Схемотехнические основы работы компьютера

Введение

Современный компьютер выполняет огромное количество операций: обрабатывает числа, сравнивает значения, хранит данные, выполняет команды программ, управляет устройствами ввода и вывода.

На уровне пользователя всё это выглядит достаточно просто. Мы запускаем программу, вводим данные и получаем результат. Однако возникает вопрос:

Каким образом физические электронные устройства компьютера способны выполнять логические и арифметические операции?

Процессор не «понимает» числа и команды в человеческом смысле. На физическом уровне его работа основана на обработке электрических сигналов, которым сопоставляются два логических состояния:

0;
1.

Из большого количества простых элементов, работающих с такими состояниями, строятся более сложные устройства:

логические элементы → логические схемы → сумматоры и другие узлы → регистры → арифметико-логическое устройство → процессор.

Поэтому для понимания работы компьютера полезно пройти этот путь от самых простых логических операций до выполнения процессором команды программы.

Двоичные состояния и логический сигнал

Компьютер работает с цифровой информацией. В основе цифрового представления лежит двоичная система, в которой используются только две цифры:

0 и 1.

На физическом уровне этим значениям соответствуют определённые состояния электрического сигнала.

Упрощённо можно считать:

- наличие сигнала или высокий уровень → **1**;
- отсутствие сигнала или низкий уровень → **0**.

Важно понимать, что это не означает, что внутри компьютера буквально существует «единица» или «ноль» как отдельные предметы. Речь идёт о состояниях физических схем.

Логическая схема получает один или несколько входных сигналов и формирует выходной сигнал в соответствии с определённым правилом.

Например:

входы: A и B
выход: Y

Если схема выполняет операцию И, то выход будет равен 1 только в случае, когда оба входа равны 1.

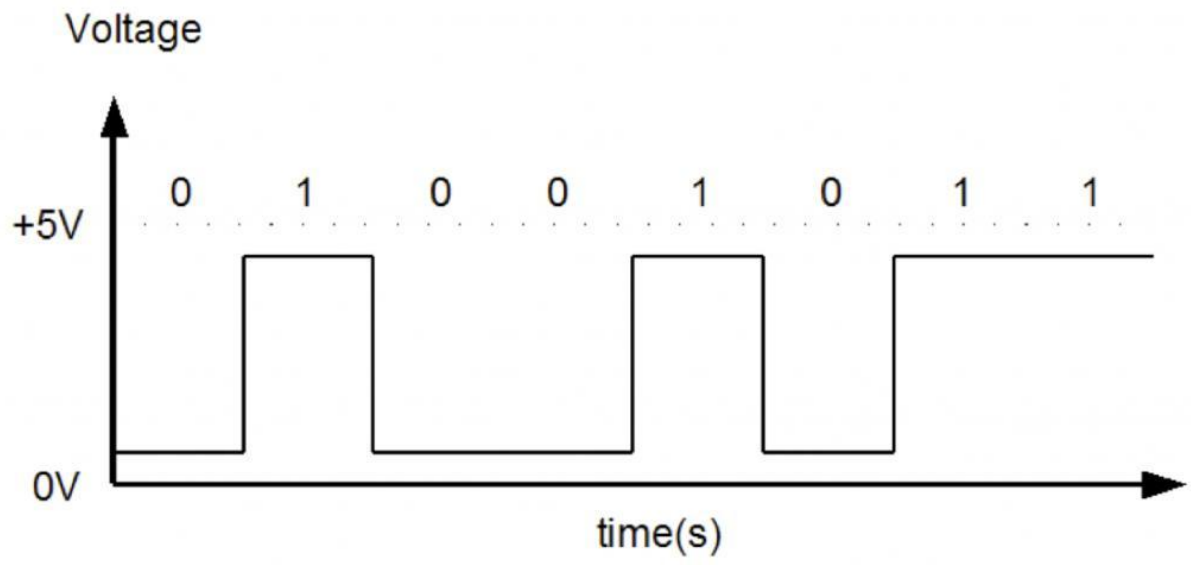


Рисунок 1 - Цифровой сигнал

Логические операции

Основой цифровых схем являются логические операции.

Наиболее важные из них:

НЕ;
И;
ИЛИ;
исключающее ИЛИ (XOR).

