Тема 2. Классификация языков программирования. Язык программирования C#. Основные элементы языка. Структура программы.

Учебные вопросы:

Введение. Основные понятия.

1. Классификация языков программирования
2. Язык программирования C#.
3. Основные элементы языка C#.
4. Структура программы на C# .

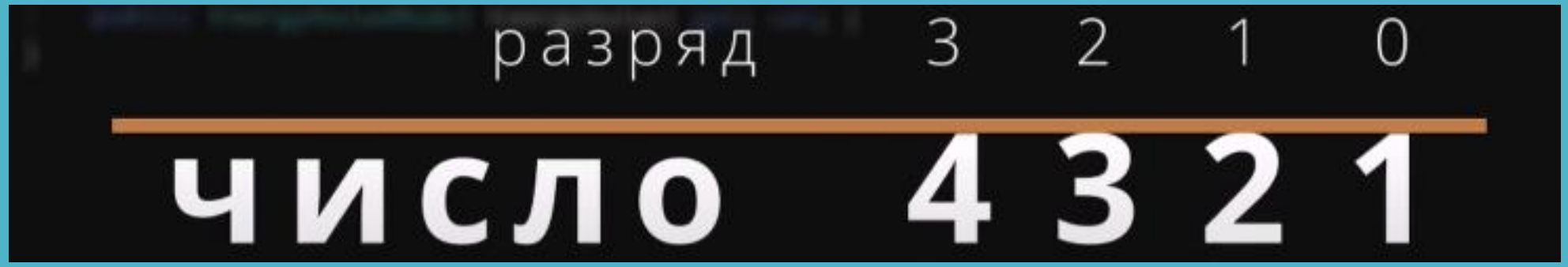
Введение. Основные понятия.

Системы счисления

Система счисления – это способ записи чисел с помощью определенных символов, называемых цифрами.

Позиционные системы: В таких системах значение цифры зависит от ее позиции в числе. Например, в десятичной системе цифра "2" в числе "25" означает 2 десятка, а в числе "52" – всего 2 единицы.

Эта позиция называется разрядом. Нумерация разрядов начинается с «0» и производится справа налево.



Основание – это количество различных цифр, используемых в системе счисления.

Десятичная система имеет основание 10 (цифры от 0 до 9).

Двоичная система (используется в компьютерах) имеет основание 2 (цифры 0 и 1).

Шестнадцатеричная система имеет основание 16 (цифры от 0 до 9 и буквы A, B, C, D, E, F).

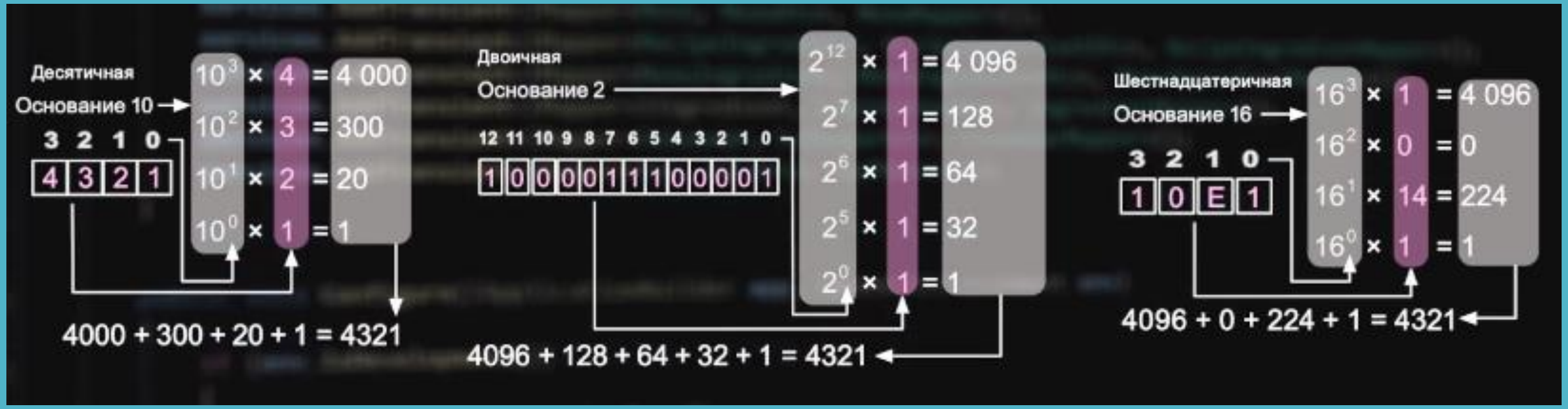
Система счисления (основание)	Символы
Десятичная (10)	0,1,2,3,4,5,6,7,8,9
Двоичная (2)	0,1
Шестнадцатеричная (16)	0,1,2,3,4,5,6,7,8,9, A,B,C,D,E,F

10, 11, 12, 13, 14, 15

Система счисления	Представление числа “4321”
Десятичная	4321
Двоичная	1000011100001
Шестнадцатеричная	10E1

Перевод из десятичной системы в любую другую. Число делится на основание новой системы счисления, остаток от деления становится последней цифрой в новом представлении числа. Процесс повторяется с частным от деления, пока оно не станет меньше основания.

Перевод из любой системы счисления в десятичную. Каждая цифра числа умножается на соответствующую степень основания системы счисления, а затем полученные значения суммируются.



Единицы измерения количества информации

Единицы измерения количества информации используются для определения размера файлов, емкости носителей данных и других величин, связанных с цифровыми данными.

Основные единицы измерения

- **Бит (bit):** Самая маленькая единица информации. Может принимать два значения: 0 или 1.
- **Байт (byte):** Состоит из 8 бит. Это основная единица для измерения размера текстовых файлов, изображений и других небольших данных.
- **Кбайт (Килобайт):** Равен 1024 байта. Используется для измерения размера более крупных файлов, таких как короткие аудиозаписи или небольшие документы.
- **Мбайт (Мегабайт):** Равен 1024 килобайта. Используется для измерения размера фотографий, небольших видеороликов и других средних по размеру файлов.

	Биты							
Байт	1	0	0	1	0	0	1	1

двоичная | десятичная

11111111 = 255

0..255

1 байт = 8 бит

1 килобайт (Кб) = 2^{10} байт = 1024 байт

1 мегабайт (Мб) = 2^{10} Кб = 1024 Кб

1 гигабайт (Гб) = 2^{10} Мб = 1024 Мб

1 терабайт (Тб) = 2^{10} Гб = 1024 Гб

Компьютерная архитектура

Компьютерная архитектура - это концептуальная структура компьютера, которая определяет, как он работает на самом базовом уровне. Она описывает организацию компонентов компьютера, взаимодействие между ними и способы выполнения инструкций.

Компьютерная архитектура – это, по сути, план или схема, по которой строится компьютер.

Это как чертеж дома: он определяет, из каких частей состоит здание, как они соединены между собой и как работают.

Основные компоненты компьютера (10 минут)

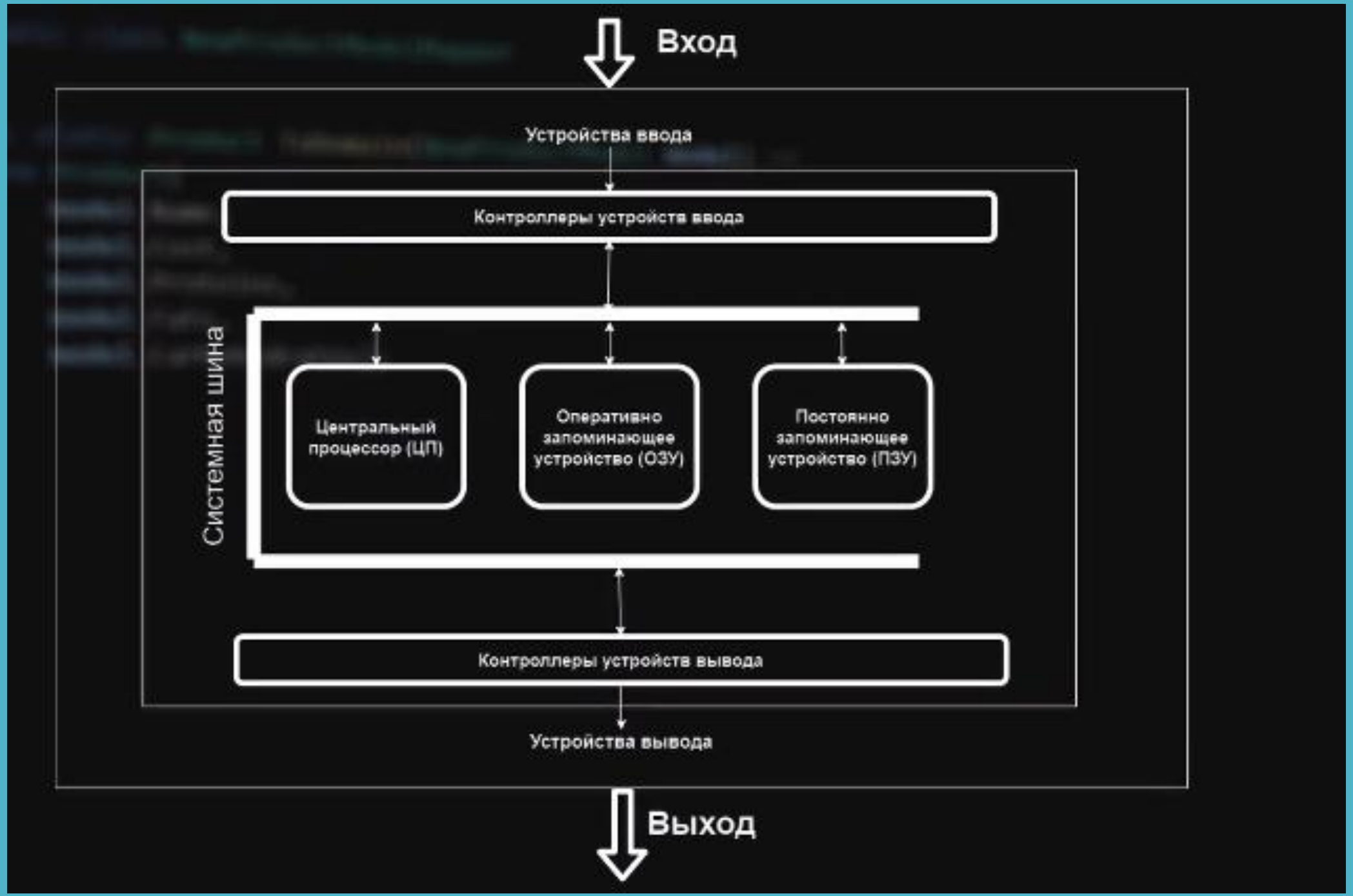
1.Процессор (Центральный процессор, CPU): "Мозг" компьютера, который выполняет инструкции программ.

