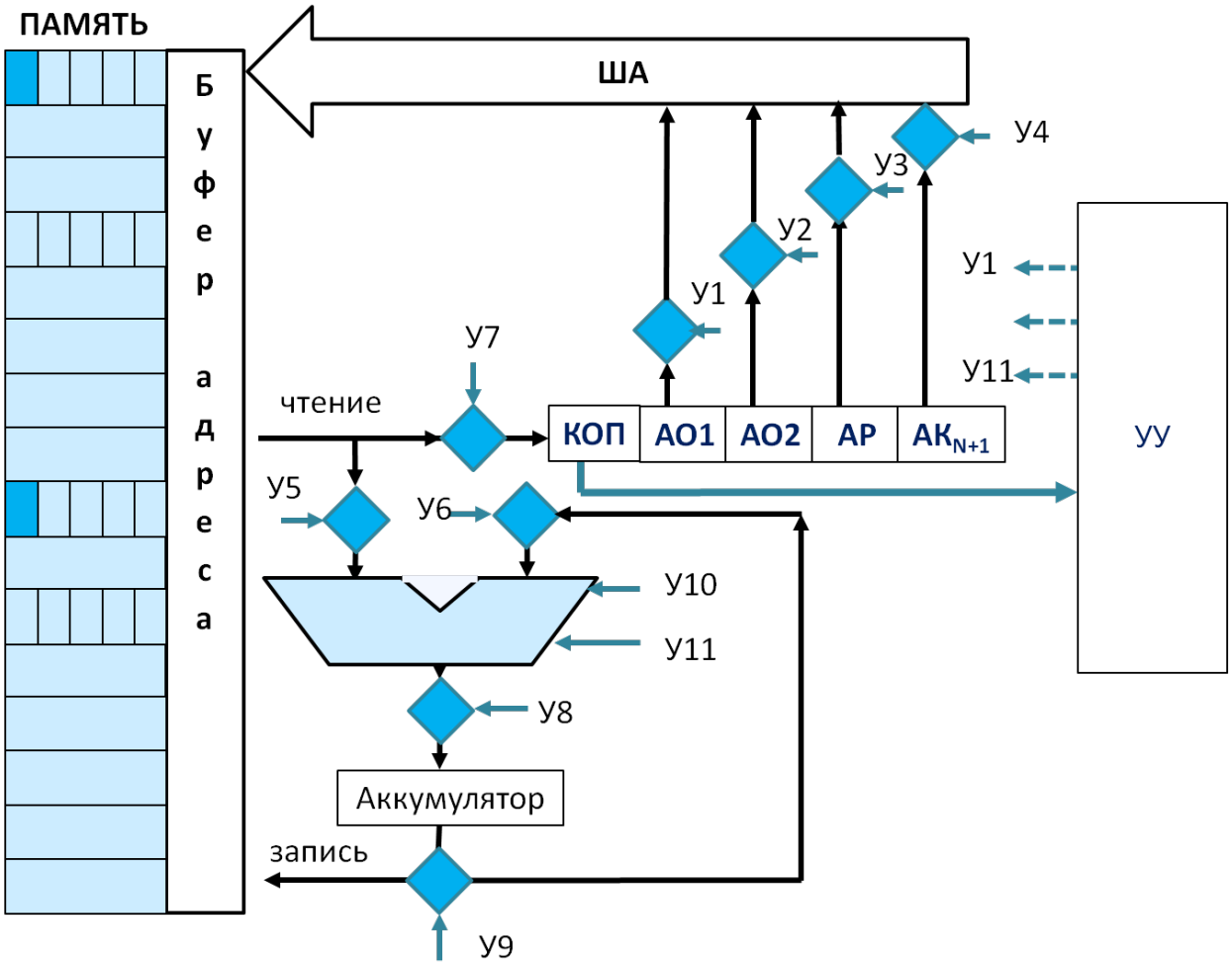


Рис.6.9. Организация работы центрального процессора при выполнении четырехадресной команды

Выполнение четырехадресной команды состоит из следующих этапов:

1.1. код операции (КОП) поступает в устройство управления (УУ);

1.2. УУ формирует управляющие сигналы (команды) У1, У5,У8 и У10 для вызова из памяти операнда 1, его прохождения через АЛУ и записи в АКК;

1.3. затем УУ формирует управляющие сигналы (команды) У2, У5,У6 и У11 для вызова из памяти операнда 2, и выполнения АЛУ операции (У11);

1.4. УУ формирует У8 – сигнал записи результата в аккумулятор;
1.5. УУ формирует У3 и У9 для записи результата в ячейку памяти АР;
1.6. УУ формирует У4 и У7 для записи в РК следующей команды.

Возникает вопрос – нельзя ли упростить эту схему? Можно, если:

1. Разместить команды последовательно – убрать поле «Адрес следующей команды». Но тогда на схеме появятся: счетчик команд (IP) и команды перехода.

2. Если результат операции не пересылать в память, а использовать как операнд следующей операции, то достаточно одноадресных команд: арифметических и логических, пересылки (память – аккумулятор и обратно), перехода (передачи управления), ввода-вывода. Тогда нужен **аккумулятор** – специальный регистр для хранения результата.

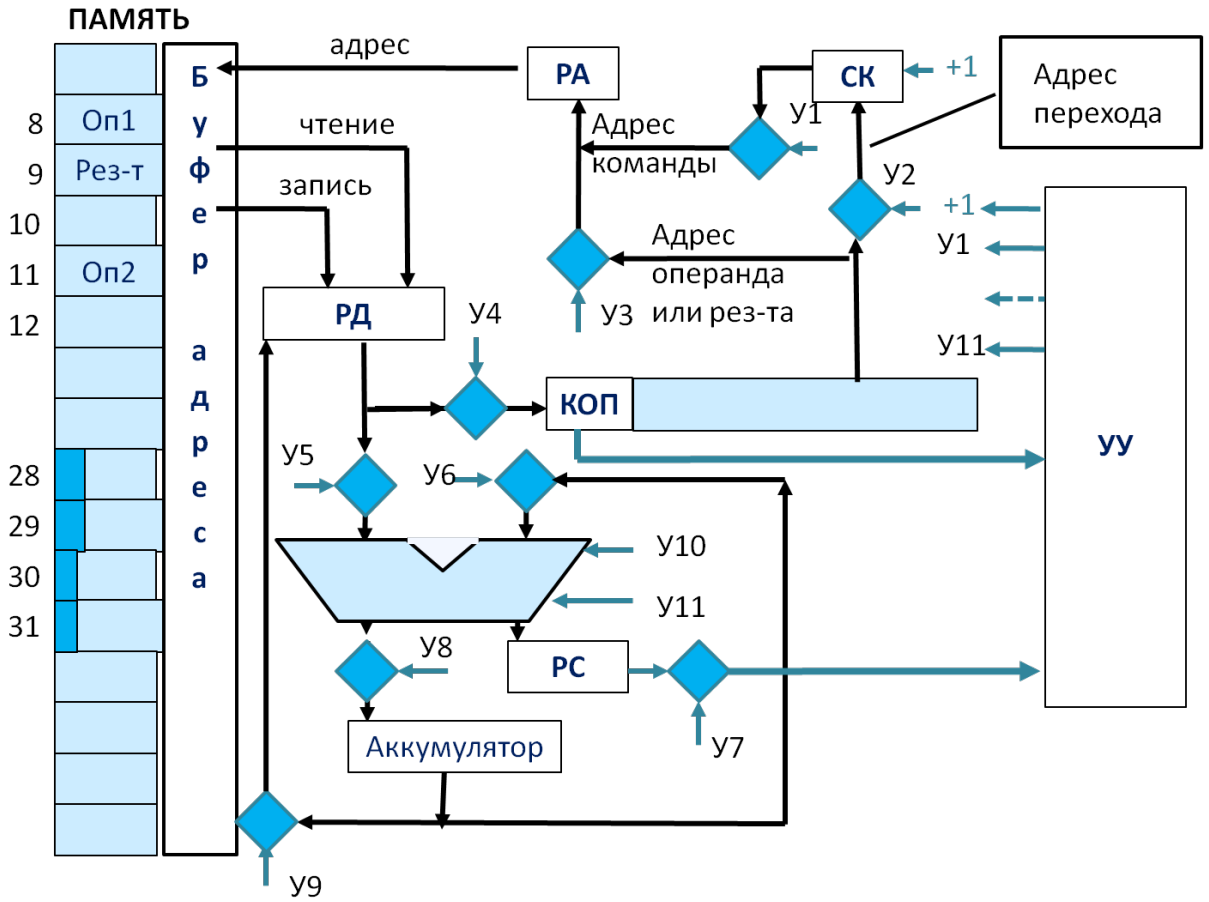


Рис.6.10. Организация работы центрального процессора при выполнении одноадресной команды

Последовательность действий при выполнении одноадресной команды следующая:

Первый операнд (Оп1) хранится в аккумуляторе, В СК хранится адрес команды, например, 28;

1. УУ формирует У1 – адрес команды 28 из СК считывается в РА, в памяти из ячейки с адресом 28 считывается команда и помещается в РК (У4);
2. СК наращивается на +1;

3. УУ считывает из РК КОП и переходит к её выполнению;
4. Из РК считывается (У3) и пересылается через РА в память адрес операнда 2 (11), значение из ячейки памяти 11 переписывается в РД;
5.Выполняется операция (У5, У6, У11), результат переписывается с выхода АЛУ в акк. (У8).

Далее переход к безадресным командам (нуль-адресная адресация). Для этого используется **стековая организация памяти** и **постфиксная нотация** – инверсная запись математических выражений.

Стековая организация памяти.

Обращение к ячейкам стековой памяти производится последовательно с помощью специального **указателя стека** (УС). УС определяет рабочую (на данный момент) ячейку. Каждая ячейка СЗУ снабжена тэгом – специальным признаком хранимой информации.

Одноместная операция всегда использует вершину стека и помещает результат в вершину.

Двухместная операция всегда использует в качестве операндов содержимое вершины и подвершины, помещает результат в вершину.