Кроме них широко используются производные элементы:

И-НЕ (NAND);
ИЛИ-НЕ (NOR).

Каждая операция имеет определённое правило обработки входных значений.

Для описания работы логической схемы используется **таблица истинности**.

Таблица истинности показывает все возможные комбинации входных значений и соответствующее каждой комбинации значение выхода.

Логическая операция НЕ

Операция **НЕ** имеет один вход.

Она изменяет значение на противоположное:

если на входе 0, на выходе 1;
если на входе 1, на выходе 0.

Таблица истинности:

A	**Y = НЕ A**
0	1
1	0

Логическое выражение:

Y = ¬A

или

Y = NOT A

Логический элемент, выполняющий эту операцию, называется **инвертором**.

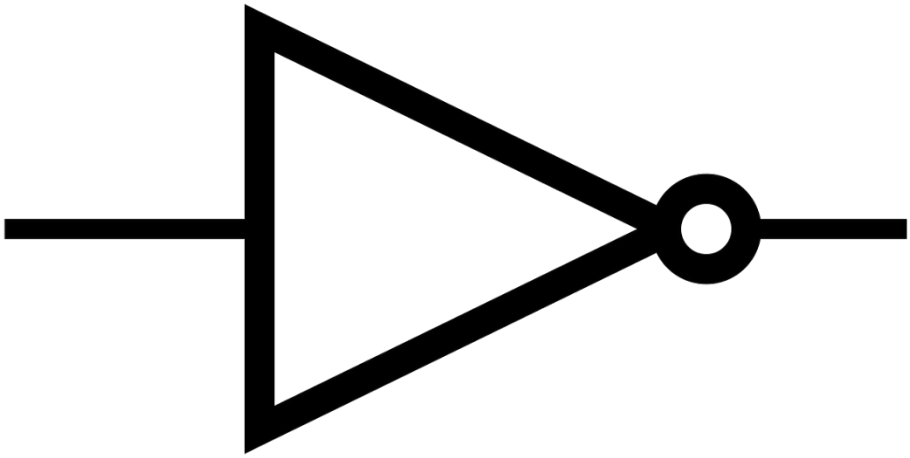


Рисунок 2 - Логический элемент НЕ

Логическая операция И

Операция **И** имеет два или более входа.

Для двух входов результат равен 1 только тогда, когда оба входа равны 1.

Таблица истинности:

A	**B**	**Y = A И B**
0	0	0
0	1	0
1	0	0
1	1	1

Логическое выражение: **Y = A ∧ B**

В программировании эту операцию можно сопоставить с логическим оператором AND.

Например:

A = 1

B = 1

A AND B = 1

Во всех остальных случаях результат равен 0.

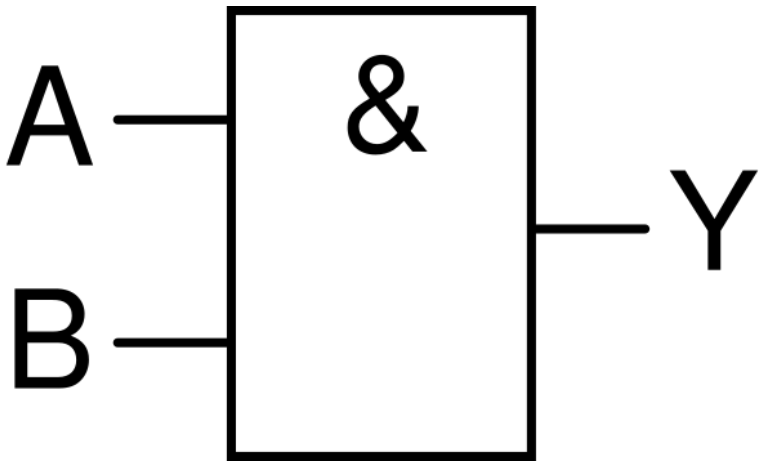


Рисунок 3 - Логический элемент И

Логическая операция ИЛИ

Операция **ИЛИ** также может иметь два или более входа.

Результат равен 1, если хотя бы один вход равен 1.

A	B	Y = A ИЛИ B
0	0	0
0	1	1
1	0	1
1	1	1

Логическое выражение:

Y = A ∨ B

В программировании аналогом является операция OR.