2.Оперативная память (RAM): Временное хранилище данных, с которыми работает процессор. Быстрая, но данные в ней теряются при выключении компьютера.

3.Жёсткий диск (HDD) или твердотельный накопитель (SSD): Постоянное (энергонезависимое) хранилище данных. Значительно медленнее чем RAM.

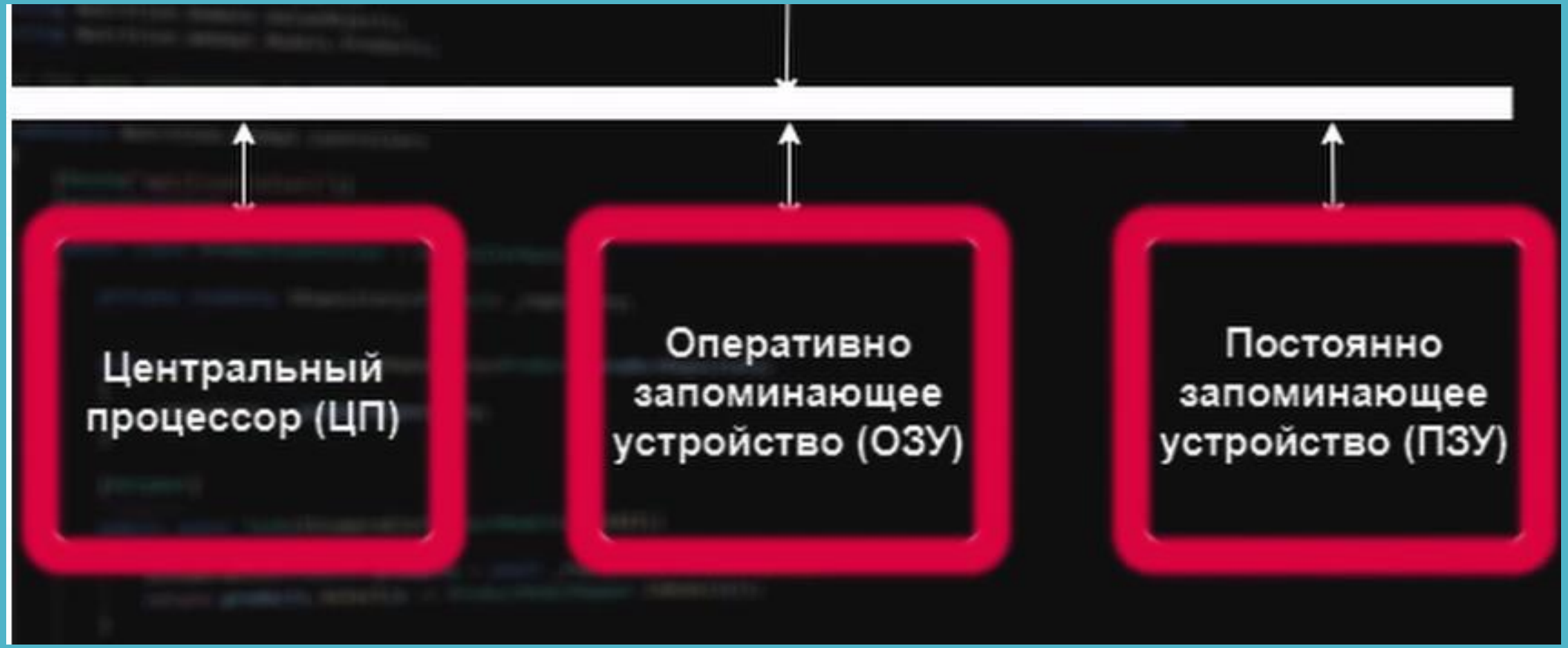
4.Материнская плата: "Основная плата" компьютера, к которой подключаются все компоненты.

5.Графический процессор (GPU): Специализированный процессор для обработки графики и изображений.


Вход
Устройства ввода
Контроллеры устройств ввода
Системная шина
Центральный процессор (ЦП)
Оперативно запоминающее устройство (ОЗУ)
Постоянно запоминающее устройство (ПЗУ)
Контроллеры устройств вывода
Устройства вывода
Выход

Контроллеры устройств ввода - это специализированные микросхемы, которые управляют работой устройств ввода, таких как клавиатура, мышь, сканер, микрофон и т.д. Они преобразуют сигналы от этих устройств в формат, понятный для компьютера, и передают их в процессор.

Контроллеры устройств вывода - это специализированные микросхемы, которые управляют работой устройств вывода, таких как монитор, принтер, динамики, проектор и т.д. Они принимают данные от процессора, преобразуют их в формат, понятный устройству вывода, и управляют его работой.


Центральный процессор (ЦП)
Оперативно запоминающее устройство (ОЗУ)
Постоянно запоминающее устройство (ПЗУ)

Процесс выполнения программы:

Программа загружается с жёсткого диска в оперативную память.

Процессор считывает инструкции программы из оперативной памяти и выполняет их.

Процессор может считывать и записывать данные в оперативную память по мере необходимости.

Хранение и передача данных:

Данные, с которыми работает процессор, временно хранятся в оперативной памяти.

Долговременные данные хранятся на жёстком диске или SSD.

Для передачи данных между компонентами используется системная шина (bus).


Процессор
ОЗУ
ПЗУ

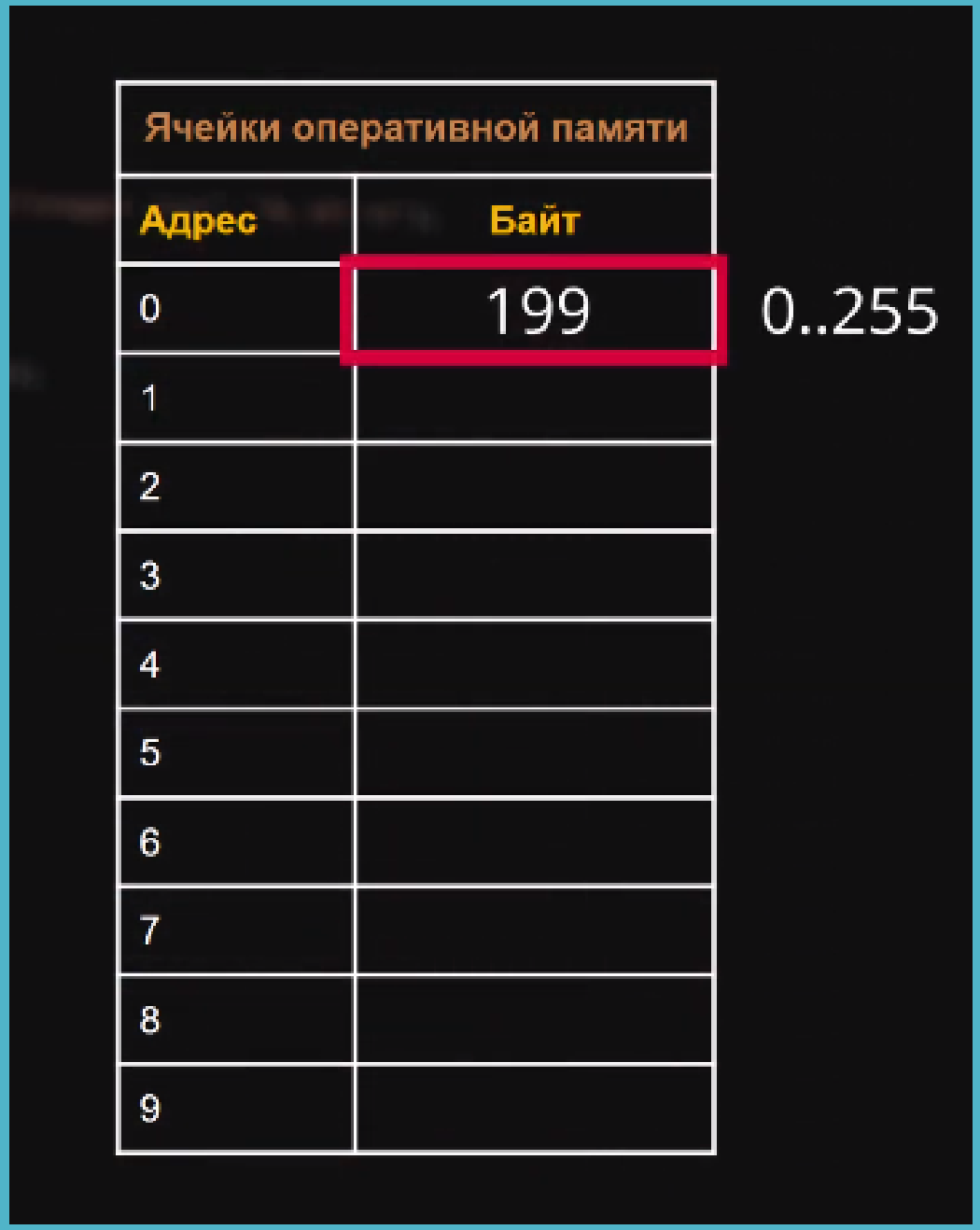
Оперативная память

Оперативная память это множество ячеек, каждая из которых может хранить **1 байт** данных и имеет свой числовой адрес.