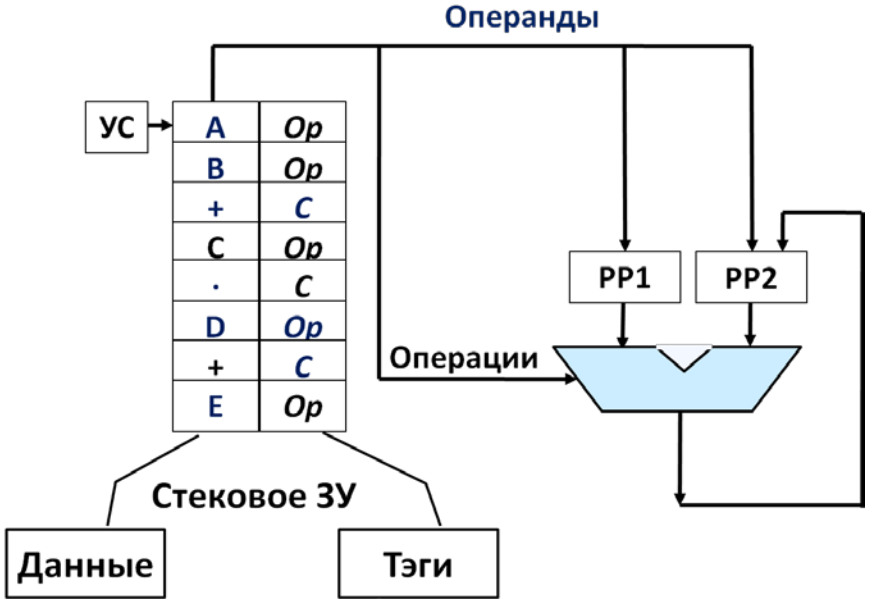


Рис.6.11. Организация работы центрального процессора при выполнении безадресной команды

Достоинство безадресной системы команд – высокое быстродействие.

Слабая сторона безадресной системы команд – необходимость инверсной или постфиксной записи, усложняющей работу программиста.

Для нас привычна т.н. инфиксная запись математических выражений, в которой знак операции расположен между операндами. RPN – ПОЛИЗ – ПОЛьская Инверсная Запись или ОПН – Обратная Польская Нотация, ОБН – Обратная Бесскобочная Нотация. Названа в честь польского логика и математика Яна Лукасевича, разработавшего в 20-х годах XX века

префиксную нотацию, в которой знак операции располагался перед операндами. Обратная польская нотация или постфиксная запись, в которой знак операции расположен после операндов, разработана австралийским философом Чарльзом Хэмблином в 50-х годах. Ниже даны примеры инфиксной и постфиксной записи одного и того же выражения:

$\big((A+B)\times C\big)\big/\big(D+E\times F-J\big)$ – инфиксная нотация,

$A\ B+C\times D\ E\ F\times+J-/$ – постфиксная нотация.

Достоинство постфиксной нотации – отсутствие проблемы приоритета операций при чтении записи и, как следствие, отсутствие скобок.

Преобразование инфиксной в постфиксную нотацию осуществляется по алгоритму Дейкстры (E.W. Dijkstra).

Процессор при нуль-адресной адресации работает следующим образом (на примере вычисления выражения, приведенного выше):

Шаг 1.Ввод первого операнда (*A*);
Шаг 2. Ввод второго операнда (*B*);
Шаг 3. Ввод операции (+);
Шаг 4. Выполнение операции (*A*+*B*);
Шаг 5. Запись результата операции;
Шаг 6. Ввод третьего операнда (*C*);
Шаг 7. Ввод операции (×);
Шаг 8. Выполнение операции (*A*+*B*)×*C*;
и так далее, до выполнения всей записанной последовательности.

Литература.

1. *Таненбаум Э., Остуртин Т.* Архитектура компьютера // 6-е издание. — СПб.: Питер, 2013. — 811 с., илл. — ISBN: 978-5-496-00337-7.
2. *Цилькер Б.Я., Орлов С.А.* Организация ЭВМ и систем // Учебник для вузов. — СПб.: Питер, 2004. — 668 с.: ил. ISBN 5-94723-759-8
3. *С. А. Майоров, Г. И. Новиков* Принципы организации цифровых машин // Машиностроение, 1974. 434 С.
4. *Карцев М.А.* Архитектура цифровых вычислительных машин // М.: Наука, 1978. – 295 С.
5. http://vssit.ucoz.ru/index/0-4 электронный ресурс
6. У. *Столлингс.* Структурная организация и архитектура компьютерных систем, 5-е изд. М.: Вильямс, 2002, 896 с.
7. *Чекменев С.Е.* Архитектура вычислительных систем. Конспект лекций.
8. *Китаев Ю.В.* Конспект по курсу "Электроника и VG" – цифровые и микропроцессорные устройства // http://de.ifmo.ru/--books/electron/ электронный ресурс.

7. Запоминающие устройства (Память)

7.1. Классификация компьютерной памяти

По критерию расположения относительно элементов архитектуры компьютера можно выделить следующие виды компьютерной памяти:

1. **Процессорная память** – это регистры команд 1-го уровня, ониразмещены на одном кристалле с процессором. РОН – часть центрального процессора (сверхоперативная (СОЗУ) или кэш-память), она проигрывает по емкости оперативной памяти, но превосходит её по быстродействию;
2. **Внутренняя память** – регистры расположены на одной плате с процессором (кэш память 2-го и последующих уровней, а также основная память **ОП** и **ПЗУ**); **ОП** или **ОЗУ** обеспечивают и чтение, и запись, а **ПЗУ** – только чтение, т.к. для записи требуется особая процедура – «прожиг»; ПЗУ предназначено для хранения информации, подлежащей обязательному сохранению при выключении компьютера: стандартных программ, констант, таблиц символов и т.п..
3. **Внешняя память** – ЗУ большой емкости, расположенные отдельно от основной платы, связь с этими устройствами организуется посредством контроллеров: магнитные и оптические диски, твердотельная память, etc.

Первые два вида памяти иногда объединяют вместе, называя внутренней памятью.

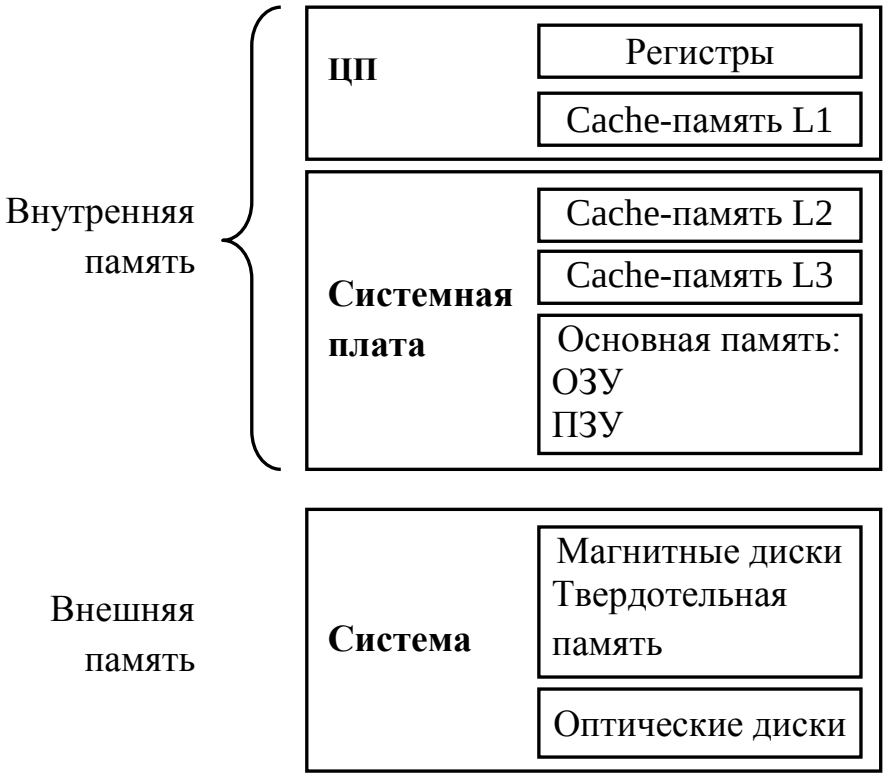


Рис.7.1. Классификация запоминающих устройств по критерию их положения относительно основных элементов архитектуры компьютера

Быстродействие памяти определяется следующими характеристиками:

1. **Время доступа** (интервал от занесения адреса до занесения в память);

2. **Длительность цикла** или период обращения. При произвольном доступе это минимальное время между двумя последовательными обращениями к памяти, определяется как время доступа плюс дополнительное время, обусловленное затуханием сигналов на линиях или временем, необходимым для восстановления считанных данных;

3. **Скорость передачи** (в память или из нее).

Различают следующие методы доступа к памяти:

1. **Последовательный** – для доступа к нужному элементу необходимо прочитать все предшествующие ему элементы (примеры: стриммер, ленточный магнитофон, перфолента);

2. **Прямой** – каждая запись имеет свой адрес, который отражает ее физическое размещение на носителе, обращение к памяти осуществляется к началу записи с последовательным доступом к последующим единицам (например, магнитный или лазерный диск);

3. **Произвольный** – каждая ячейка имеет уникальный физический адрес (например, **flash** память) – запоминающие устройства с произвольной выборкой (**RAM** – Random Access Memory);

4. **Ассоциативный** – доступ не по адресу, а по содержимому – иначе называется контекстный доступ или контекстно-адресуемая память.

RAM (Random Access Memory) — «память с произвольным доступом». Аббревиатурой RAM часто обозначают основную память, называемую также оперативным запоминающим устройством **ОЗУ**.

ROM (Read-Only Memory) — «постоянное запоминающее устройство» (ПЗУ), термин **ПЗУ** относится ко всем видам энергонезависимой памяти, включая и перезаписываемые.

Выделяют следующие виды **RAM** или **ОЗУ**: статическую (**SRAM**), динамическую (**DRAM**), регистровую (**RG**). Более детально:

SRAM (Static RAM) —статическое ОЗУ. Энергозависимая память, регенерации не требующая, но более дорогая и менее емкая, чем DRAM.

DRAM (Dynamic RAM) —память, которая требует постоянного восстановления (регенерации) своего содержимого даже при включенном питании. Динамическое ОЗУ или **ЗУПВ** («запоминающее устройство с произвольной выборкой»), последний термин применяется обычно к динамической разновидности RAM.

SDRAM — (Synchronous DRAM) «синхронная DRAM». Отличается от простой **DRAM** наличием специального логического блока и двухбанковой структурой. Все операции записи/чтения синхронизированы с основным тактовым сигналом.

VRAM (Video RAM) — «видео RAM». Специально разработанная для видеоадаптеров разновидность DRAM с двухпортовой организацией, допускающей одновременное обращение к ней от двух разных устройств.

NRAM (Nano RAM) – энергонезависимая память на нанотрубках.

FRAM, FeRAM (Ferroelectric RAM) — экспериментальная разновидность энергонезависимой памяти на основе ферроэлектрического принципа хранения информации.

MRAM (Magnetic RAM) — экспериментальная разновидность скоростной энергонезависимой памяти на основе магниторезистивного эффекта.