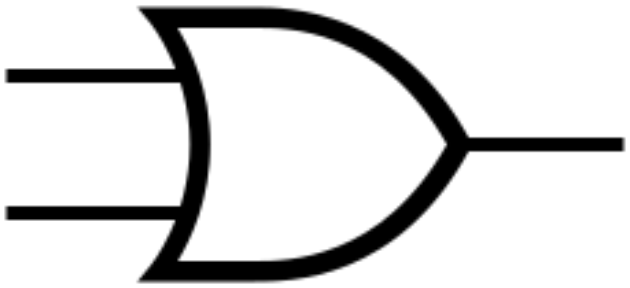


Рисунок 4 - Логический элемент ИЛИ

Исключающее ИЛИ — XOR

Особое значение для цифровых схем имеет операция **исключающее ИЛИ**, или **XOR**.

Её результат равен 1 тогда, когда входы различаются.

A	B	Y = A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Таким образом:

одинаковые входы → 0

разные входы → 1

Именно это свойство XOR делает его особенно важным при построении схем двоичного сложения.

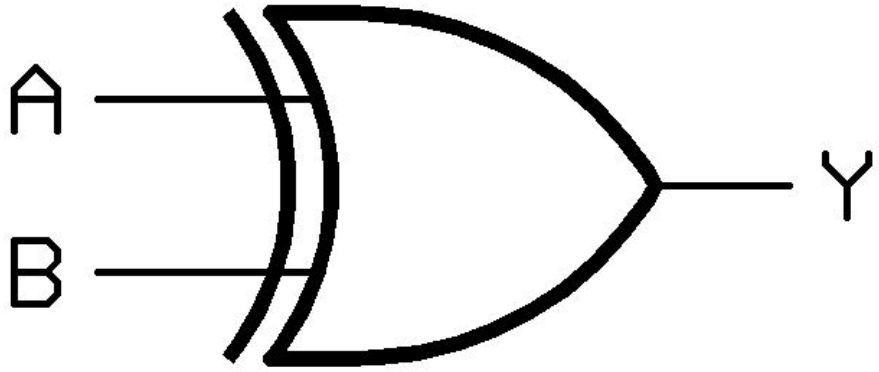


Рисунок 5 - Логический элемент XOR

Элементы И-НЕ и ИЛИ-НЕ

На практике используются также универсальные логические элементы.

И-НЕ

И-НЕ представляет собой комбинацию операций И и НЕ:

Y = ¬(A ∧ B)

Таблица истинности:

A	B	A И B	Y = И-НЕ
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Этот элемент называется **NAND**.

ИЛИ-НЕ

ИЛИ-НЕ:

Y = ¬(A ∨ B)

A	B	A ИЛИ B	Y = ИЛИ-НЕ
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

Этот элемент называется **NOR**.

Элементы NAND и NOR интересны тем, что являются **функционально полными**: используя только один из этих типов элементов, можно построить схемы, реализующие другие логические операции. В схемотехнике это имеет большое значение.