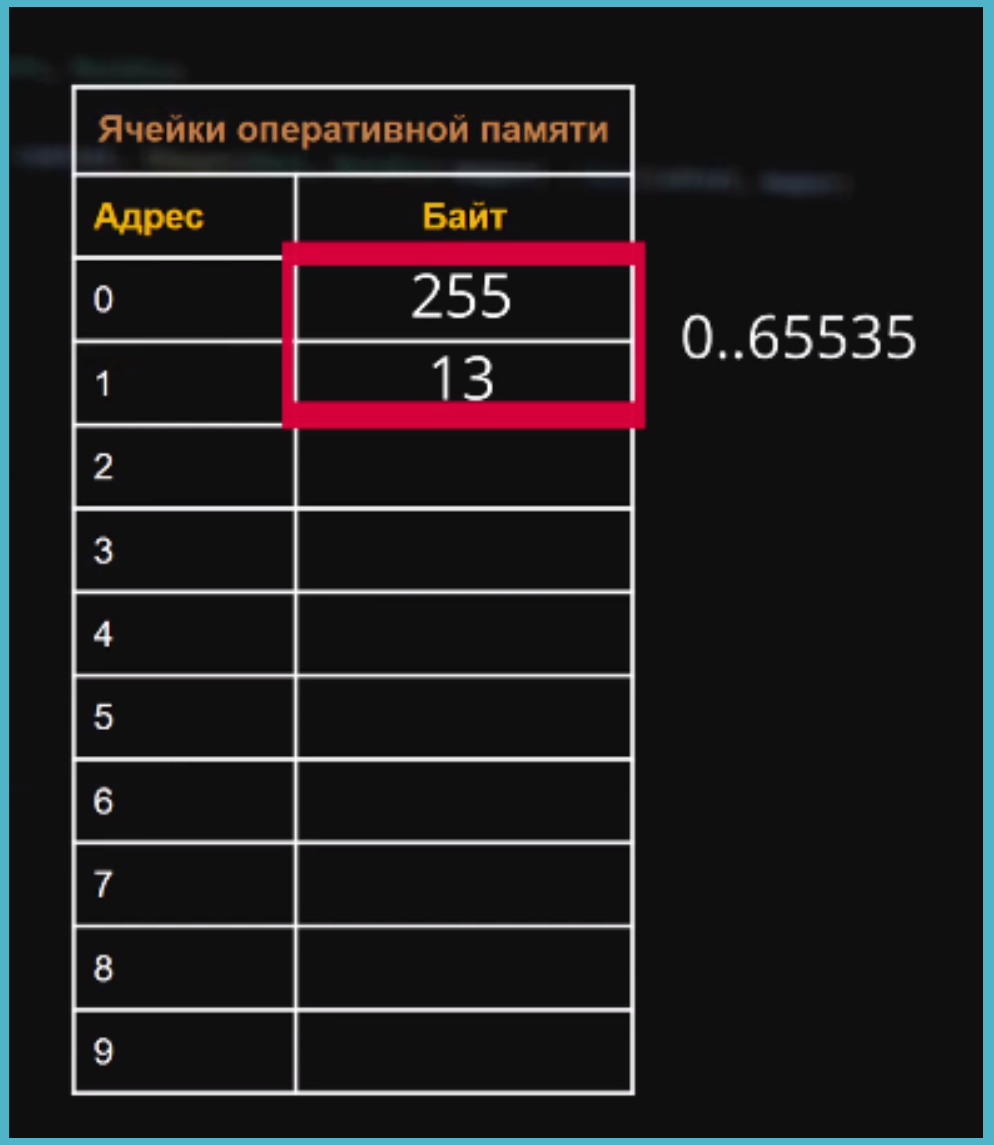
Ячейки оперативной памяти	
Адрес	**Байт**
0	124
1	2
2	0
3	0
4	32

С помощью адреса можно считывать и записывать информацию в ячейку.

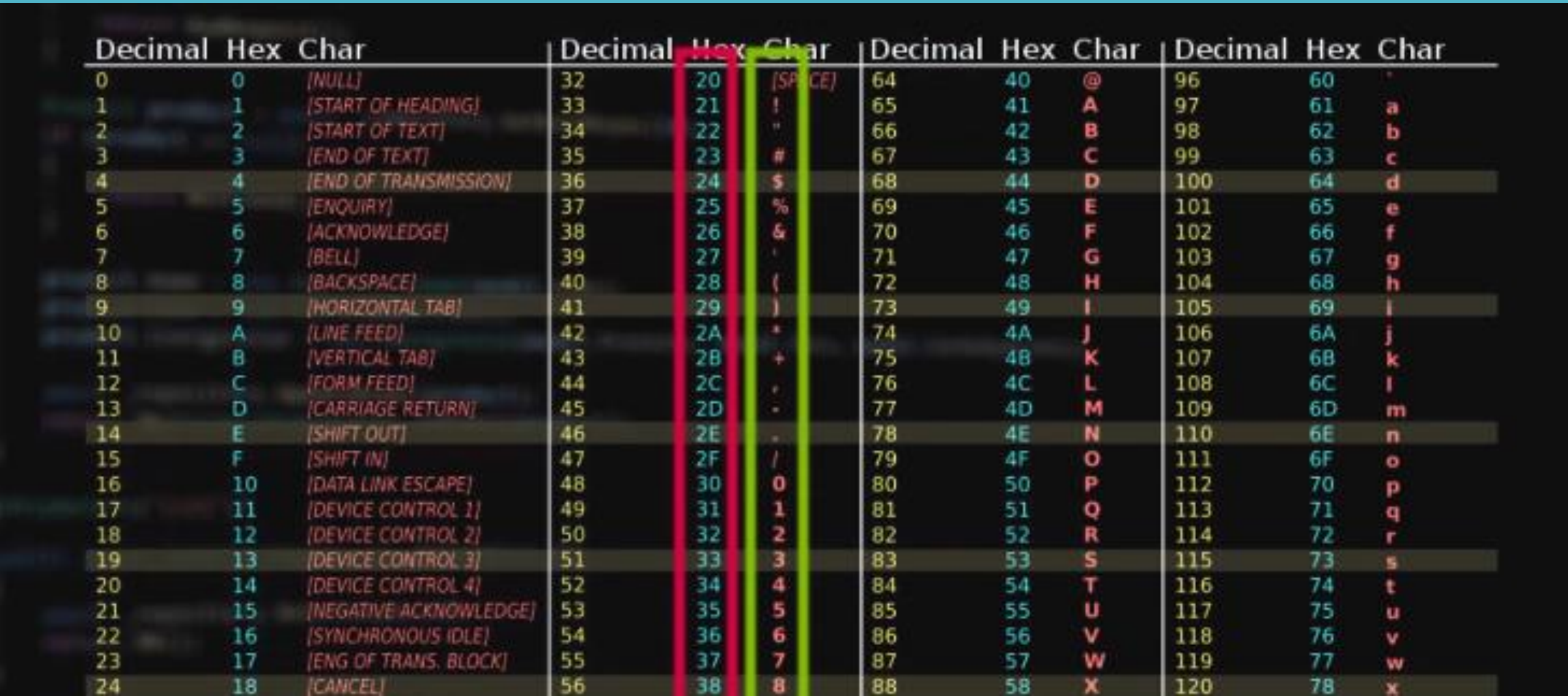
1 байт может хранить число от 0 до 255.



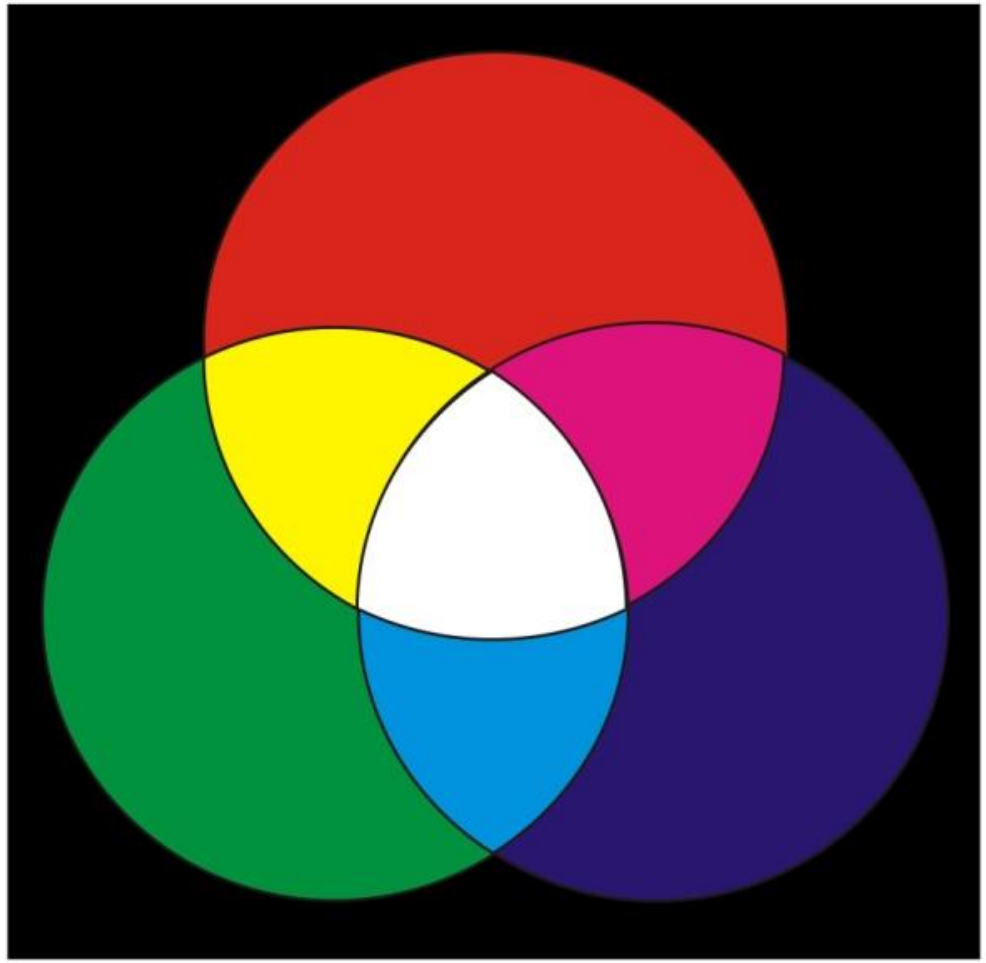
Если нужно хранить большее число, то можно использовать несколько байт (2, 4, 8, 16…)



В цифровом виде можно представлять любые данные, в т.ч. текст и цвет.



Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x



	R	G	B
Красный	255	0	0
Зеленый	0	255	0
Синий	0	0	255
Черный	0	0	0
Белый	255	255	255
Желтый	255	255	0
Пурпурный	255	0	255
Голубой	0	255	255

Устройство центрального процессора и взаимодействие с памятью

Центральный процессор (ЦПУ или CPU) - это "мозг" компьютера. Он выполняет все вычисления и управляет работой других компонентов системы.

Основные компоненты ЦПУ:

- **Арифметико-логическое устройство (АЛУ):** Выполняет арифметические (сложение, вычитание и т.д.) и логические (сравнение, инверсия и т.д.) операции над данными.
- **Устройство управления:** Координирует работу всех компонентов процессора, извлекает инструкции из памяти, декодирует их и управляет их исполнением.
- **Регистры:** Небольшие области памяти внутри процессора, используемые для хранения промежуточных результатов вычислений, адресов памяти и других данных, к которым требуется быстрый доступ.
- **Кэш-память:** Быстрая память небольшого объема, которая хранит копии данных из основной памяти. Это позволяет процессору быстрее получать доступ к часто используемой информации.

Взаимодействие процессора с памятью

Процессор постоянно взаимодействует с оперативной памятью (ОЗУ), где хранятся программы и данные. Этот процесс можно представить следующим образом:

1. **Извлечение инструкции:** Устройство управления извлекает из памяти следующую инструкцию для выполнения.
2. **Декодирование инструкции:** Инструкция декодируется, чтобы определить, какое действие должен выполнить процессор.
3. **Получение операндов:** Если для выполнения инструкции необходимы данные, они извлекаются из памяти или регистров.
4. **Выполнение инструкции:** АЛУ выполняет указанную инструкцию над полученными операндами.
5. **Запись результата:** Результат операции записывается в регистр или память.
6. **Переход к следующей инструкции:** Устройство управления переходит к выполнению следующей инструкции.

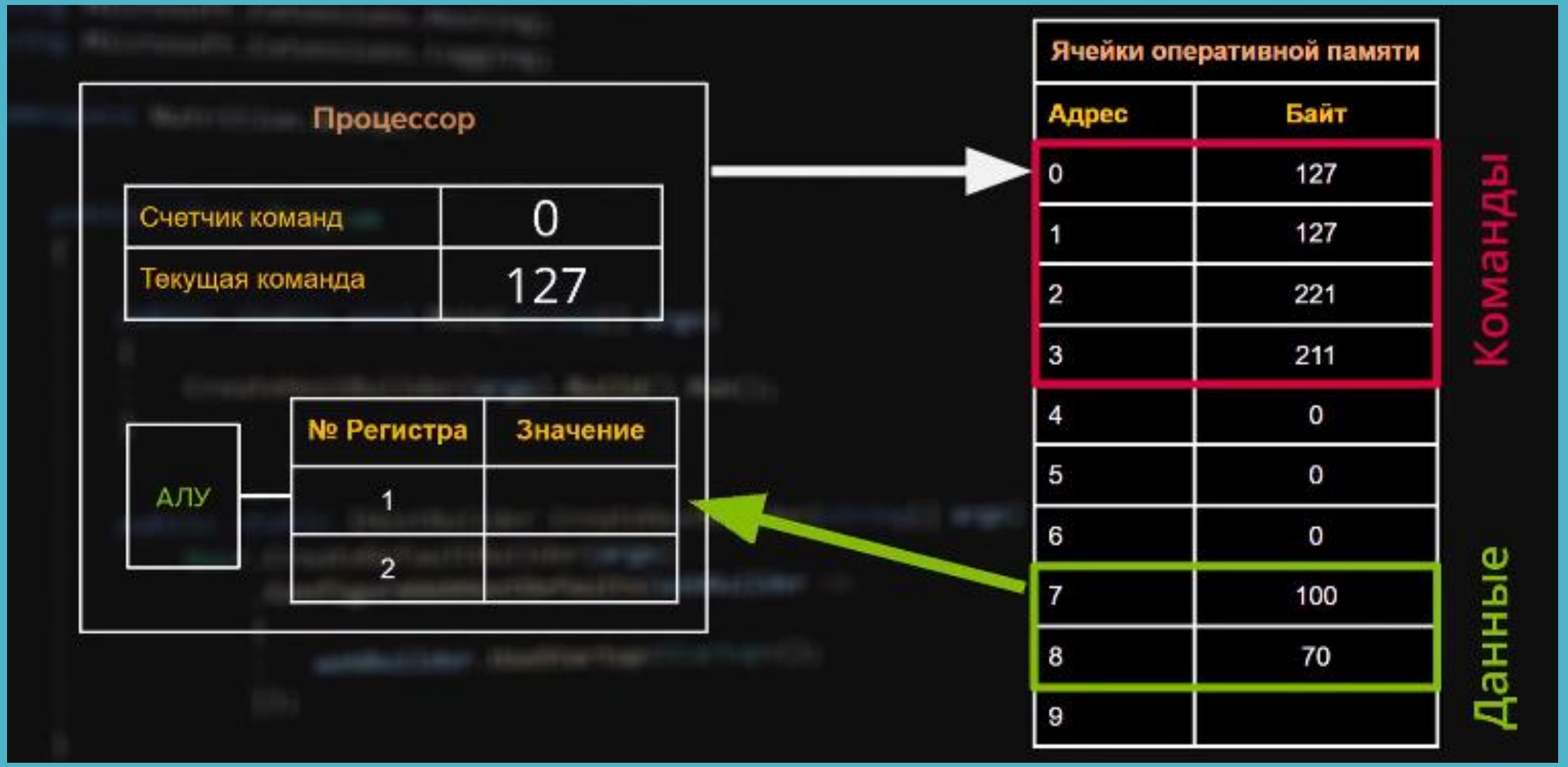
Процессор состоит из двух основных компонентов: арифметико-логическое устройство (АЛУ) и регистров (ячек памяти).

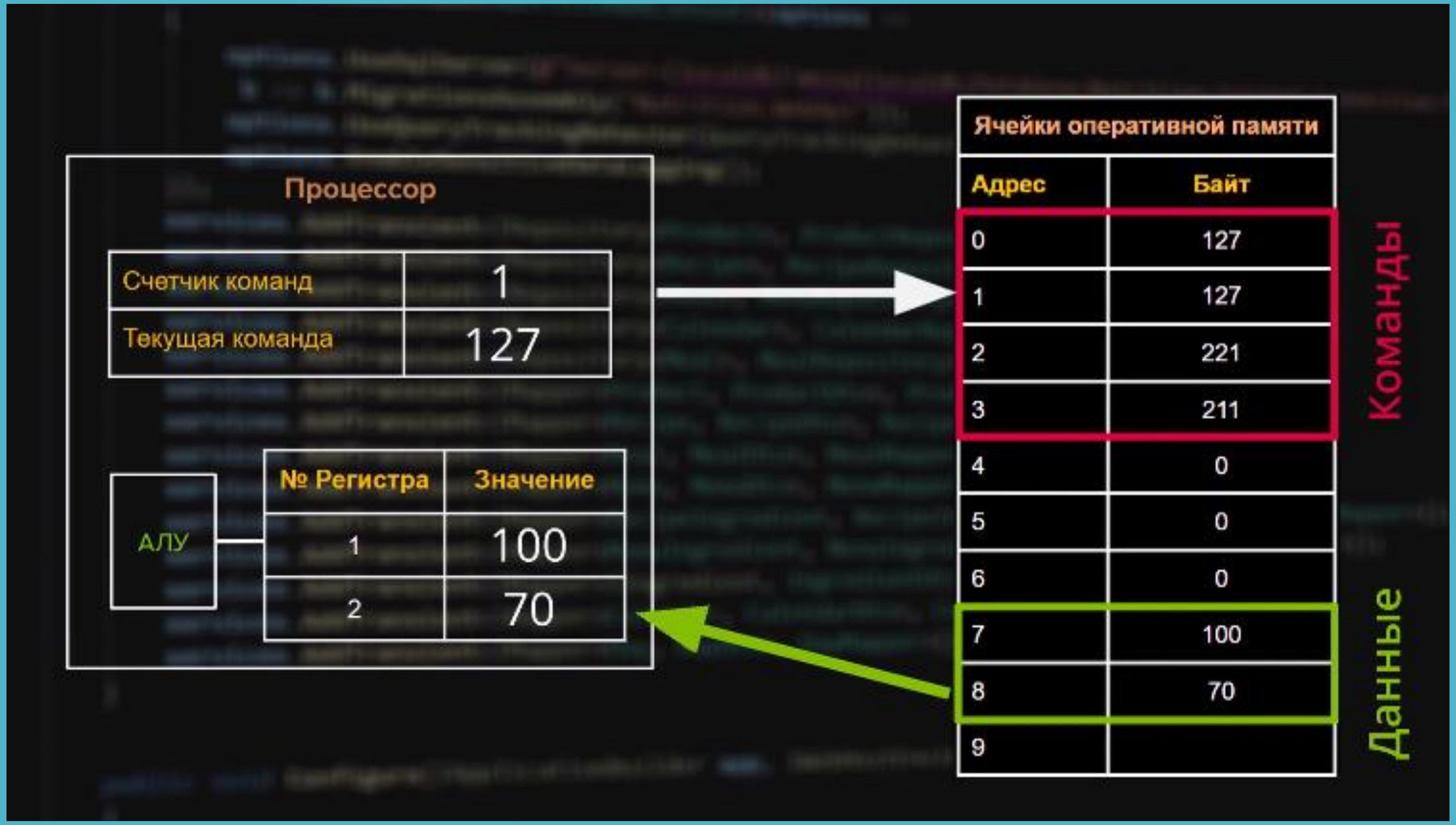
Для того чтобы поместить данные из памяти в регистры и выполнить над ними операции существует специальный набор команд. Программа состоит из комбинации таких команд.