NVRAM (Nonvolatile RAM) — «энергонезависимая RAM» (от латинского volatilis -летучий). В принципе охватывает все разновидности EPROM и EEPROM (в том числе и Flash-память).

Виды **ROM** или **ПЗУ**:

OTPROM (One Time Programmable ROM) — «однократно программируемое ПЗУ». Разновидность OTPROM – «масочное ПЗУ», оно программируется («прошивается») изготовителем, пользователь его перепрограммировать не может.

PROM (Programmable ROM) – «программируемое ПЗУ» (ППЗУ) – иногда говорят об однократно программируемых пользователем ЗУ.

EPROM (Erasable Programmable ROM) — «стираемая/программируемая ROM — ППППЗУ, «перепрограммируемое ПЗУ». Иногда употребляется, как синоним UV-EPROM.

EEPROM (Electrically Erasable Programmable ROM) — «электрически стираемое перепрограммируемое ПЗУ», **ЭСППЗУ**.

UV-EPROM (Ultra-Violet EPROM) — «ультрафиолетовая EPROM», УФППЗУ. Исторически это была первая коммерческая разновидность EPROM, операция стирания в этом устройстве производилась коротким облучением ультрафиолетом через специальное окошко – поэтому flash.

Flash memory — изначально термин обозначал новую разновидность EEPROM, в которой чтение/запись для ускорения процесса производятся сразу целыми блоками. На сегодня это фактически синоним **EEPROM**, термин общеупотребим и обозначает любые разновидности **ЭСППЗУ**.

Необходимо упомянуть также программируемые логические матрицы и устройства различных типов (**PLM, PML, PLA, PAL, PLD, FPGA** etc.), включающие в себя широкий выбор логических элементов и устройств на одном кристалле.

Элемент памяти (**ЭП**) реализуется посредством триггера или транзистора с плавающим затвором (flash-память). Ячейка памяти (**ЯП**) – это упорядоченный набор ЭП.

Статическая память (**SRAM**) обычно строится на D-триггерах – защелках.

Динамическая память (**DRAM**), обеспечивающая большую емкость по сравнению с **SRAM,** строится на основе конденсаторов. В силу того, что размер конденсатора на интегральной микросхеме меньше размера D-триггера, при одинаковых геометрических размерах емкость динамической памяти обычно выше, чем емкость статической. Вместе с тем, схема

DRAM сложнее, т.к. используется мультиплексирование адресных входов и более сложная схема управления, включающей две адресные линии (старшую и младшую), мультиплексоры и демультиплексоры, а также схему регенерации информации, необходимость в которой обусловлена тем, что конденсаторы, в отличие от триггеров, со временем разряжаются.

Перепрограммируемое ПЗУ (или репрограммируемое – РПЗУ). РПЗУ допускают многократную перезапись информации, в том числе, перепрограммирование, пользователем. Для этого при построении РПЗУ используется соответствующая элементная база, основанная на МОП транзисторах с "плавающим затвором". В режиме записи для записи нуля на ЭП подается электрический импульс. Стирание информации производится либо УФ-излучением (**EPROM**), либо электрически (**EEPROM**). При стирании все ячейки переводятся в состояние "1". Срок хранения информации достигает нескольких лет.

FLASH-память строится на основе МДП-транзисторов. МДП-транзистор (**М**еталл – **Д**иэлектрик – **П**олупроводник) или МОП-транзистор (**М**еталл – **О**ксид – **П**олупроводник) – полевой транзистор с изолированным затвором рис.7.2. Работа МДП-транзистора основана на управлении поверхностными свойствами полупроводника за счет изменения потенциала затвора.

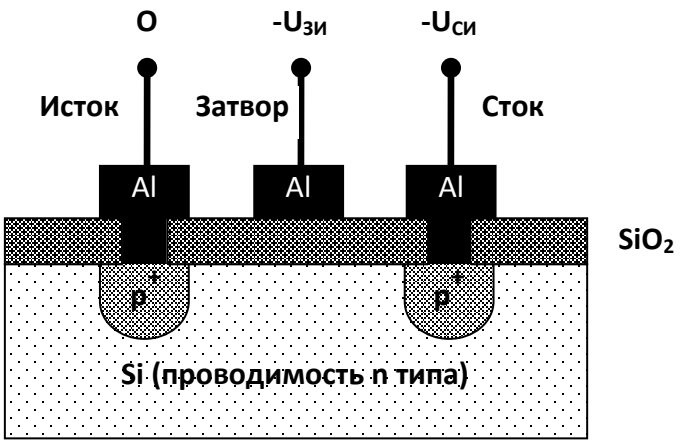


Рис.7.2. МДП-транзистор с индуцированным каналом

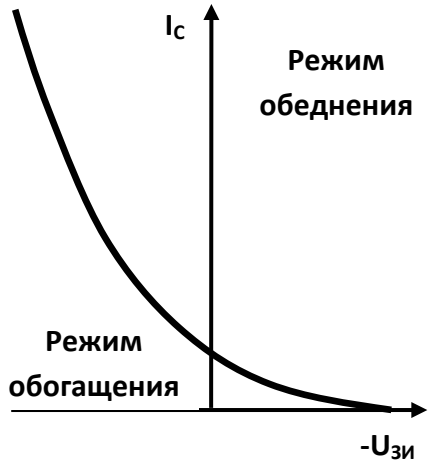


Рис.7.3. Сток-затворная характеристика МДП-транзистора со встроенным каналом

Выделяют два типа МДП-транзисторов: со встроенным каналом проводимости между истоком и стоком, и с индуцированным. В первом случае при отсутствии напряжения на затворе между истоком и стоком существует область, содержащая носители заряда – это канал проводимости. Изменение напряжения на затворе позволяет увеличивать или уменьшать число носителей в этой области, т.е. усиливать или ослаблять ток, вплоть до его полного прекращения – запирания транзистора (рис.7.3). В МДП-транзисторе с индуцированным каналом

(рис.7.2), наоборот – при отсутствии тока на затворе между стоком и истоком носителей недостаточно для протекания тока, подача напряжения на затвор позволяет увеличить число носителей – открыть транзистор.

Для использования МДП-транзисторов в качестве ячеек памяти они модифицированы (рис.7.4) – в схему рис. 7.2 введен дополнительный изолированный, т.е. окруженный диэлектриком, затвор. Такой затвор называется плавающим. При записи информации затвор вводит или, наоборот, удаляет заряд на плавающем затворе – для этого используется тоннельный эффект.

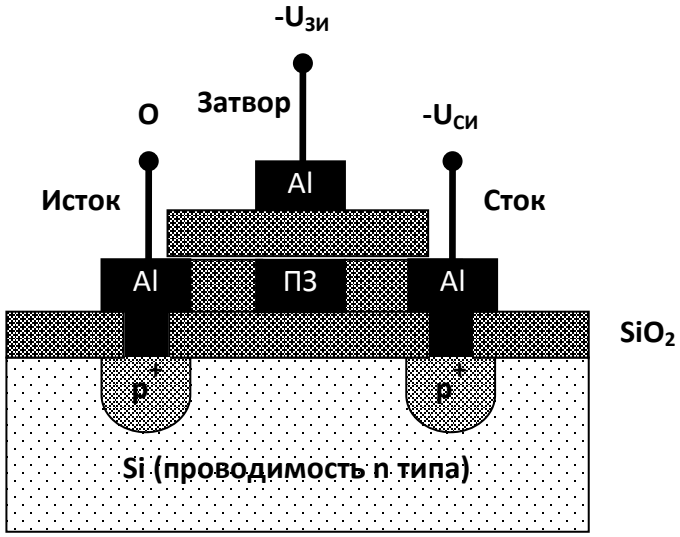


Рис.7.4. МДП-транзистор с плавающим затвором

Плавающий затвор (ПЗ на рис.7.4) хранит заряд, обеспечивая требуемый режим: проводимость соответствует хранению единицы, а непроводимость – нуля.

Литература

1. *Таненбаум Э., Остуртин Т.* Архитектура компьютера // 6-е издание. — СПб.: Питер, 2013. — 811 с., илл. — ISBN: 978-5-496-00337-7.
2. *Цилькер Б.Я., Орлов С.А.* Организация ЭВМ и систем // Учебник для вузов. — СПб.: Питер, 2004. — 668 с.: ил. ISBN 5-94723-759-8
3. *С. А. Майоров, Г. И. Новиков* Принципы организации цифровых машин // Машиностроение, 1974. 434 С.
4. *Карцев М.А.* Архитектура цифровых вычислительных машин // М.: Наука, 1978. – 295 С.
5. http://vssit.ucoz.ru/index/0-4 электронный ресурс
6. У. *Столлингс.* Структурная организация и архитектура компьютерных систем, 5-е изд. М.: Вильямс, 2002, 896 с.
7. *Чекменев С.Е.* Архитектура вычислительных систем. Конспект лекций.
8. *Китаев Ю.В.* Конспект по курсу "Электроника и VG" – цифровые и микропроцессорные устройства // http://de.ifmo.ru/--books/electron/ электронный ресурс.

8. Вычислительные системы

8.1. Понятие системы

Термин «система» в настоящее время используется крайне широко, но его популярность имеет и обратную сторону – достаточно часто предполагаемая интуитивная понятность термина имеет следствием разное его понимание собеседниками и, как результат, взаимное непонимание. Приведем несколько часто встречающихся определений системы:

- Нечто, большее, чем просто сумма её элементов.
- Совокупность взаимосвязанных сущностей, воспринимаемая как единое целое.
- Множество взаимосвязанных элементов, обособленное от среды и взаимодействующее с ней, как единое целое.
- Конечное множество функциональных элементов и отношений между ними, выделенное из среды в соответствии с определенной целью в рамках определенного временного интервала.
- Отражение в сознании субъекта свойств объектов и их отношений в решении задачи исследования, познания.

Нетрудно видеть, что эти определения отличаются расплывчатостью и неконструктивностью в том плане, что их сложно использовать для построения системы. Попробуем определить систему через её атрибуты, т.е. неотъемлемые свойства.

Начнем с **функциональных атрибутов:**

- Синергетичность (от συνεργία - σύν — вместе, ἔργον — дело, труд, работа);
- Эмерджентность (от *emergent* — возникающий, неожиданно появляющийся);
- Целенаправленность или телеологичность (наличие целевой функции);
- Альтернативность (выбор путей достижения цели и самоорганизации для достижения цели как ограничения числа степеней свободы).

В качестве структурных атрибутов отметим два взаимосвязанных:

- Структурированность;
- Иерархичность.