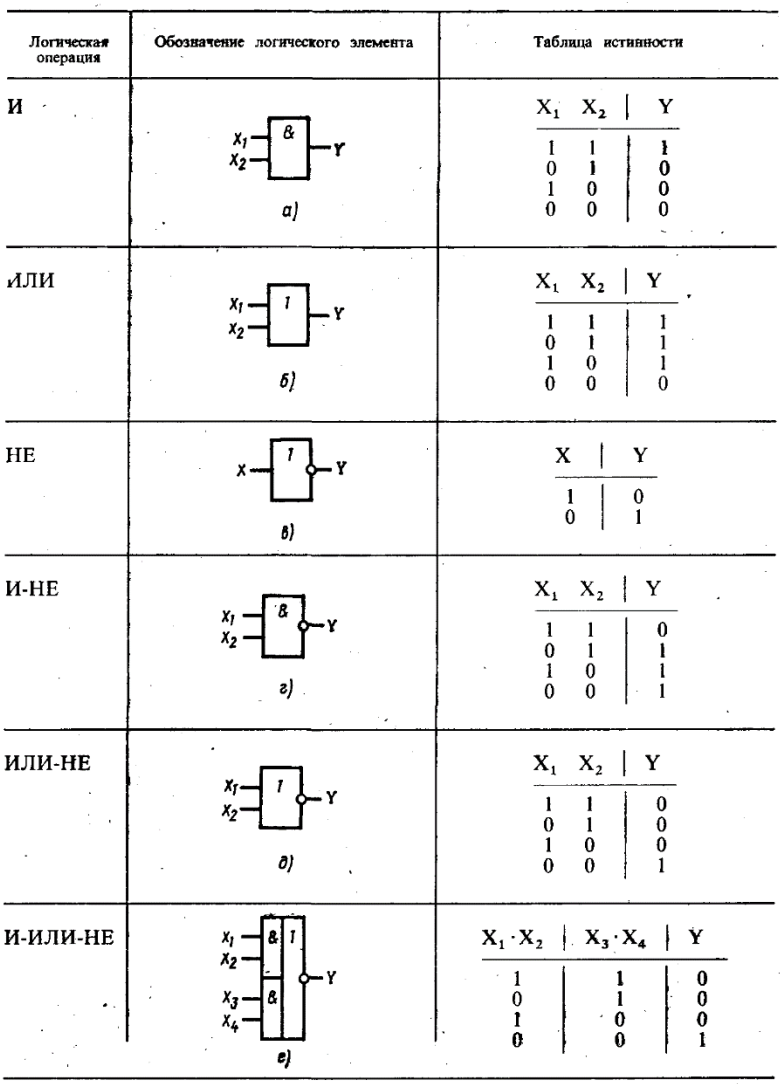


Рисунок 6 - Основные логические элементы

Логическая схема

Отдельный логический элемент выполняет одну простую операцию.

Однако компьютер должен выполнять гораздо более сложные действия.

Для этого несколько элементов соединяются между собой.

Получается **логическая схема**.

Например, выход одного элемента может поступать на вход другого:

A, B → AND → XOR → Y

В этом случае результат работы всей схемы зависит от последовательности преобразований сигнала.

Работу такой схемы можно описать:

1. логическими элементами;
2. логическим выражением;
3. таблицей истинности.

Таблица истинности позволяет полностью описать поведение комбинационной логической схемы для всех возможных входных комбинаций.

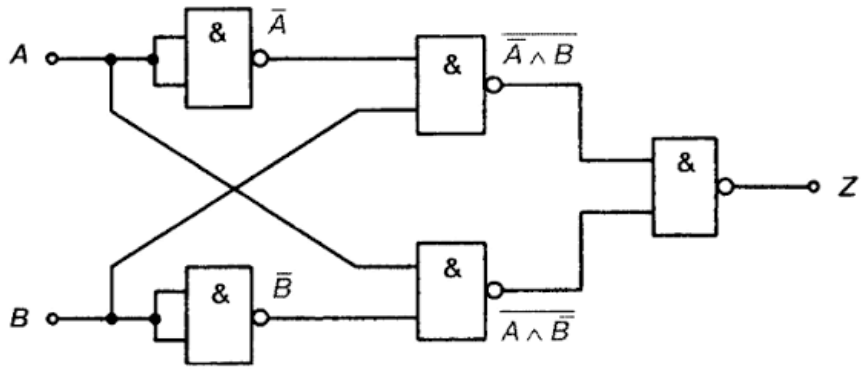


Рисунок 7 - Пример составной логической схемы

От логики к арифметике

На первый взгляд логические операции и арифметика относятся к разным областям.

Однако двоичное сложение можно построить с помощью тех же логических элементов.

Рассмотрим сложение двух однобитных чисел.

Возможны четыре варианта:

A	**B**	**Результат**
0	0	0
0	1	1
1	0	1
1	1	10_2

Последний случай особенно важен.

При сложении:

$1 + 1 = 10_2$

Получаются два бита:

0 — результат текущего разряда;

1 — перенос в следующий разряд.

Следовательно, схема должна уметь формировать **два результата**:

сумму;

перенос.

Полусумматор

Схема, которая складывает два однобитных числа без учёта входящего переноса, называется **полусумматором**.

У неё:

2 входа:

A;

B.

И:

2 выхода:

S — сумма;

C — перенос.