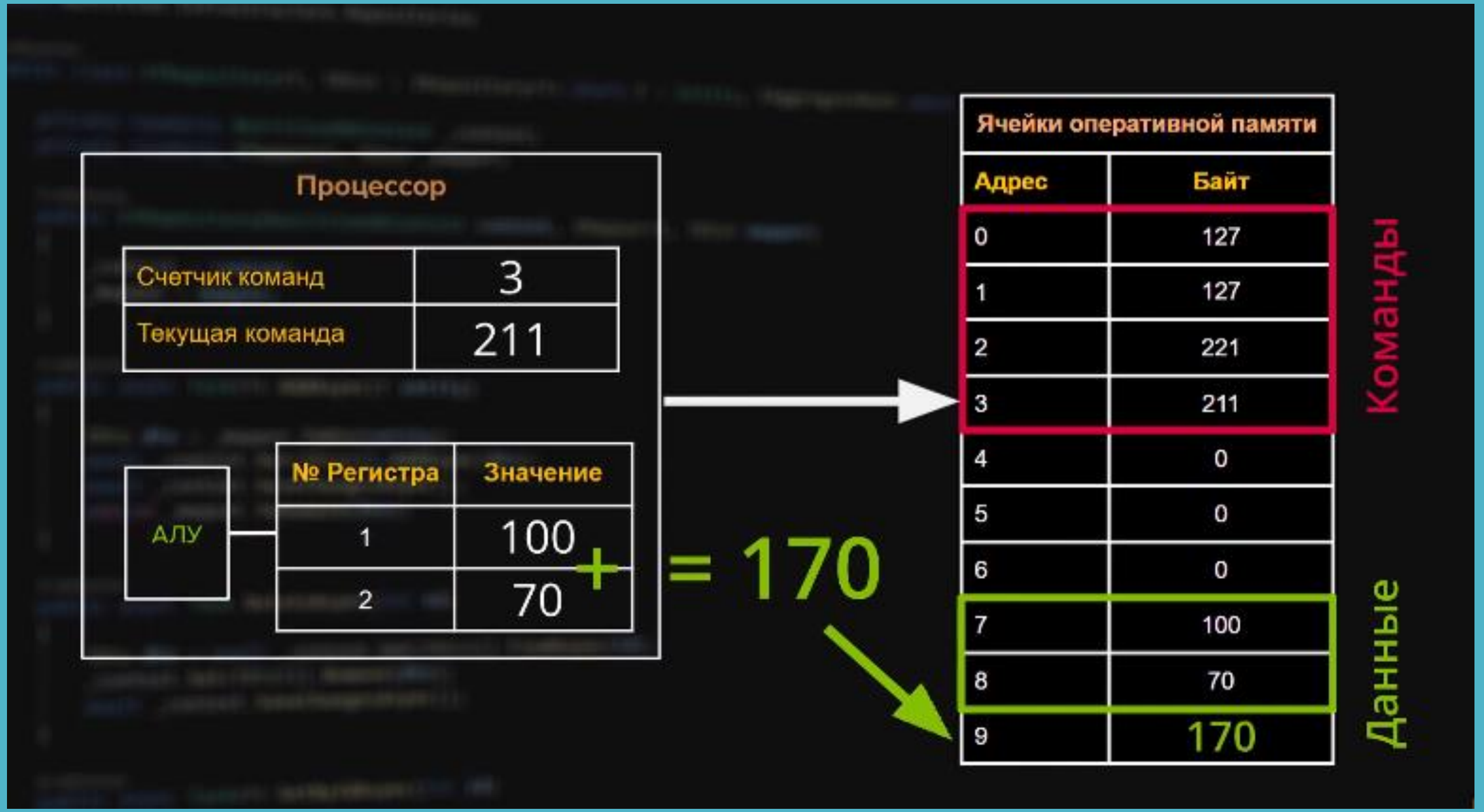
Номер команды	Описание команды
127	Поместить данные из оперативной памяти в регистр процессора
221	Сложить данные в регистрах
211	Поместить данные из регистров процессора в оперативную память

Программа, также как и данные, хранится в памяти.

Чтобы перемещаться между командами существует регистр, называемый счетчиком команд.




Процессор
Счетчик команд
1
Текущая команда
127
АЛУ
№ Регистра
Значение
1
100
2
70
Ячейки оперативной памяти
Адрес
Байт
0
127
1
127
2
221
3
211
4
0
5
0
6
0
7
100
8
70
9
Команды
Данные


Процессор
Счетчик команд
3
Текущая команда
211
АЛУ
№ Регистра
Значение
1
100
2
70
+
= 170
Ячейки оперативной памяти
Адрес
Байт
0
127
1
127
2
221
3
211
4
0
5
0
6
0
7
100
8
70
9
170
Команды
Данные

Программные средства

Центральный процессор понимает только машинные инструкции (набор двоичных цифр).

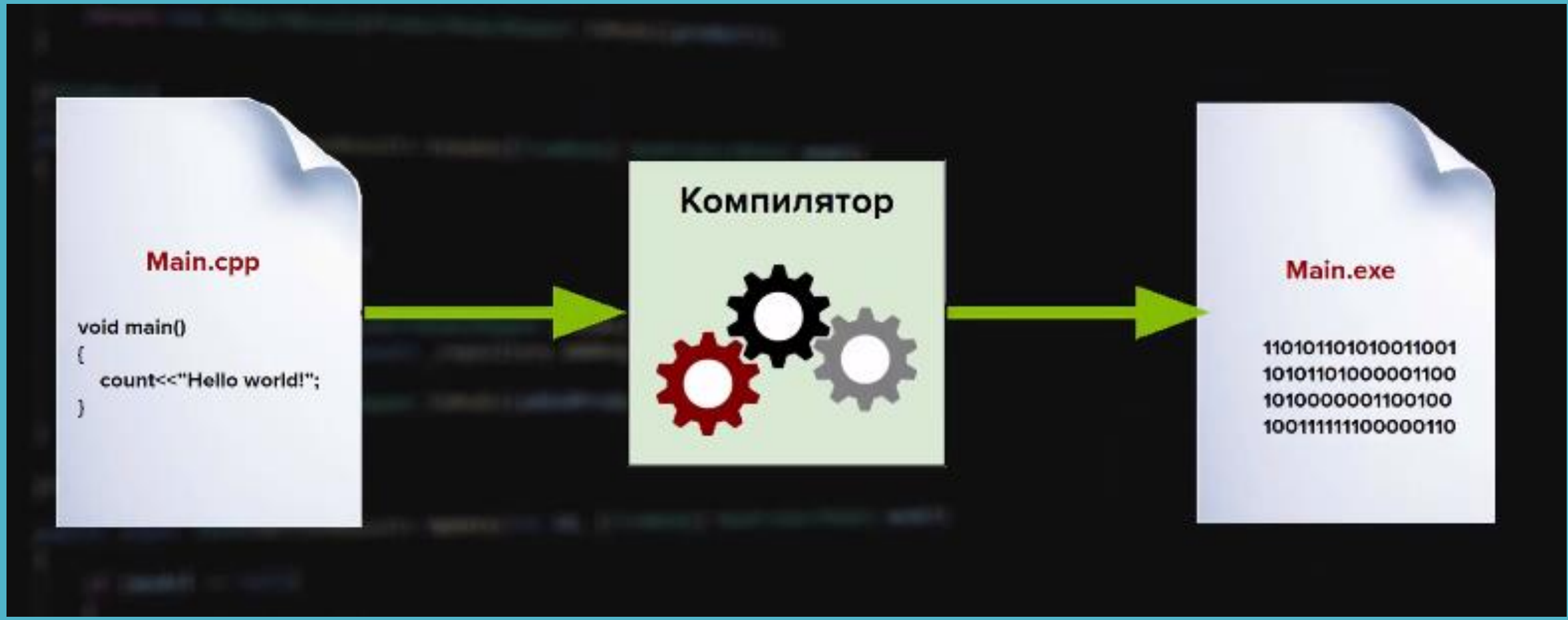
Языки программирования позволяют писать текст программы, понятный для человека.

Для преобразования такого текста в машинные инструкции, существуют специальные программы (компиляторы и интерпретаторы).

Компилятор – это программа, которая полностью переводит весь исходный код на язык машинных инструкций (машинный код) за один проход.