Структурированность означает наличие связей между элементами системы – её внутреннюю связность или взаимозависимость элементов системы. Вопрос классификации связей элементов системы выходит за рамки нашего курса, поэтому рассматривать его сейчас не будем. Упомянем лишь, что важна как длина связей, так и их сила, при этом они должны находиться в определенном балансе. Этот баланс отражается ранее упомянутой нами автокорреляционной функцией (1.1).

Необходимо обратить внимание на то, что иерархичность является не просто свойством, но именно атрибутом, т.е. неотъемлемым свойством системы. Это противоречит доминирующему в современном массовом сознании мнению о ненужности иерархии и всеобщем равенстве, но

математика утверждает, что устойчивы только системы, имеющие иерархическую организацию, т.е. не только горизонтальные связи, но и вертикальные.

Поскольку нас интересуют компьютерные системы, т.е. системы, предназначенные для работы с информацией, то необходимо упомянуть закон Эшби (*William Ross Ashby* 06.09.1903 – 15.11.1972), который гласит:

«*Разнообразие (энтропию) системы можно понизить не более чем на величину количества информации в управляющей системе об управляемой, которое равно разнообразию (энтропии) управления за вычетом потери информации от неоднозначного управления.*» [1]

Пусть X – управляемая система, т.е. множество её состояний x, $x \in X$, а U – множество управлений, где $u \in U$ – управления.

Определение 1: Управлением называется отображение

$$u : x \to y \in Y \subseteq X \quad , \tag{8.1}$$

где x и y – состояния системы до и после управления. Введем энтропию H. Если имеет место

$$H(y) \geq H(x) ,$$

то система неуправляема. Таким образом, цель управления определяется как снижение энтропии управляемой системы

$$H(y) < H(x).$$

Состояние системы после управления

$$H(y) \geq H(x) - I(u,x), \tag{8.2}$$

где $I(u,x) = H(u) - H(u|x)$ – количество информации (использована энтропийная мера информации) в управляющей системе об управляемой. Таким образом, для того, чтобы управлять, надо знать больше, чем объект управления! Компетентность – необходимое условие, сам по себе административный статус не обеспечивает эффективность управления. Более того, управление со стороны некомпетентного лица неэффективно.

8.2. Параллелизм вычислений и пути его достижения. Закон Амдала

Понятие вычислительной системы неразрывно связано с проблемой решения сложных вычислительных задач, требующих повышения вычислительной мощности. Один из основных методов – переход к **параллельным вычислениям**, под которыми понимаются процессы обработки данных, характеризующиеся одновременным выполнением нескольких операций. Как мы уже изучили ранее, возможен параллелизм на уровне программ и на уровне устройств.

В целом, можно выделить следующие режимы выполнения независимых частей программы:

1. многозадачный режим или режим разделения времени – здесь используется один процессор, т.е. этот режим – псевдопараллельный, т.к. в

любой момент времени исполняется только одна часть программы, а остальные ждут своей очереди; в этом режиме проявляются многие эффекты «настоящих» параллельных вычислений;

2. параллельное выполнение как на нескольких процессорах, так и посредством конвейерных и векторных процессоров;

3. распределенные вычисления на нескольких процессорах или компьютерах, объединенных в сеть – этот режим сопряжен с потерями времени на передачу данных, поэтому режим эффективен для параллельных алгоритмов с низкими требованиями к межпроцессорному (межкомпьютерному) обмену данными.

Интуитивно очевидно, что параллелизм на уровне устройств (на уровне программ мы его уже рассмотрели ранее) может быть обеспечен следующими методами:

1. независимостью функционирования отельных узлов и элементов вычислителя (или системы);

2. избыточностью элементов вычислителя.

Последнее может быть достигнуто как их дублированием, так и использованием специализированных для решения отдельных задач устройств (сопроцессоров).

Закон Амдала (Gene Amdahl, 1967) дает оценку ускорения вычислений при использовании n процессоров. Пусть T_S – время решения задачи на однопроцессорном устройстве, тогда выигрыш во времени решения задачи на устройстве из n процессоров может быть оценен следующим образом:

$$S = \frac{T_S}{T_{Pn}} = \frac{n}{1+(n-1)\cdot f}\ , \qquad (8.3)$$

где f – доля операций, которые должны выполняться последовательно одним из процессоров. Можно видеть, зависимость (8.3) с ростом параметра f имеет тенденцию к насыщению:

$$\lim_{n\to\infty} S = \frac{1}{f}\ .$$

Таким образом, закон Амдала формулирует фундаментальное ограничение на рост производительности при распараллеливании вычислений. Вместе с тем, оценка (8.3), как было несколько позже (1988) показано Густафсоном и Барсисом, излишне пессимистична.

Закон Густафсона – Барсиса. При выводе закона Амдала предполагалось, что рост n не сопровождается ростом объема задачи. Но в реальности более мощные системы приобретаются для решения более сложных задач или, если на новой вычислительной системе решается старая задача, то пользователь стремится улучшить качество результата, например, повышением его точности. Это ведет к росту объема вычислений, при этом рост объема вычислений часто сопровождается снижением доли последовательных операций за счет роста параллельных. Например, в некоторых задачах вычисляемая формула позволяет разбить

большой массив данных N на подмассивы N/n, для них параллельно выполнить некоторые операции, а уже затем их результаты подвергнуть той же операции. Таким образом может быть получена экономия.

Этот факт позволил несколько улучшить оценку эффективности распараллеливанию, даваемую законом Амдала (8.3), и сформулировать **закон масштабируемого ускорения**.

Пусть t_S и t_P – времена выполнения последовательной и параллельной частей вычислений, тогда доля последовательно выполняемых расчетов

$$g(n) = \frac{t_S}{t_S + t_P / n} .$$

Отсюда можно получить оценку достижимого при распараллеливании ускорения вычислений, известную как закон Густафсона – Барсиса:

$$S = \frac{T_S}{T_{Pn}} = \frac{t_S + t_P}{t_S + t_P / n} = \frac{\left(t_S + t_P / n\right)\left(g(n) + n\left(1 - g(n)\right)\right)}{t_S + t_P / n} = n + (1 - n)g(n). \quad (8.4)$$

Поскольку закон Густафсона – Барсиса дает оценку эффективности организации параллельных вычислений при увеличении сложности решаемых задач (а не просто их объема!), то он часто называется законом масштабируемого ускорения. Выигрыш, даваемый законом Густафсона – Барсиса (8.4) относительно закона Амдала (8.3), оценивает творческую роль человека в решении задачи – выигрыш достигается не «тупой силой» компьютера по принципу «компьютер есть – ума не надо», а организацией процесса решения задачи, т.е. именно умом человека, формулирующего способ вычислительного решения задачи – алгоритм и программу.

8.3. Классификация вычислительных систем. Систематика Флинна

Для классификации вычислителей в контексте параллелизма вычислений часто используется классическая систематика (или классификация) Флинна (M.J.Flynn, 1966, 1972), основанная на анализе способов взаимодействия потоков. Поэтому начнем с рассмотрения **концепции потоков.**

В современных компьютерах используется многозадачный режим выполнения программы, т.е. работа процессора организована так, что он попеременно выполняет несколько программ. Для этого необходимо определить единицы работы процессора и других узлов вычислителя, между которыми и будет делиться его ресурс (время). На сегодня основной единицей работы выступает процесс (иногда встречается термин задача). Несколько процессов могут быть объединены в задание. Процессы могут включать в себя несколько потоков (нитей).

Процесс – это выполняемая программа (в стадии выполнения). Процесс может находиться в следующих состояниях:

Порождение – подготавливаются условия для первого выполнения на процессоре;

Активное состояние – программа выполняется;
Ожидание – программа не выполняется на процессоре по причине занятости какого-либо ресурса;
Готовность – программа не выполняется, но для ее исполнения представлены все ресурсы, кроме времени процессора;
Окончание – нормальное или аварийное завершение программы.

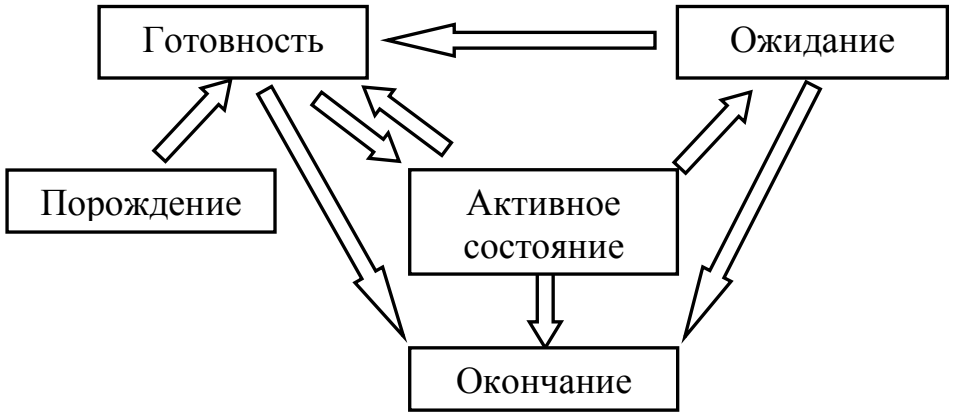


Рис.8.1. Состояния процесса и их связи

Процессы обмениваются между собой данными по однонаправленным линиям коммутации – **каналам**. Каналы делятся на:

Синхронные – сообщение может быть передано только после получения подтверждения о приеме предыдущего – принимающий процесс не выполняется, пока не получит данные, а передающий - пока не получит подтверждение о приеме данных.

Асинхронно/синхронные – сообщение копируется в буфер и пересылается оттуда одновременно с работой передающего процесса. Приме сообщений синхронный - процесс блокируется до момента приема.

Асинхронные – и передача, и прием завешаются сразу.

Возможная проблема «Deadlock», суть в том, что один процесс не может передать сообщение другому, т.к. тот ждет от него подтверждения. В этом случае применяется **Семафор** – защищенная переменная, значение которой можно опрашивать и менять только при помощи специальных операций ***wait*** и ***signal*** и операции инициализации ***init***. Двоичные семафоры могут принимать только значения 0 и 1. Семафоры со счетчиками могут принимать неотрицательные целые значения.

Процессы общаются между собой через ОС, потребляют ресурсы, так как:

- процессы решают общую задачу;
- процессы работают с одними и теми же данными;
- процессы используют одно и то же право доступа к ресурсам.

ОС рассматривает такие связанные процессы как обыкновенные и изолирует их друг от друга. В результате, так как на создание каждого такого процесса ОС тратит системные ресурсы, возникает проблема

неэффективного их использования: у каждого процесса своя физическая память, свои устройства ввода-вывода, своё виртуальное пространство.