Для получения суммы используется XOR:

S = A XOR B

Для получения переноса используется AND:

C = A AND B

Получаем:

A	**B**	**S**	**C**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Таким образом, простой полусумматор можно построить всего из двух логических элементов:

XOR;

AND.

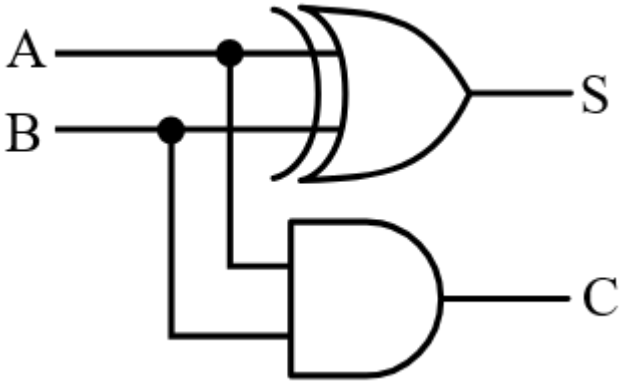


Рисунок 8 - Полусумматор

Полный сумматор

При сложении многоразрядных чисел необходимо учитывать перенос из предыдущего разряда.

Например:

0111

+ 0001

1000

При обработке каждого разряда необходимо учитывать не только два складываемых бита, но и перенос.

Поэтому полный одноразрядный сумматор имеет:

три входа:

A;

B;

Cin — входящий перенос.

И:

два выхода:

S — сумма;

Cout — выходящий перенос.

Таблица истинности:

A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1

A	B	Cin	S	Cout
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Эта таблица показывает, что полный сумматор реализует обычное двоичное сложение, только на уровне одного разряда. Такая таблица и схема полного сумматора приводятся, например, в материалах ИНТУИТ.

Для суммы можно записать:

S = A XOR B XOR Cin

Перенос:

Cout = (A AND B) OR (Cin AND (A XOR B))

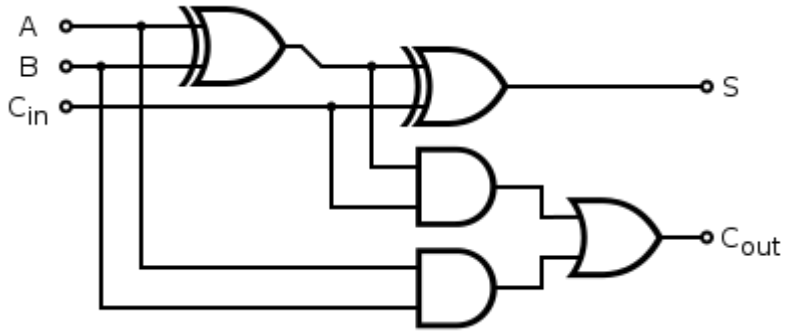


Рисунок 9 - Полный одноразрядный сумматор

Многоразрядный сумматор

Одного сумматора недостаточно для сложения обычных компьютерных чисел.

Например, число 25 в двоичной системе:

11001_2

Для обработки каждого разряда используется отдельный одноразрядный сумматор.

Несколько полных сумматоров соединяются последовательно.

Перенос с предыдущего разряда передаётся на вход следующего:

A0 B0 → [Сумматор 0] → S0

↓

C1

↓

A1 B1 → [Сумматор 1] → S1

↓

C2

↓

A2 B2 → [Сумматор 2] → S2

...

Таким образом, из простых одноразрядных схем можно построить устройство для сложения многоразрядных чисел.

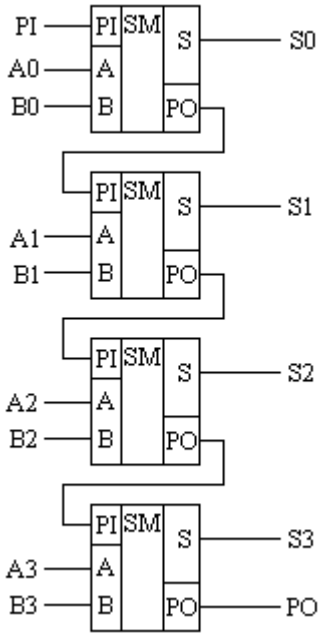


Рисунок 10 - Многоразрядный сумматор

Почему сумматор важен для процессора

Возникает закономерный вопрос:

Зачем мы так подробно рассматриваем сумматоры?