Полученный машинный код сохраняется в отдельном файле (обычно с расширением **.exe** для Windows) и может быть запущен в любое время без повторной компиляции.

Компиляция – это как перевод целой книги на другой язык.


Main.cpp
void main()
{
count<<"Hello world!";
}
Компилятор
Main.exe
1101011010010011001
1010110100000110 0
1010000001100100
1001111111000001 10

Преимущества компиляции:

- **Высокая скорость выполнения:** Машинный код выполняется непосредственно процессором без дополнительных преобразований.
- **Независимость от компилятора:** После компиляции программа может выполняться на любой машине с совместимым процессором.

Недостатки компиляции:

- **Длительное время компиляции:** Особенно для больших проектов.
- **Привязка к конкретной платформе:** Машинный код, скомпилированный для одной архитектуры процессора, обычно не совместим с другой.

Схема взаимодействия между исходным кодом, компилятором .NET и механизмом выполнения .NET.

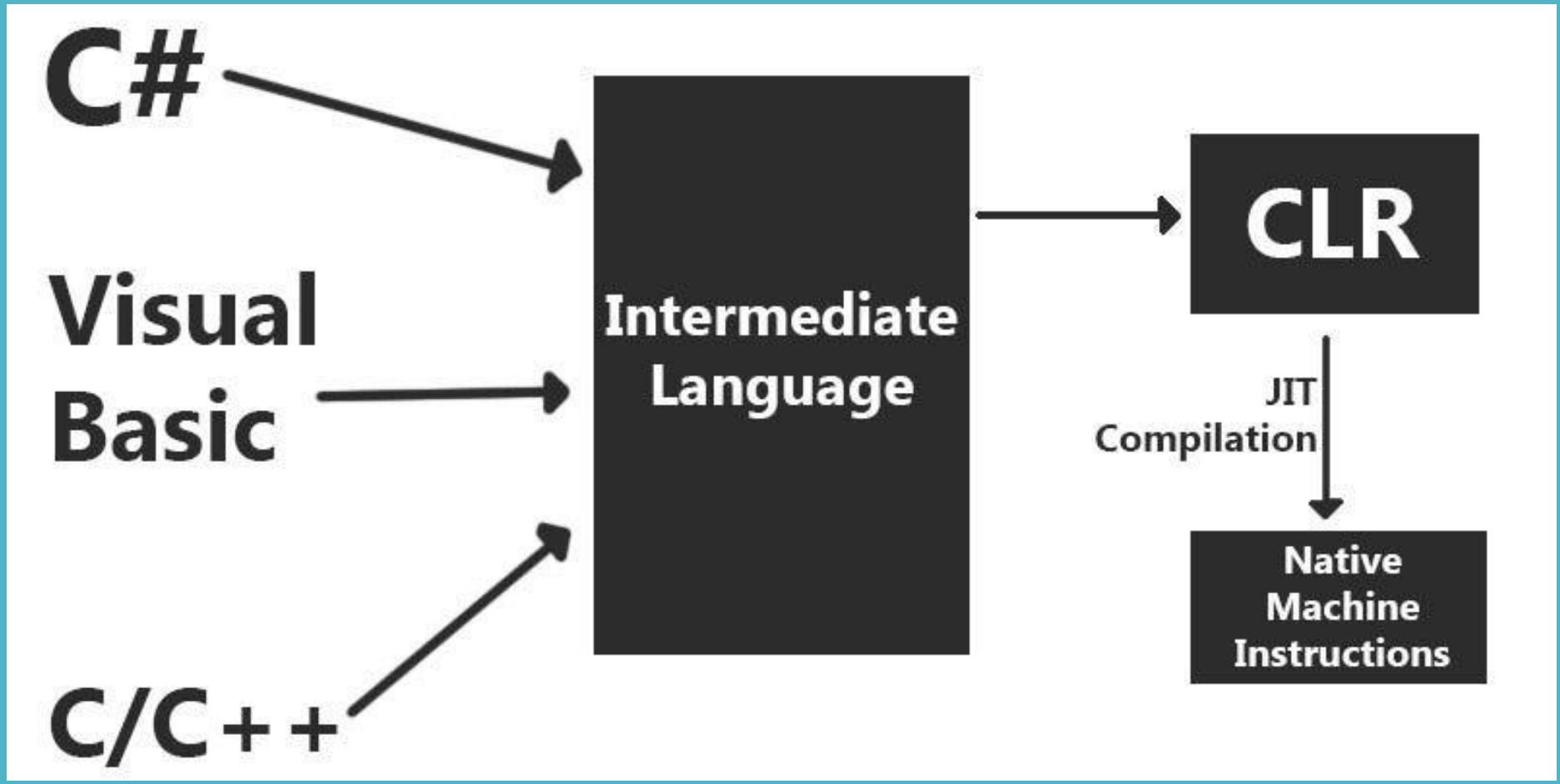
Программист создает исходный код приложения на языке, который поддерживает технологию .NET (языке C#, C++/CLI, Visual Basic .NET и т.д.). Приложение создается в некоторой среде программирования, например Microsoft Visual Studio. Компилятор формирует сборку – файл, который содержит **CIL**-инструкции, метаданные и манифест.

После запуска на выполнение этого приложения на некотором компьютере (некоторой платформе), в работу запускается механизм выполнения .NET. Предварительно, на компьютере должна быть установлена одна из версий (как минимум) .NET Framework.

JIT-компилятор осуществляет компиляцию сборки с учетом (привязкой) аппаратных и программных особенностей компьютера, на котором происходит запуск приложения.

После этого приложение выполняется.

CIL (Common Intermediate Language) - это язык промежуточного кода, используемый в .NET Framework и .NET.



Интерпретатор – это программа, которая анализирует и выполняет исходный код **построчно**.

Он не создает отдельный машинный файл, а выполняет код "на лету".

Это можно сравнить с синхронным переводчиком, который переводит речь говорящего сразу же, слово за слово.


Исходный текст программы
1. a = 5;
2. b = 10;
3. c = a + b;
4. print c;
5. ...
Интерпретатор
1. mov a, 5
2. mov b, 10
3. mov c, a; add c, b
4. push c; call print
5. ...
выполнить
выполнить
выполнить
выполнить
выполнить

Преимущества интерпретации:

- **Быстрая разработка:** Изменения в коде можно сразу увидеть, нет необходимости каждый раз перекомпилировать всю программу.
- **Портативность:** Интерпретаторы часто используются для создания кроссплатформенных приложений.

Недостатки интерпретации:

- **Низкая скорость выполнения:** Каждая строка кода анализируется и выполняется интерпретатором заново при каждом запуске программы.
- **Зависимость от интерпретатора:** Для выполнения программы всегда нужен интерпретатор.

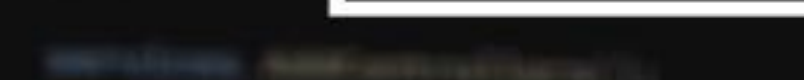
Операционная система

Операционная система (ОС) - это набор программ, которые управляют компьютером и позволяют запускать на нем другие программы.

Что делает операционная система?

- **Управляет аппаратными ресурсами:** Распределяет процессорное время, память, устройства ввода-вывода (клавиатура, мышь, принтер и т.д.) между различными программами.
- **Обеспечивает интерфейс для пользователя:** Позволяет пользователю взаимодействовать с компьютером через графический интерфейс (например, Windows, macOS) или командную строку.
- **Запускает программы:** Загружает и запускает программы, предоставляя им необходимые ресурсы.
- **Обеспечивает файловую систему:** Организует данные на жестком диске и других носителях информации, позволяя пользователю создавать, удалять и изменять файлы.
- **Обеспечивает безопасность:** Защищает компьютер от несанкционированного доступа и вредоносных программ.

Физические устройтсва



Операционная система

Прикладная программа

Функции ОС

- Работа с файловой системой
- Передача данных по сети
- Работа с устройствами ввода-вывода
- Взаимодействие приложений

1. Классификация языков программирования.

По уровню абстракции:

- **Низкоуровневые языки:**
  - Язык ассемблера.
  - Характеристики: близки к машинному коду, высокоэффективны, но сложны для понимания.
- **Высокоуровневые языки:**
  - C, C++, Java, Python, C#.
  - Характеристики: проще для понимания и использования, абстрагированы от аппаратного обеспечения.

По парадигме программирования

•**Императивные языки:** Описывают последовательность действий, которые должен выполнить компьютер. Большинство языков программирования относятся к этому типу.

- **Примеры:** C, C++, Java, Python.

•**Объектно-ориентированные языки:** Представляют данные и действия над ними в виде объектов.

- **Примеры:** C++, Java, Python, C#.

•**Функциональные языки:** Описывают вычисления как применение функций.

- **Примеры:** Haskell, Lisp, F#.

•**Логические языки:** Используются для решения задач, основанных на формальной логике.

- **Примеры:** Prolog.

Мультипарадигменные языки: поддерживают несколько парадигм программирования.

- **Примеры:** C++, Java, Python, C#.

Классификация по области применения

- **Системное программирование:** C, C++.
- **Веб-разработка:** JavaScript, PHP, Python, Ruby.
- **Научные расчеты:** Python, MATLAB, R.
- **Разработка мобильных приложений:** Swift (iOS), Kotlin (Android).
- **Базы данных:** SQL.
- **Искусственный интеллект:** Python, Lisp.
- **Игровое программирование:** C++, C#.

Компилируемые и интерпретируемые.