Решение этой проблемы возможно в рамках многопоточной обработки, в основу которой положен учет тесных связей между отдельными ветвями вычисления одного и того же задания.

NB! Многопоточная обработка – это не многопрограммная обработка!

Выделяют следующие потоки (или последовательности):

- потоки управления;
- потоки данных;
- потоки запросов.

Поток управления: очередная команда получает возможность исполнения сразу после того как ей передано управление. Таким образом, поток управления – это последовательность выполнения команд в ходе выполнения программы. Применяется поток управления в случаях, когда возможно одновременное (параллельное) выполнение ряда операций.

Особенности потока управления:

- Данные рассылаются между командами не непосредственно, а через ячейки общей памяти. Обращение производится к общей памяти.
- Операнды могут находиться и в самих командах для улучшения доступа к ним.
- Поток управления по своей сути является последовательным, параллельное выполнение команд организуется вводимыми в него специальными управляющими операндами, обеспечивающими переходы.

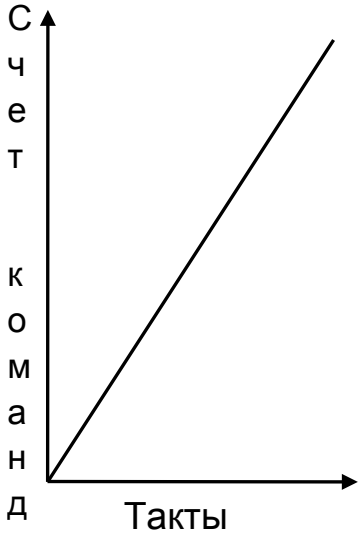


Рис.8.2. Поток управления без переходов

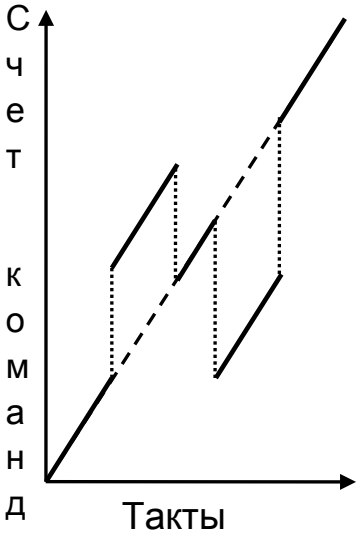


Рис.8.3. Поток управления с переходами

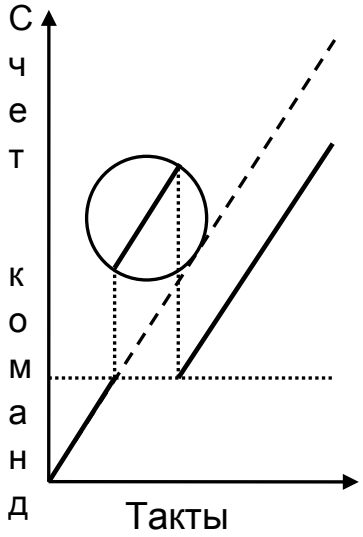


Рис.8.4. Поток управления с процедурой (обведена)

Процедура. Вызов процедуры аналогичен команде перехода в том смысле, что он также передает управление – изменяет поток управления, но, в отличие от команд перехода, после выполнения процедуры управление возвращается команде, вызвавшей процедуру (рис.8.4). Тело

процедуры можно рассматривать как определение новой команды на более высоком уровне.

Обычно процедуры имеют иерархию. Если требуется равноправие процедур, например, для параллельной обработки данных на одном процессоре, то применяются **сопрограммы**. Каждая сопрограмма работает как бы одновременно с другими сопрограммами так, как будто у нее есть собственный процессор.

Здесь важно различать сопрограммы и подпрограммы.

Таблица 8.1 Сравнение подпрограмм и сопрограмм

	Подпрограмма	Сопрограмма
Точки входа	Подпрограмма всегда имеет одну точку входа.	Сопрограмма имеет одну точку входа и внутри последовательность возвратов и следующих за ними точек входа.
Возврат управления	Подпрограмма возвращает управление только один раз. После повторного вызова подпрограммы управление начинается со стартовой точки входа.	Сопрограмма может возвращать управление много раз. После повторного вызова сопрограммы управление начинается с места, в котором было передано управление.

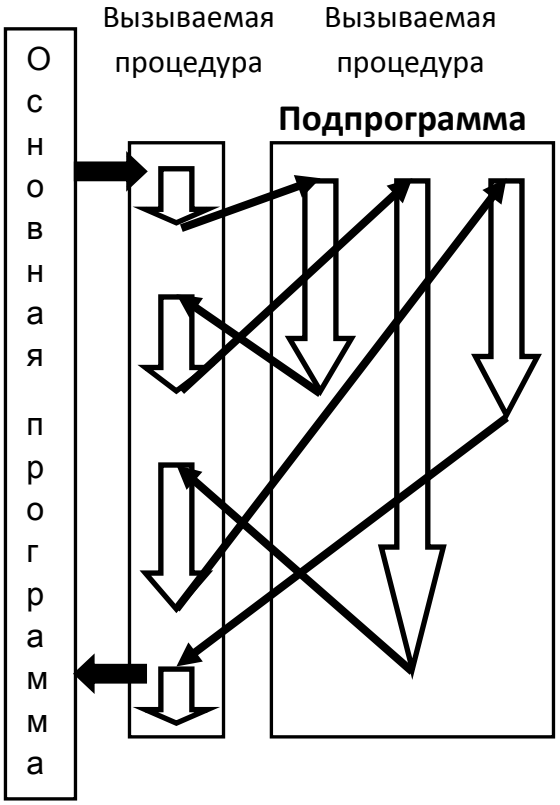


Рис.8.5. Организация управления с вызовом подпрограммы

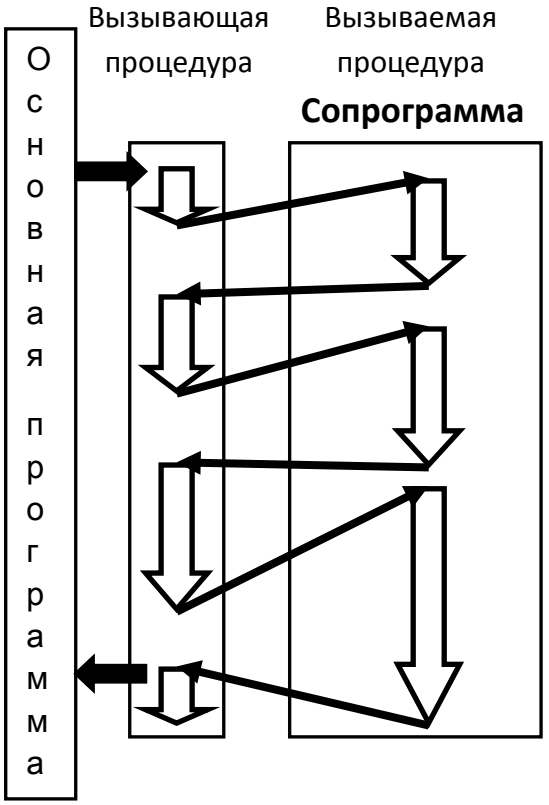


Рис.8.6. Организация управления с вызовом сопрограммы

Выделяют три типа элементов потока управления:

- **контейнеры** – обеспечивают структуры в пакетах;
- **задачи** – обеспечивают функциональность;
- **элементы управления очередностью** – соединяют выполняемые компоненты, контейнеры и задачи в упорядоченный поток управления.

Контейнеры – это объекты, содержащие структуру пакетов и службы для задач. Поддерживают повторение потоков управления в пакетах, а также группируют задачи и контейнеры в единые рабочие объекты. Кроме задач, контейнеры могут включать в себя другие контейнеры.

Пакеты используют контейнеры для решения следующих задач:

- Повторение задач для каждого элемента: файлов, папок, схем или управляющих объектов. Например, пакет может выполнять инструкции, размещенные в нескольких файлах.
- Повторение задач до тех пор, пока значение определенного выражения не будет равно ***false***. Например, пакет может ежедневно посылать сообщения по электронной почте в течение заданного срока.
- Группировка задач и контейнеров, выполнение которых, успешное или аварийное, учитывается как для единого объекта. Например, пакет может группировать задачи, удаляющие и добавляющие строки в таблице базы данных, а затем фиксировать их или же производить откат всех связанных задач в случае сбоя одной из них.

Задачи – это элементы потока управления, которые определяют рабочие модули, выполняющиеся в потоке управления пакета. Если в пакете несколько задач, они связаны и упорядочены в потоке управления с помощью **управления очередностью**.

Элементы управления очередностью связывают исполняемые объекты, контейнеры и задачи в пакетах в **поток управления** и задают условия, которые определяют, выполняются ли исполняемые объекты.

В качестве исполняемого объекта могут быть контейнеры «цикл по элементам» и «цикл по каждому элементу», контейнеры последовательности, задача или **обработчик события.** Обработчики событий также используют управление очередностью для связывания своих исполняемых объектов в поток управления.

Поток данных: очередная команда получает возможность исполнения сразу после того как появляются операнды.

Особенности потока данных:

- наличествует специальный механизм пересылки данных для рассылки промежуточных данных непосредственно между командами;
- операнды могут выть встроены в команды;
- не используется традиционная концепция общей памяти данных, в результате нет конфликтов и временных потерь;
- ограничения по последовательности выполнения вычислительных действий определяются только зависимостями между данными, что позволяет полностью реализовать присущий модели параллелизм.

Отметим, что параллелизм здесь – естественный. Управление вычислениями децентрализовано.

Поток запросов: очередная команда получает возможность исполнения сразу после того как появляются необходимость в её результате.

Особенности потока запросов:

- все программные структуры, команды и аргументы являются выражениями;
- не используется обычная концепция памяти для данных;
- нет никаких ограничений по последовательность выполнения вычислительных действий сверх ограничений, присущих собственно самим запросам на операнды;
- нет общей памяти;
- нет системы централизованного управления.

Теперь можно перейти к собственно классификации архитектуры вычислительных систем, основанной на анализе способов взаимодействия потоков выполняемых команд и данных. Согласно систематике Флинна выделяют следующие четыре класса: **SISD, SIMD, MISD, MIMD.** В последнее время ряд специалистов выделяет еще один класс – **MSIMD.**

1. **SISD** (Single Instruction, Single Data) – вычислительные системы с одиночным потоком команд и одиночным потоком данных (Рис.8.7). Это обычные последовательные фон-Неймановские компьютеры с одним ЦП, способным обрабатывать только один поток последовательно исполняемых команд. Сюда же относятся и вычислители суперскалярной архитектуры, позволяющей за счет специальных аппаратных решений параллельное выполнение нескольких скалярных операций (операций над парами чисел).