Потому что арифметические операции процессора в конечном счёте реализуются аппаратными схемами.

АЛУ — **арифметико-логическое устройство** — является частью процессора и выполняет арифметические и логические операции над данными.

К основным операциям относятся:

- сложение;
- вычитание;
- логическое И;
- логическое ИЛИ;
- XOR;
- инверсия;
- сравнение;
- сдвиги и другие операции.

Поэтому между программой и логическим элементом существует вполне конкретная цепочка:

программа

↓

машинная команда

↓

процессор

↓

АЛУ

↓

логические схемы

↓

логические элементы

↓

электрические сигналы

Это одна из основных идей данной темы.

Регистры — хранение данных внутри процессора

Логические схемы могут не только вычислять значения. Процессору необходимо также **хранить промежуточные результаты**. Для этого используются элементы памяти. Одним из фундаментальных элементов цифровой техники является **триггер**.

Триггер способен хранить одно из двух состояний:

0 или 1.

Следовательно:

1 триггер ≈ 1 бит информации.

Объединяя несколько таких элементов, можно получить устройство, способное хранить несколько битов.

Например:

8 бит → 8-разрядный регистр

Регистр представляет собой набор элементов памяти, предназначенный для хранения двоичного значения.

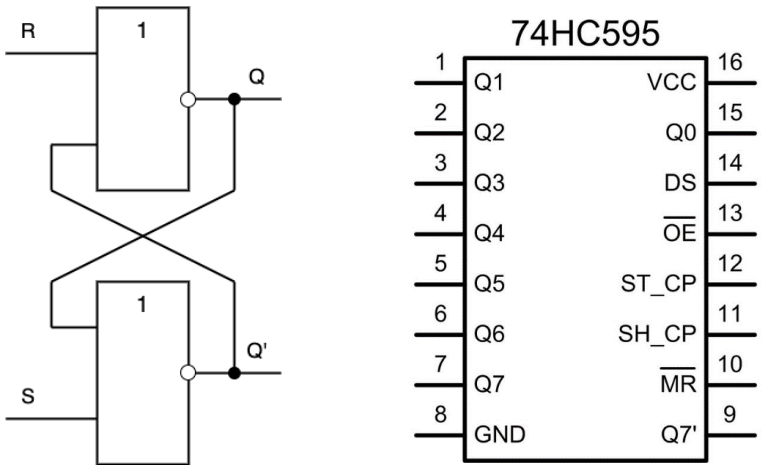


Рисунок 11 - Триггер и регистр

Арифметико-логическое устройство

Теперь можно объединить рассмотренные элементы в более крупную систему.

АЛУ — арифметико-логическое устройство — выполняет арифметические и логические операции над данными.

Упрощённо его можно представить как блок, который получает:

два операнда;

код операции.

И формирует:

результат;

дополнительные признаки результата.

Если код операции соответствует сложению, АЛУ использует соответствующую арифметическую схему.

Если выбрана операция AND, используются соответствующие логические схемы.

Таким образом, АЛУ можно рассматривать как **набор аппаратных средств для выполнения операций над двоичными данными**.

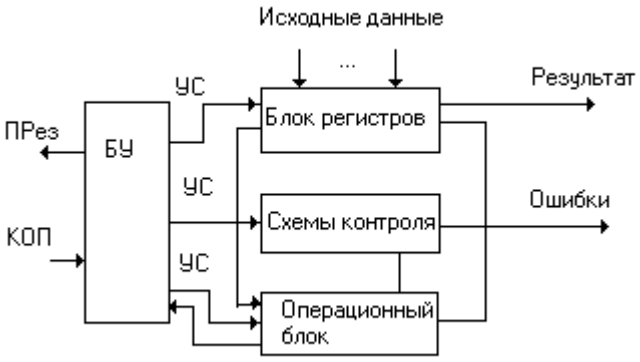


Рисунок 12 - Структура АЛУ

Как процессор выполняет команду

Теперь можно перейти от отдельных схем к работе процессора. Программа состоит из команд. Команда указывает процессору, **какое действие необходимо выполнить**.