Компилируемые: C, C++, Pascal
Интерпретируемые: Visual Basic Script (VBScript), JavaScript, Python, PHP
Условно компилируемые: C# и остальные языки .Net, Java для Java-машины

Выбор языка программирования

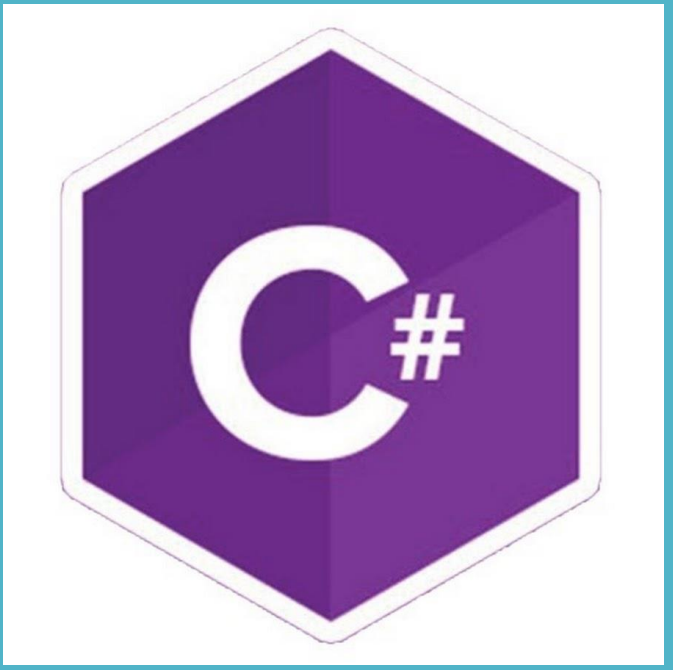
Выбор языка программирования зависит от следующих факторов:

- **Задача:** Для каждой задачи есть наиболее подходящие языки.
- **Производительность:** Для ресурсоемких задач могут потребоваться низкоуровневые языки.
- **Читаемость и поддержка:** Чем проще код, тем легче его поддерживать и модифицировать.
- **Сообщество:** Большое сообщество разработчиков может обеспечить хорошую поддержку и множество библиотек.
- **Личные предпочтения:** Выбор языка также может зависеть от личных предпочтений программиста.

2. Язык программирования C#.

C# (произносится "си шарп") - это современный, популярный, универсальный, объектно-ориентированный, компилируемый язык программирования, разработанный компанией Microsoft.

Он был создан в конце 90-х годов как часть инициативы **.NET** Framework и с тех пор претерпел значительную эволюцию.

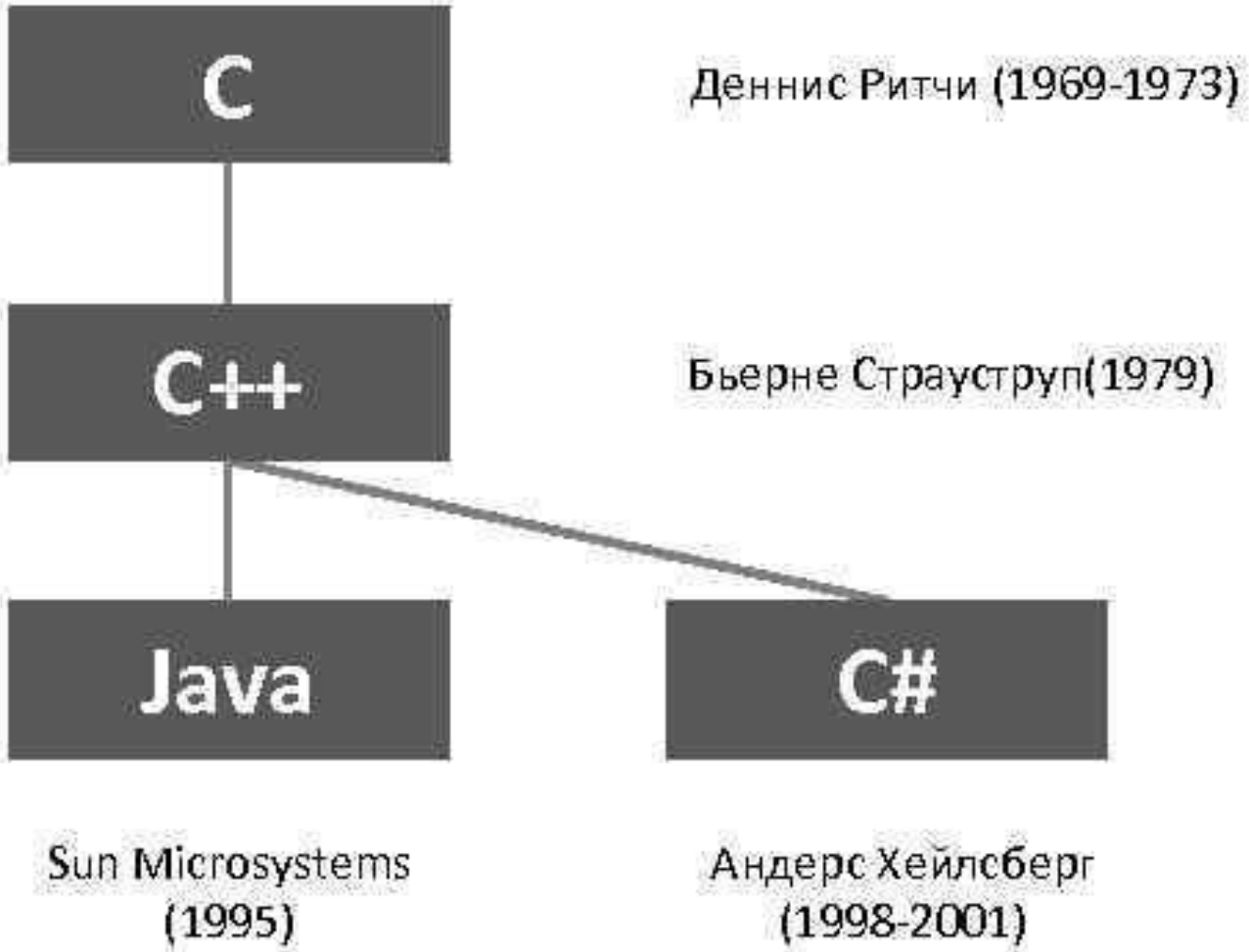


Корнями C# уходит в C и C++: Синтаксис **C#** во многом схож с языками **C** и **C++**, что делает его знакомым для разработчиков, имеющих опыт работы с этими языками.

Влияние Java: C# также перенял многие идеи и концепции из языка **Java**, например, автоматическое управление памятью и строгую типизацию.

.NET Framework: Язык C# был создан специально для платформы .NET Framework, которая предоставляет богатый набор инструментов и библиотек для разработки различных типов приложений.

История развития C++



Почему C# так популярен?

- **Мощный и универсальный:** C# позволяет разрабатывать широкий спектр приложений, от небольших консольных программ до крупных корпоративных систем.
- **Высокая производительность:** Компилятор C# генерирует высокоэффективный код.
- **Безопасность:** Строгая типизация и сборка мусора помогают предотвращать распространенные ошибки программирования.
- **Большое сообщество:** Существует огромное количество ресурсов, библиотек и инструментов для разработки на C#.
- **Поддержка корпораций:** Microsoft активно развивает C# и .NET Framework, обеспечивая стабильность и совместимость.

Области применения C#

- **Разработка Windows-приложений:** Десктопные приложения, игры.
- **Веб-разработка:** ASP.NET для создания динамических веб-сайтов.
- **Мобильная разработка:** Xamarin для создания кроссплатформенных мобильных приложений.
- **Игры:** Unity и другие игровые движки.
- **Бэкенд-разработка:** Создание серверной части веб-приложений.
- **Научные расчеты:** Благодаря библиотекам, таким как Math.NET.
- **Облачные приложения** на Microsoft Azure

Слабые стороны C#:

- **Ориентированность на Windows.** Несмотря на кроссплатформенность, C# все равно ассоциируется с Windows, что в некоторых случаях может усложнить разработку мультиплатформенных приложений.
- **Производительность.** В определенных ситуациях C# будет менее производительным, чем другие низкоуровневые языки программирования, такие как C++.
- **Зависимость от .NET Framework.** Приложения, написанные на C#, обычно требуют наличие .NET Framework на компьютере пользователя, что иногда создает проблемы совместимости.
- **Замкнутость на Microsoft.** Развитие C# и его инструментов контролируется Microsoft, что может вызвать определенные ограничения и зависимость от компании.

3. Основные элементы языка C#.

Типы данных

C# — язык со строгой статической типизацией. Это означает, что каждая переменная должна иметь определенный тип данных, и этот тип нельзя изменить во время выполнения программы. Основные типы данных включают:

- **Целочисленные со знаком:** sbyte, short, int, long.
- **Целочисленные без знака:** byte, ushort, uint, ulong
- **Вещественные:** float, double, decimal.
- **Символьные:** char.
- **Логические:** bool.
- **Строковые:** string.

Переменные

Переменная — это именованная область памяти, используемая для хранения данных определенного типа.

В C# переменные объявляются с указанием типа и присваиванием начального значения.

```
int age = 30;
string name = "Иван";
double pi = 3.14159;
```

Операторы

Операторы используются для выполнения различных действий над значениями. Существуют арифметические, логические, сравнения и другие операторы.

```
int x = 10, y = 5;
x = 33; // Присвоение
int sum = x + y; // Сложение
bool isGreater = x > y; // Сравнение
```

4. Структура программы на C# .

Программа на C# представляет собой набор инструкций, которые компьютер должен выполнить в определенной последовательности.

Эти инструкции организованы в определенной структуре, которая помогает сделать код более читаемым, понятным и поддерживаемым.

Основные компоненты программы на C#:

Пространства имен (namespaces)

Цель: Организация кода в логические группы, предотвращение конфликтов имен.