2. **SIMD** (Single Instruction, Multiple Data) – вычислительные системы с одиночным потоком команд и множественным потоком данных (Рис.8.8).

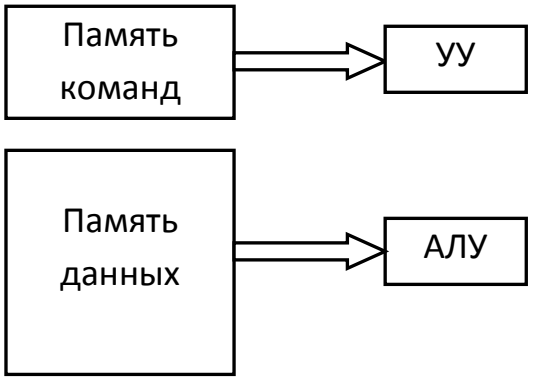


Рис.8.7. ВС класса SISD

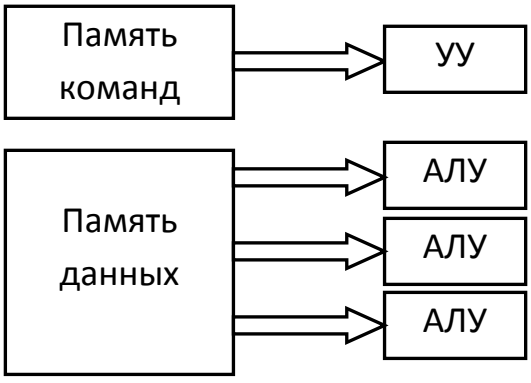


Рис.8.8. ВС класса SIMD

В этот класс попадают следующие типы вычислительных систем:

- *однопроцессорные, векторно-конвейерные* суперкомпьютеры – имеется один поток векторных команд и много потоков данных, организованных так, что каждый элемент вектора входит в отдельный поток данных;

- *многопроцессорные вычислительные системы*, в которых в каждый момент времени может выполняться одна и та же команда для обработки нескольких информационных элементов; такой архитектурой обладают, например, многопроцессорные системы с собственной памятью каждого процессора, единым устройством управления и «жесткой» конфигурацией. Этот подход одно время был популярен, но сейчас применяется в основном для специализированных систем, поскольку большое число процессоров, выполняющих одну операцию для всех данных, дает колоссальный выигрыш в производительности, но число таких задач не очень велико. Поэтому на сегодня в основном это:
- *векторные компьютеры*, использующие специально разработанные векторные центральные процессоры – при работе в векторном режиме они обрабатывают данные практически параллельно, благодаря этому скорость обработки возрастает в разы сравнительно со скалярным режимом;
- *матричные процессоры*, также имеющие векторные команды и проводящие векторную обработку, но не конвейерным методом, как векторные суперкомпьютеры, а матрицами процессоров.

Касательно двух последних типов отметим, что в силу похожести моделей вычислений на векторных и матричных компьютерах они часто рассматриваются как эквивалентные. Для них характерно наличие таких циклов на элементах массива, в которых данные, полученные на одной итерации, на другой не используются.

Вместе с тем, конвейерная организация работы векторного процессора ограничивает его функциональность, в то время, как матричные процессоры дают существенно большую гибкость, поскольку элементы последних – фактически универсальные программируемые вычислители и, как следствие, решаемая на них параллельная задача может быть достаточно сложной, в том числе, содержать ветвления. Поэтому сегодня термин «матричный процессор» часто применяется как синоним **SIMD**.

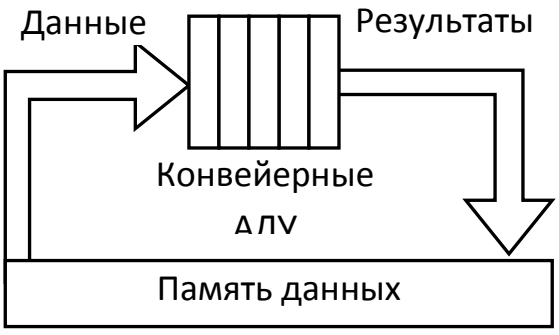


Рис.8.9. Векторно-конвейерная ВС класса SIMD

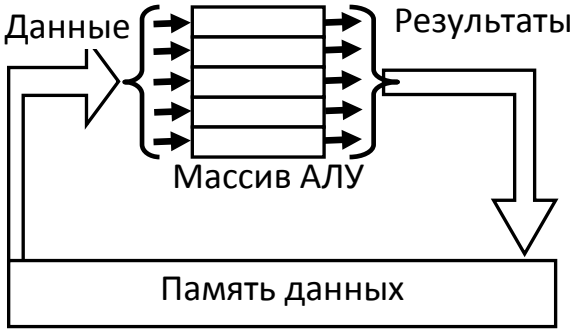


Рис.8.10. ВС класса SIMD с матрицей процессоров

3. **MISD** (Multiple Instruction, Single Data) – вычислительные системы с множественным потоком команд и одиночным потоком данных. Относительно этого типа вычислительных систем одно время была

популярна точка зрения, что реальных систем, соответствующих данному классу, не существовало вообще, а сам этот класс был введен исключительно исходя лишь из соображений полноты и завершенности классификации. Но в настоящее время существуют адекватные **MISD** системы – **распределенные мультипроцессорные системы с общими данными**. Практический пример такой системы – обычная локальная сеть персональных компьютеров, работающая с единой базой данных, в которой несколько компьютеров обрабатывают один поток данных. Но такая «локалка» относится к **MISD** только до того момента, пока ряд пользователей работают над одной задачей и одной базой данных. Если пользователи переходят на обработку собственных данных, то тотчас же «локалка» превращается в **MIMD.**

4. **MIMD** (Multiple Instruction, Multiple Data) – этот класс охватывает большинство многопроцессорных вычислительных систем, включая невекторные суперЭВМ. Класс включает все уровни параллелизма, начиная с конвейера операций. Подразумевается не только многопроцессорность в смысле аппаратного обеспечения, но и *многопроцессность* – одновременность выполнения множества вычислительных процессов. В качестве синонима часто используется термин «мультипроцессор».

В системах **MIMD** каждый процессорный элемент независим от других в том смысле, что исполняет свою программу. Модель вычислений на мультипроцессоре (MIMD) – совокупность независимых процессов, эпизодически обращающихся к разделяемым данным.

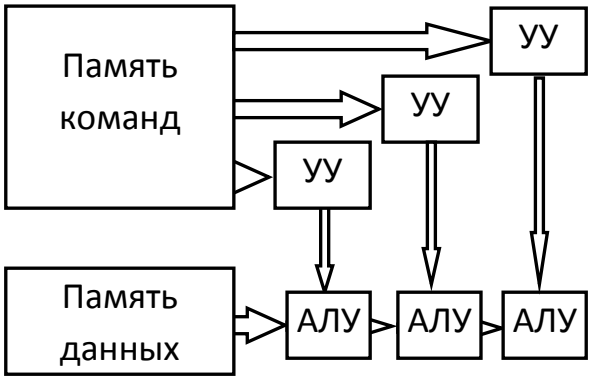


Рис.8.7. ВС класса MISD

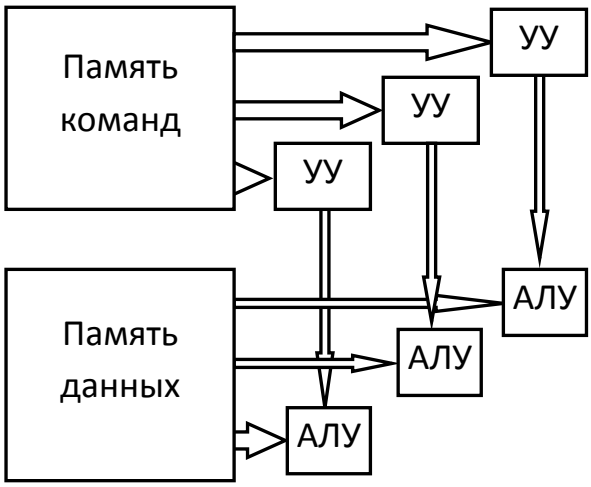


Рис.8.8. ВС класса MIMD

На сегодня MIMD – основное направление развития супер-компьютеров, т.к. обладает следующими достоинствами:

- Обеспечивает существенные гибкость и производительность – эта архитектура может работать и как однопользовательская высокопроизводительная система, и как многопользовательская, и как

их комбинация. Эти аппаратные возможности реализуются разработкой соответствующих языков программирования и компиляторов.

- Позволяет оптимизировать затраты по критерию стоимость/эффективность – в системах MIMD используются массово выпускаемые, т.е. недорогие, процессоры.

Вместе с тем, надо отметить, что на сегодня MIMD превратился в чрезвычайно широкий класс архитектур. Поэтому обсуждается вопрос поиска другого, более адекватного современному уровню развития супер-компьютеров подхода к классификации, чем классическая таксономия Флинна.

5. MSIMD – сильносвязанные комплексы класса MIMD. Этот класс систем введен недавно, его появление обусловлено тем, что в настоящее время многие системы строятся как мультипроцессорные (MIMD), в которых в качестве процессоров используются процессоры типа SIMD или векторные. Такое решение позволяет объединить сильные стороны каждой из архитектур: векторный параллелизм используется в тех частях программы, которым он в наибольшей степени адекватен, а гибкость MIMD-архитектуры, в свою очередь, позволяет получить выигрыш при реализации остальных частей программы.