Упрощённо выполнение команды можно представить четырьмя этапами:

1. **выборка команды**;
2. **декодирование команды**;
3. **выполнение команды**;
4. **запись результата**.

Этот процесс повторяется для следующих команд программы.

17.1. Выборка

Процессор получает очередную команду из памяти.

17.2. Декодирование

Процессор определяет, что означает полученная команда:

— какую операцию выполнить;

— какие данные использовать;

— куда записать результат.

17.3. Выполнение

Необходимые данные поступают в соответствующие узлы процессора.

Если требуется арифметическая или логическая операция, её выполняет АЛУ.

17.4. Запись результата

Полученный результат сохраняется в регистре или передаётся дальше.

Рисунок 13. Цикл выполнения команды

Вставить крупную круговую или линейную схему:

Память → Выборка → Декодирование → Выполнение → Запись результата → следующая команда

Внутри этапа «Выполнение» можно показать АЛУ.

Пример: что происходит при сложении

Рассмотрим упрощённый пример:

5 + 3

В двоичной системе:

5 = 0101

3 = 0011

Процессор должен выполнить операцию сложения.

Упрощённая последовательность:

1. Команда загружается из памяти.

↓

2. Процессор определяет, что требуется выполнить сложение.

↓

3. Значения 0101 и 0011 поступают в соответствующие регистры.

↓

4. АЛУ получает операнды и код операции сложения.

↓

5. Сумматоры внутри АЛУ выполняют двоичное сложение.

↓

6. Получается результат 1000_2.

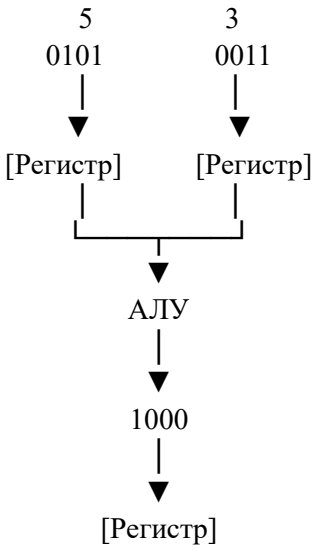
↓

7. Результат сохраняется в регистре или передаётся в другой узел процессора.

То есть даже такая простая операция, как:

5 + 3

на аппаратном уровне превращается в последовательность действий множества электронных схем.



Что происходит с условием программы

Та же система позволяет выполнять не только арифметические операции.

Рассмотрим условие:

```
if a > b:
    ...
```

Для человека это означает:

«если a больше b».

Для процессора необходимо получить аппаратный результат сравнения.

В зависимости от архитектуры сравнение может выполняться через арифметические операции, логические схемы и формирование специальных признаков результата.

В итоге процессор получает информацию, которая позволяет определить, нужно ли продолжать выполнение следующего участка программы или перейти к другой команде.

Таким образом, конструкция языка высокого уровня:

if

в конечном итоге связана с аппаратными механизмами:

сравнение → логическая/арифметическая схема → результат → изменение последовательности выполнения команд.

Это показывает, насколько далеко может простираться связь между языком программирования и аппаратной частью компьютера.

Общая структура процессора

Теперь можно объединить основные элементы.

Упрощённо процессор включает:

- **арифметико-логическое устройство (АЛУ)**;
- **регистры**;
- **устройство управления**;
- внутренние соединения для передачи данных и управляющих сигналов.

Упрощённая схема:

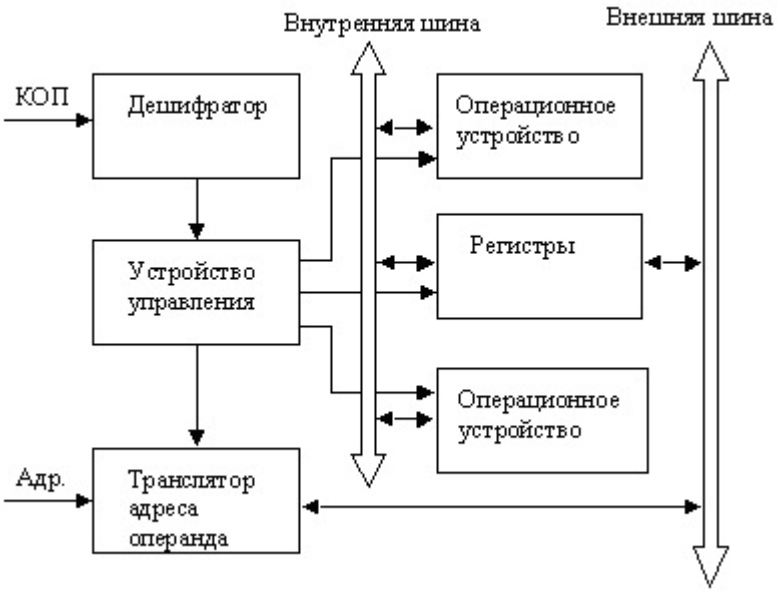


Рисунок 14 - Основные узлы процессора

Теперь можно представить всю рассмотренную систему как несколько уровней.

Уровень 1. Логические операции

НЕ, И, ИЛИ, XOR

↓

Уровень 2. Логические элементы

Физические электронные схемы, реализующие эти операции.

↓

Уровень 3. Комбинационные схемы

Несколько элементов объединяются для решения более сложной задачи.

Например:

полусумматор → полный сумматор → многоразрядный сумматор.

↓

Уровень 4. Функциональные узлы

Из логических схем строятся:

- сумматоры;
- регистры;
- счётчики;
- дешифраторы;
- мультиплексоры;
- другие узлы.

↓

Уровень 5. АЛУ и другие узлы процессора

Функциональные узлы объединяются для выполнения операций над данными.

↓

Уровень 6. Процессор

Процессор объединяет вычислительные, запоминающие и управляющие компоненты.

↓

Уровень 7. Выполнение программы

Процессор последовательно выполняет машинные команды программы.

Почему таблицы истинности имеют такое значение

Таблица истинности кажется простой: несколько нулей и единиц.

Однако она позволяет строго описать поведение цифровой схемы.

Например, для полного сумматора таблица содержит все возможные комбинации трёх входов.

Из неё можно определить:

— какой результат должна выдавать схема;

— когда возникает перенос;

— соответствует ли построенная схема требуемой функции.

Таким образом, таблица истинности является своеобразным **описанием поведения логической схемы**.

По таблице можно:

1. определить функцию;
2. построить логическое выражение;
3. построить логическую схему;
4. проверить правильность уже созданной схемы.

Именно поэтому таблицы истинности являются важным инструментом при проектировании цифровых устройств. Материалы по схемотехнической реализации ЭВМ также рассматривают таблицу истинности как один из исходных способов задания проектируемой комбинационной схемы.

Главное, что необходимо понять

Схемотехника компьютера может выглядеть очень сложной, если рассматривать процессор как единое устройство. Однако если двигаться снизу вверх, структура становится понятнее.

Простые логические операции позволяют создавать логические элементы.

Логические элементы объединяются в схемы.

Из схем создаются сумматоры, регистры и другие функциональные узлы.

Из функциональных узлов строятся АЛУ и другие части процессора.

Процессор объединяет эти узлы и выполняет машинные команды.

Поэтому между простой логической операцией и работой программы существует непосредственная связь:

логический элемент → схема → функциональный узел → процессор → машинная команда → программа.

Компьютер не является «чёрным ящиком», который каким-то неизвестным образом выполняет программы. Его работа основана на огромном количестве относительно простых физических и логических операций, организованных в сложную систему.

Основные термины

Логический сигнал — физический сигнал, которому сопоставляется логическое значение 0 или 1.

Логический элемент — электронная схема, выполняющая определённую логическую операцию.

Таблица истинности — таблица, описывающая результат логической операции или схемы для всех возможных комбинаций входных значений.

Логическая схема — совокупность соединённых между собой логических элементов.

Полусумматор — схема для сложения двух однобитных чисел без входящего переноса.

Полный сумматор — схема для сложения двух битов с учётом входящего переноса.

Регистр — устройство, предназначенное для хранения двоичной информации.

АЛУ — арифметико-логическое устройство процессора, выполняющее арифметические и логические операции.

Процессор — устройство, выполняющее машинные команды программы и управляющее обработкой данных.

Цикл выполнения команды — последовательность действий процессора по выборке, декодированию и выполнению команды.