Синтаксис: using System;

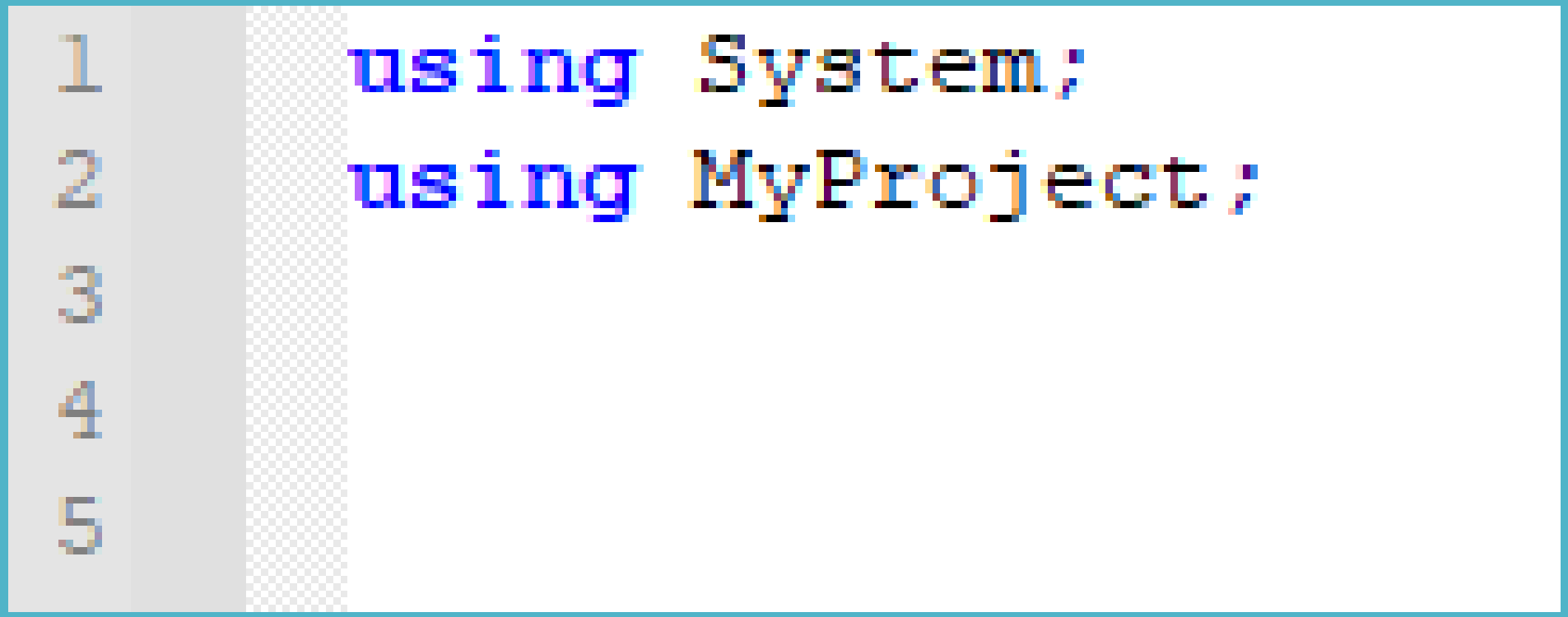
Пример: Пространство имен System содержит множество стандартных классов и типов данных, таких как Console, Math и другие.

Пространство имен объявляется с помощью ключевого слова **namespace** и фигурных скобок, внутри которых размещается код, относящийся к этому пространству.

```
1  namespace MyProject
2  {
3      // Код, относящийся к пространству имен
4      class MyClass
5      {
6          // ...
7      }
8  }
```

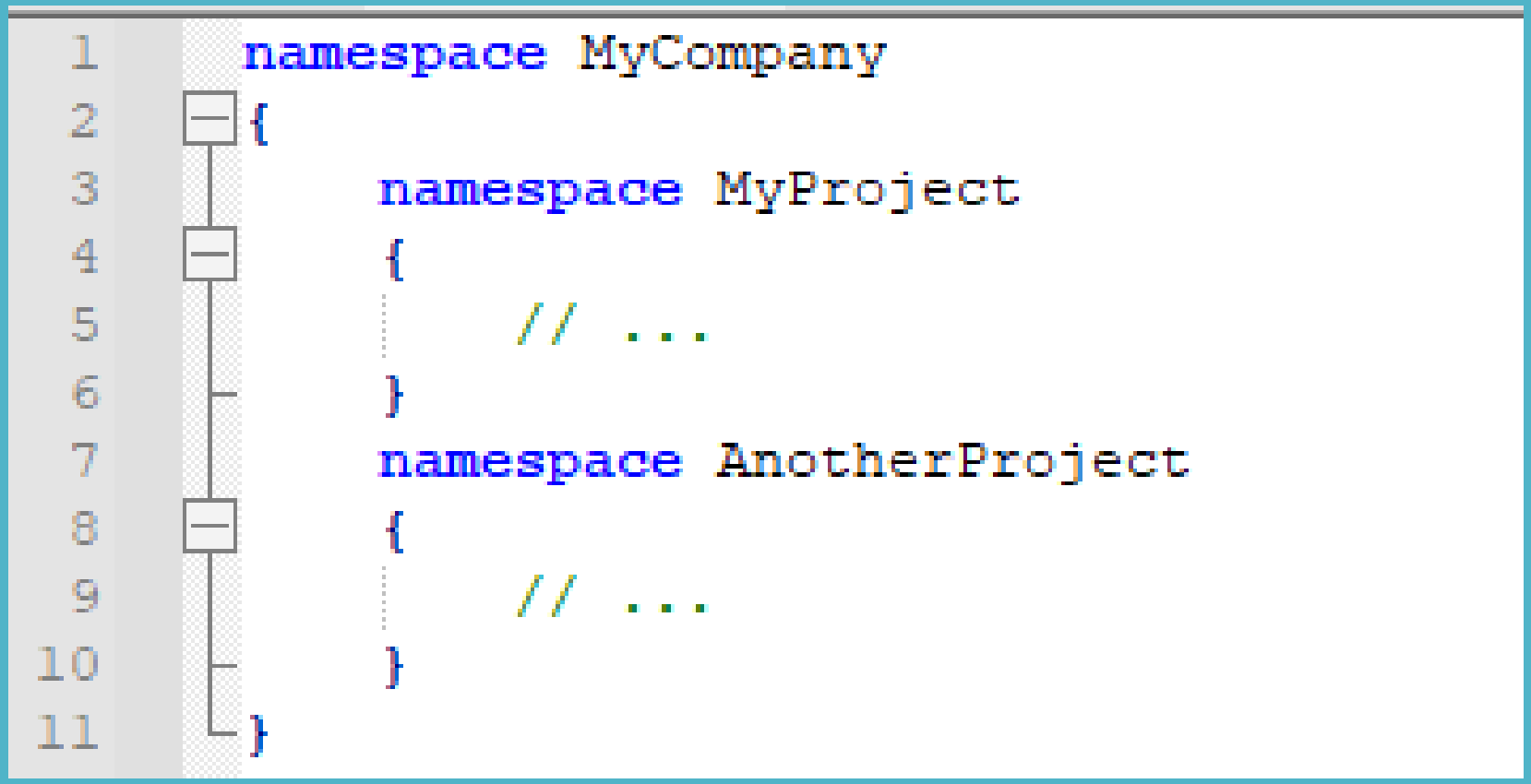
Директива using:

Для работы с элементами из других пространств имен, можно использовать директиву **using**.



Вложенные пространства имен:

Пространства имен могут быть вложенными, что позволяет создавать более иерархическую структуру кода.



```
1  namespace MyCompany
2  {
3      namespace MyProject
4      {
5          // ...
6      }
7      namespace AnotherProject
8      {
9          // ...
10     }
11 }
```

Стандартные пространства имен в C#:

System: Содержит фундаментальные классы, такие как Console, String, Math и другие.

System.Collections: Содержит классы для работы с коллекциями данных (списки, словари и т.д.).

System.IO: Содержит классы для работы с файлами и потоками.

System.Net: Содержит классы для работы с сетью.

Классы:

- Являются основными строительными блоками в C#.
- Определяют **свойства** (data members) и **методы** (functions), которые описывают поведение объектов.
- Каждый объект в C# является экземпляром какого-либо класса.

```
1   public class MyClass
2   {
3       // Поля, свойства, методы
4   }
```

Методы (функции):

Блоки кода, выполняющие определенные задачи.

Имеют имя, список параметров и тело (блок кода).

Метод **Main** является точкой входа в программу.

Точка входа — это метод в программе, с которого начинается выполнение кода.

В одной программе может быть только один метод **Main**.

Пример простой программы на C#:

```
1   using System;
2
3   namespace MyFirstProgram
4   {
5       class Program
6       {
7           static void Main(string[] args)
8           {
9               Console.WriteLine("Hello, World!");
10          }
11      }
12  }
```

using System;: Подключаем пространство имен System, которое содержит множество стандартных классов.

namespace MyFirstProgram: Создаем собственное пространство имен для организации нашего кода.

class Program: Определяем класс Program, который будет содержать точку входа в нашу программу.

static void Main(string[] args): Метод Main является точкой входа в программу. Он принимает массив строк args, который может использоваться для передачи аргументов командной строки.

Console.WriteLine("Hello, World!");: Выводит строку "Hello, World!" на консоль.

Домашнее задание:

1. Повторить материал лекции.
2. Учебное пособие. с.7

Материалы лекций:

https://github.com/ShViktor72/Education

Обратная связь:

colledge20education23@gmail.com

Список литературы:

1. Жаксыбаева Н.Н. Основы объектно-ориентированного программирования: язык C#. Часть 1./ Учебное пособие предназначено для учащихся технического и профессионального образования, Алматы, 2010,

2. https://learn.microsoft.com/ru-ru/dotnet/csharp/