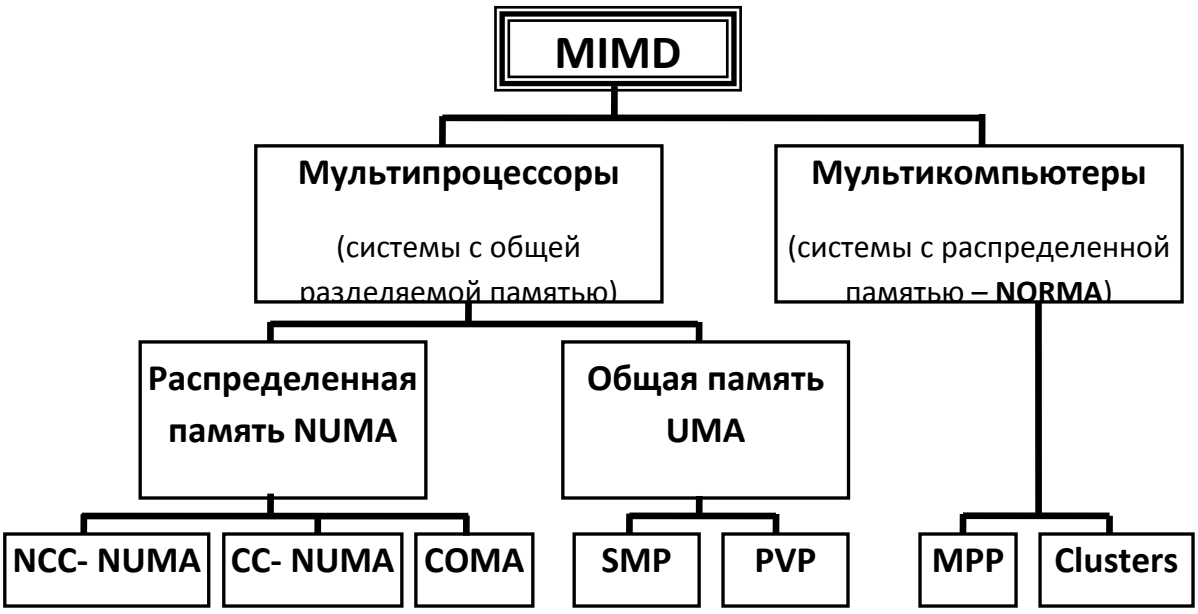


Рис.8.9. Классификация ВС MIMD

Внутри класса MIMD можно выделить два больших подкласса: системы с общей, т.е. разделяемой, основной (оперативной) памятью, называемые также мультипроцессорами, и системы с распределенной памятью или мультикомпьютеры.

Мультипроцессоры собираются обычно из относительно небольшого числа процессоров – до нескольких десятков. Это ограничение связано с

тем, что по мере роста числа и производительности процессоров возникает и резко обостряется проблема организации их доступа к памяти – требования полосе пропускания канала связи с памятью возрастают отнюдь не линейно.

Мультипроцессоры, в свою очередь, делятся на два вида по критерию организации доступа к памяти:

- **NUMA** – uniform memory access, т.е. системы с неоднородным доступом к памяти, включающие в себя:
  - **NCC-NUMA** – non-cache-coherent memory access;
  - **CC-NUMA** – cache-coherent non-uniform memory access;
  - **COMA** – cache-only memory architecture
- **UMA** –uniform memory access, т.е. системы с однородным доступом к памяти, включающие в себя:
  - **SMP** – symmetric multiprocessor;
  - **PVP** – parallel vector processor.

В системах с однородным доступом к памяти (UMA) (Рис.8.10) группа процессоров под управлением одной операционной системы работает с общей основной памятью, что увеличивает скорость и производительность, при этом у каждого процессора есть своя кэш-память.

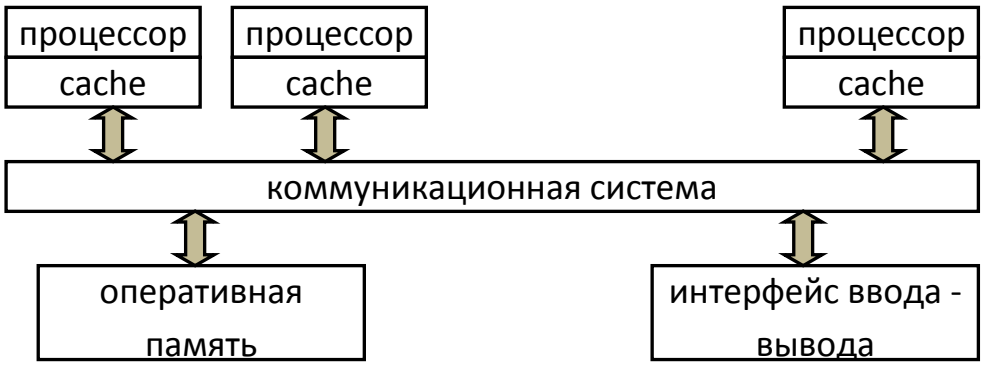


Рис.8.10. Схема архитектуры UMA

Для систем с однородным доступом к памяти (UMA) характерна проблема, известная как проблема когерентности КЭШей (Cache coherence problem) или проблема однозначности содержимого разных кэшей. Суть в том, что если используется разделяемая переменная, то копии её значения в один момент времени могут оказаться в разных кэшах. Каждый процессор может изменить её значение независимо от других, что приведет к неоднозначности вычислений.

Решение проблемы на аппаратном уровне – при изменении значения общей переменной все копии этой переменной в кэшах отмечаются как недействительные и последующий доступ к этой переменной потребует обязательного обращения к основной памяти. Следствие – снижение быстродействия и трудности создания систем UMA с большим числом процессоров.

Существует также проблема синхронизации одновременно выполняемых потоков команд – при наличии общих данных изменения в каждый момент может выполнять только один поток.

От этих проблем свободны системы с неоднородным доступом к памяти (NUMA – Рис.8.11). Неоднородный доступ обеспечивается физическим разделением памяти.

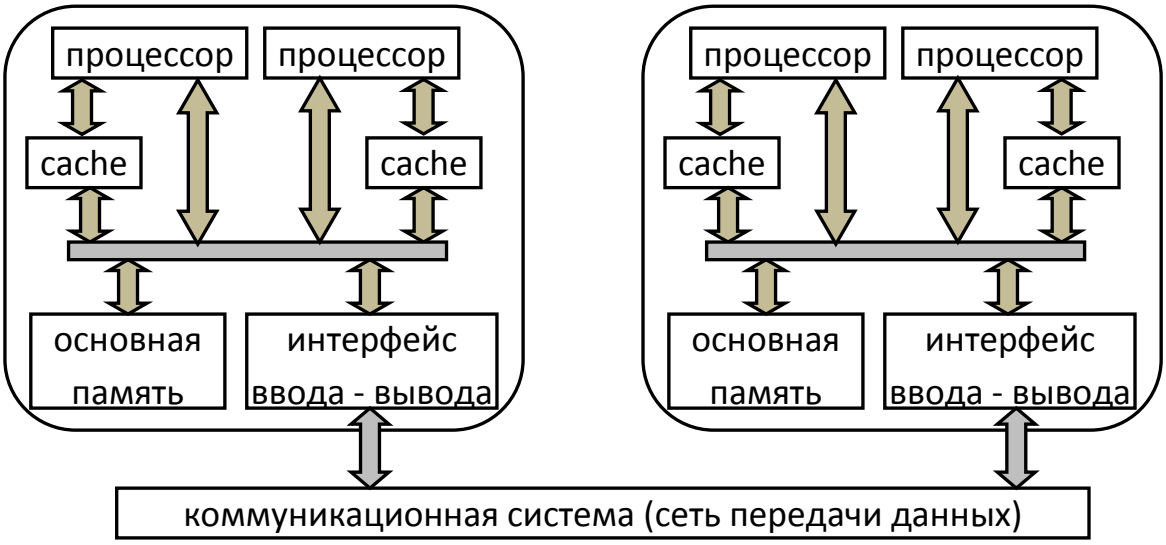


Рис.8.11. Схема архитектуры NUMA

В системах с распределенной памятью **NORMA** (мультикомпьютерах – рис.8.12) общий доступ ко всей памяти системы не обеспечивается – принципиальное отличие от архитектур NUMA в том, что каждый процессор использует только свою локальную память, а доступ к данным на других процессорах организован посредством специальных операций передачи сообщений. Здесь выделяют два класса:

- **MPP** – massive parallel processors;
- **Clusters**.

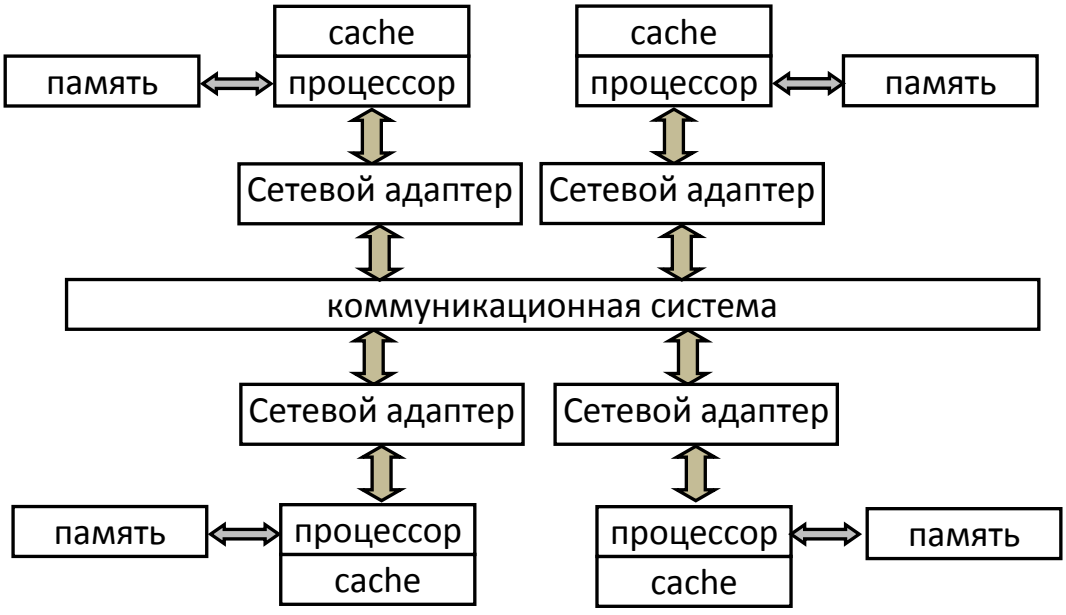


Рис.8.12. Схема архитектуры NORMA

Достоинства архитектуры NORMA:

1. отсутствует конкуренция за доступ к каналам связи с памятью;
2. размер системы ограничивается только сетью (её характеристиками), связывающей отдельные процессоры;
3. нет проблемы когерентности кэш-памяти.

Недостатки архитектуры NORMA :

Сложность организации эффективного межпроцессорного обмена информацией, а именно:

- затраты времени на формирования и пересылку межпроцессорного сообщения;
- необходим запрос на прерывание и его обработка для реакции на запрос от другого процессора.

Кластерные вычислительные системы или просто **кластеры** иногда в шутку называют «суперкомпьютерами для бедных» или «собери суперкомпьютер своими руками».

Кластер – разновидность параллельной или распределенной системы. Это группа компьютеров, объединенных высокоскоростными каналами связи и с точки зрения пользователя представляющая собой единый вычислительный ресурс. Кластер строится на базе массово выпускаемых компонентов, состоит из стандартных серверов, объединенных высокопроизводительной системной сетью, называемой интерконнектом.

Для объединения компьютеров в сеть часто используется коммутатор (switch). Такой способ коммутации компьютеров дает топологию сети типа полного графа, обеспечивающего передачу данных между любыми двумя компьютерами, входящими в сеть. Но в этом случае каждый компьютер может участвовать только в одной операции приема-передачи данных, т.е. возможность выполнения нескольких коммутаций одновременно ограничена. Таким образом, параллельно могут выполняться только те коммуникационные операции, в которых взаимодействующие пары компьютеров не пересекаются.

Классификация кластеров:

1. Кластеры высокой доступности (HA – High-availability clasters);
2. Кластеры распределения нагрузки (Load balancing clasters);
3. Кластеры повышенной производительности (HPA - High-performance clasters);
4. Системы распределенных вычислений или Grid-системы.

Кластеры высокой доступности. Высокая доступность обеспечивается избыточностью узлов для обеспечения обслуживания при отказе одного или нескольких узлов. Минимальное число узлов – 2.

Кластеры распределения нагрузки. Для повышения производительности один или несколько узлов используются как специализированные входные, распределяющие нагрузку между остальными – вычислительными. Применяются также и специальные методы повышения надежности – **серверные фермы**. Серверной фермой

называется группа серверов, объединенных в единое целое сетью передачи данных и реализующая среду распределенной обработки данных.

Кластеры повышенной производительности. В них скорость вычислений увеличивается за счет распараллеливания потоков. Для этого используется специальное программное обеспечение. Пример такой системы – Beowulf – это набор серверов, работающих под управлением ОС Linux.

Системы распределенных вычислений. Строго говоря, Grid-системы не считаются кластерами. Узлы Grid-системы непредсказуемо подключаются и отключаются во время работы, поскольку принадлежат независимым пользователям. Поэтому задача должна быть разбита на ряд независимых процессов. Нестабильность конфигурации Grid-системы компенсируется избыточностью узлов исходя из того, что в любой момент времени окажется достаточное для выполнения задачи число компьютеров, не используемых их владельцами по своему усмотрению.

Достоинства кластерной архитектуры:

1. эффективное соотношение стоимость/производительность;
2. возможности расширения добавлением стандартных узлов;
3. высокая отказоустойчивость за счет избыточности, возможность замены узлов «на лету», без остановки всей системы;
4. простота обслуживания;
5. низкая стоимость владения.

Слабые стороны кластерной архитектуры:

1. проблемы стандартизации;
2. закрытость разработок и, как следствие, проблемы совместимости;
3. сложности одновременного доступа к файлам;
4. сложности с управлением конфигурацией, настройкой, развертыванием системы и т.п.

Узкое место кластерных систем – каналы связи между компьютерами.

1. Для решения этой проблемы были разработаны System Area Networks (SAN) – специальные межмашинные сети связи.

2. Помогает решению проблемы развитие стандартных средств коммуникации – компьютеры взаимодействовали друг с другом быстрее, чем с системами хранения данных.

3. Две сети: SAN одна для данных (медленная – сотни мегабит) и другая для сообщений (быстрая – мультигигабиты). Сети SAN, объединяющие центральные средства хранения данных с группой вычислительных узлов используются только для обмена между средствами хранения и узлами; для передачи сообщений отдельная сеть.

4. **InfiniBand**, используемая в кластере кафедры ФиОИ – это системная сеть, а не усовершенствованная SAN.

Системная сеть **InfiniBand.** Комбинация SAN, коммуникационной сети и шины ввода-вывода. Пропускная способность SAN ограничена полосой шины PCI. InfiniBand выводит шину ввода-вывода за пределы

архитектуры одиночного компьютера. Машины используют внутренние каналы связи для обмена данными с собственными ОЗУ, а внешние средства InfiniBand обращаются к контроллеру ОЗУ напрямую, минуя шину PCI.

Топология сетей обмена данными. Поскольку общение узлов мультикомпьютерной системы, т.е. обмен данными, осуществляется посредством специальных коммуникационный сетей, то любые задержки в них непосредственно сказываются на общей оценке эффективности работы системы. Поэтому существует понятие **коммуникационной трудоемкости алгоритма**, оказывающей существенное влияние на выбор методов распараллеливания задачи.

Упомянем наиболее распространенные типы топологии сети, т.е. структур коммутационных линий:

- **линейка** (*linear array* или *farm*) – простейшая система, в которой все узлы нумерованы по порядку и каждый узел, кроме первого и последнего, связан только с двумя соседними: предыдущим и последующим (Рис.8.13). Эта схема , вне зависимости от числа узлов, легко реализуема и, к счастью, соответствует структуре передачи данных при решении многих вычислительных задач, например, типа конвейерных вычислений;
- **кольцо** (*ring*) – получается замыканием линейки при соединении первого и последнего узлов (Рис.8.14);
- **звезда** (*star*) – здесь все узлы связаны с центральным – управляющим процессором (Рис.8.15). Такая структура эффективна, например, при организации централизованных схем параллельных вычислений;
- **полный граф** (*completely-connected graph* или *clique*) – это такая система, в которой между любой парой узлов (процессоров, компьютеров) имеется прямая линия связи (Рис.8.16). Такая структура минимизирует затраты при передаче данных, но число связей растет как квадрат от числа узлов;
- **решетка** (*mesh*) – здесь связи образуют прямоугольную сетку (Рис.8.17). Размерность пространства решетки обычно 2 или 3. Структура достаточно проста в реализации и, эффективна при параллельном выполнении многих численных алгоритмов, например, при анализе математических моделей, описываемых дифференциальными уравнениями в частных производных;
- **гиперкуб** (*hypercube*) – это частный случай решетки, в которой по каждой размерности N сетки имеется только два узла, т.е. всего 2^N узлов. Такой структура сети передачи применяется данных достаточно часто, она характеризуется следующими свойствами:
  - условие связи узлов – двоичные представления их номеров должны иметь только одну различающуюся позицию;

- каждый узел связан ровно с N соседями, где N – размерность;
- N-мерный гиперкуб можно разбить на два (N–1)-мерных гиперкуба, всего возможно N различных вариантов разбиений;
- длина кратчайшего пути между двумя любыми узлами равна количеству различающихся битовых значений в номерах узлов, т.е. равна расстоянию Хэмминга.

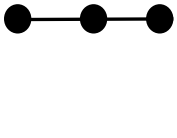


Рис.8.13. Линейка

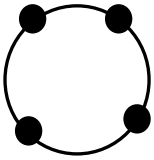


Рис.8.14. Кольцо

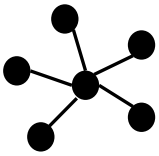


Рис.8.15. Звезда

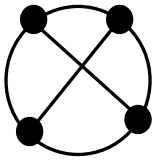


Рис.8.16. Полный граф

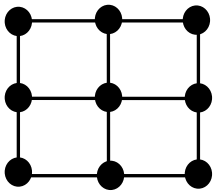


Рис.8.17. Решетка

Литература.

1. *Эшби У.Р.* Введение в кибернетику.М.: «Иностранная литература», 1959.

2. *Таненбаум Э., Остуртин Т.* Архитектура компьютера // 6-е издание. — СПб.: Питер, 2013. — 811 с., илл. — ISBN: 978-5-496-00337-7.

3. *Цилькер Б.Я., Орлов С.А.* Организация ЭВМ и систем // Учебник для вузов. — СПб.: Питер, 2004. — 668 с.: ил. ISBN 5-94723-759-8

4. *С. А. Майоров, Г. И. Новиков* Принципы организации цифровых машин // Машиностроение, 1974. 434 С.

5. *Карцев М.А.* Архитектура цифровых вычислительных машин // М.: Наука, 1978. – 295 С.

6. http://vssit.ucoz.ru/index/0-4 электронный ресурс

7. У. *Столлингс.* Структурная организация и архитектура компьютерных систем, 5-е изд. М.: Вильямс, 2002, 896 с.

8. *Чекменев С.Е.* Архитектура вычислительных систем. Конспект лекций.

9. *Китаев Ю.В.* Конспект по курсу "Электроника и VG" – цифровые и микропроцессорные устройства // http://de.ifmo.ru/--books/electron/ электронный ресурс.

10. *Гергель В.*Теория и практика параллельных вычислений // http://www.intuit.ru/studies/courses/1156/190/info электронный ресурс.

УНИВЕРСИТЕТ ИТМО

Миссия университета – генерация передовых знаний, внедрение инновационных разработок и подготовка элитных кадров, способных действовать в условиях быстро меняющегося мира и обеспечивать опережающее развитие науки, технологий и других областей для содействия решению актуальных задач.

КАФЕДРА ФОТОНИКИ И ОПТОИНФОРМАТИКИ

http://phoinf.ifmo.ru

Заведующий кафедрой – профессор, д.ф.-м.н., лауреат премии Ленинского комсомола, руководитель Российской научной школы «Фемтосекундная оптика и фемтотехнологии» Сергей Аркадьевич Козлов.

На кафедре российские и иностранные преподаватели готовят бакалавров по образовательному направлению 12.03.03 «Фотоника и оптоинформатика» и магистров по образовательному направлению 12.04.03 «Фотоника и оптоинформатика» – одному из самых инновационных направлений современной науки и техники.

Уже со второго курса студенты кафедры имеют возможность практической работы в области фотоники и оптоинформатики, участвуя в исследованиях и разработках Международных научных центров и лабораторий кафедры, а также в ведущих фирмах Санкт-Петербурга. На кафедре разрабатываются и создаются:

- оптические и квантовые технологии сверхбыстрой передачи и записи информации;
- оптические системы искусственного интеллекта и сверхбыстродействующие оптические процессоры;
- информационные оптические системы, строящиеся на основе новых физических принципов, в том числе на нанотехнологиях.

Студенты кафедры участвуют в Международных научных конференциях, публикуют свои работы в ведущих мировых изданиях. Среди студентов и аспирантов кафедры есть стипендиаты Президента и Правительства Российской Федерации, победители конкурсов научных работ, проводимых Российской Академией наук, крупнейшими мировыми научными обществами, такими, как INTAS, SPIE, CRDF, OSA, IEEE Photonics Society.

Павлов Александр Владимирович

Архитектура вычислительных систем

Учебное пособие

В авторской редакции
Компьютерная верстка А.В.Павлов
Редакционно-издательский отдел Университета ИТМО
Зав. РИО Н.Ф. Гусарова
Подписано к печати
Заказ №
Тираж 50 экз.
Отпечатано на ризографе

Редакционно-издательский отдел
Университета ИТМО
197101, Санкт-Петербург, Кронверкский пр., 49