

# Стиль программирования



# Утверждения по поводу стиля

1. Для любого языка существует один распространённый **стиль программирования**, или число таких стилей невелико.
2. Не существует и никогда не будет существовать стиля программирования, который обладал бы очевидными преимуществами по сравнению с любым из стандартных стилей.
3. В любом нетривиальном проекте желательно применять стандартный стиль программирования.
4. Каждый член команды должен писать в

```
// Пример простой программы на языке Си
// Стиль - для новичков
#include <stdio.h>
#include <math.h>

int main(int argc, _TCHAR* argv[])
{
    float x = 0;
    float y = 0;

    printf("Введите X: ");
    scanf("%f", &x);

    if (x < 0)
    {
        y = -1;
    }
    else if (x == 0)
    {
        y = 0;
    }
    else
    {
        y = sqrt(x);
    }
    printf("Результат: y = %.2f", y);
    getchar();

    return 0;
}
```

Tab  
или  
4 пробела

```
// Пример простой программы на языке Си
// Стиль - для новичков
#include <stdio.h>
#include <math.h>

int main(int argc, _TCHAR* argv[])
{
    float x = 0;
    float y = 0;

    printf("Введите X: ");
    scanf("%f", &x);

    if (x < 0)
    {
        y = -1;
    }
    else if (x == 0)
    {
        y = 0;
    }
    else
    {
        y = sqrt(x);
    }

    printf("Результат: y = %.2f", y);
    getchar();

    return 0;
}
```

```
// Пример простой программы на языке Си
// Стиль - для новичков
#include <stdio.h>
#include <math.h>

int main(int argc, _TCHAR* argv[])
{
    float x = 0;
    float y = 0;

    printf("Введите X: ");
    scanf("%f", &x);

    if (x < 0)
    {
        y = -1;
    }
    else if (x == 0)
    {
        y = 0;
    }
    else
    {
        y = sqrt(x);
    }

    printf("Результат: y = %.2f", y);
    getchar();

    return 0;
}
```

The diagram illustrates the flow of execution for the provided C code. A vertical blue line on the left side of the code acts as a starting point. From this line, several horizontal blue arrows point to the right, indicating the sequence of statements: the first arrow points to the first 'float' declaration, the second to the second 'float' declaration, the third to the 'printf' statement, the fourth to the 'scanf' statement, the fifth to the 'if' condition, the sixth to the 'y = -1' statement inside the first 'if' block, the seventh to the 'else if' condition, the eighth to the 'y = 0;' statement inside the second 'if' block, the ninth to the 'else' block, and the tenth to the 'y = sqrt(x);' statement inside the third 'if' block. Additionally, blue curly braces are placed to the right of the code to group the three conditional blocks, with arrows pointing from the 'if' and 'else if' lines to their respective closing braces.

```
// Пример простой программы на языке Си
// Стиль - для новичков
#include <stdio.h>
#include <math.h>

int main(int argc, _TCHAR* argv[])
{
    float x = 0;
    float y = 0;

    printf("Введите X: ");
    scanf("%f", &x);

    if (x < 0)
    {
        y = -1;
    }
    else if (x == 0)
    {
        y = 0;
    }
    else
    {
        y = sqrt(x);
    }

    printf("Результат: y = %.2f", y);
    getchar();

    return 0;
}
```

Вертикальный  
«пробел», используется  
для деления кода на  
логические блоки

```
// Пример простой программы на языке Си
// Стиль - эконом, логический блок одной строкой
#include <stdio.h>
#include <math.h>

int main()
{
    float x = 0, y = 0;

    // %f - float, %c - char, * == ignore
    printf("Введите X: "); scanf("%f%c", &x);

    y = (x < 0) ? -1 : sqrt(x);

    printf("Результат: y = %.2f", y); getchar();

    return 0;
}
```

# Пробелы

```
float x = 0;
```

```
scanf ("%f", &x);
```

```
if (x < 0)
```

```
{
```

```
    y = -1;
```

```
}
```

```
else if (0 == x)
```

```
{
```

```
    y = 0;
```

```
}
```

```
printf("Результат: при x = %d, y = %.2f", x, y);
```

Пробелы в  
условиях

Пробелы между  
if, for, while и  
скобкой:

Пробелы в  
операторах  
присваивания и  
математически  
выражениях:  
Пробелы между  
параметрами  
функций



```
//Пример "профи"-стиля
#include <stdio.h>
#include <math.h>

int main()
{
    float x = 0, y = 0;

    printf("Введите X: ");
    scanf("%f", &x);

    if      (x < 0)  y = -1;
    else if (x == 0) y = 0;
    else          y = sqrt(x);

    printf("Результат: y = %.2f", y);
    getchar();

    return 0;
}
```



# Аппаратное обеспечение (ака Hardware, железо)

Представляет собой фундамент (платформу) на котором строится **программное обеспечение**



Любое электронное или механическое устройство (**device**) как в сборке так и по частям можно называть аппаратным обеспечением

# Modern hardware types of computer

- Калькулятор
- **Персональный компьютер** (ПК, Personal Computer, PC)
  - **Настольный компьютер** (Desktop PC)
  - **Ноутбук** (Notebook, Лэптоп, Mobile PC)
    - **Нетбук** (Netbook)
  - **Игровая приставка** (Игровая консоль)
  - **Карманный компьютер** (КПК, PDA)
  - **Коммуникатор** (*Communicator, PDA Phone*)
  - **Смартфон** (Smartphone)
  - **Планшетный ПК** (Tablet PC)
- **Рабочая станция** (workstation)
- **Сервер** (server)
- **Суперкомпьютер**

## Архитектура фон Неймана

1. Принцип **двоичности** - для представления **данных** и **команд** используется **двоичная система счисления**.
2. Принцип **последовательного программного управления** - **программа** состоит из набора **команд**, которые **выполняются процессором** последовательно, одна после завершения другой.
3. Принцип **однородности памяти** - Как **программы (команды)**, так и **данные** хранятся в одной и той же **памяти**. Над **командами** можно **выполнять** такие же **действия**, как и над **данными**.
4. Принцип **адресуемости памяти** - структурно **основная память** состоит из пронумерованных **ячеек**; **процессору** в произвольный момент времени **доступна** любая **ячейка**.
5. Принцип **условного перехода** - возможно присутствие в **программе команд**, которые изменяют последовательность **выполнения программы**

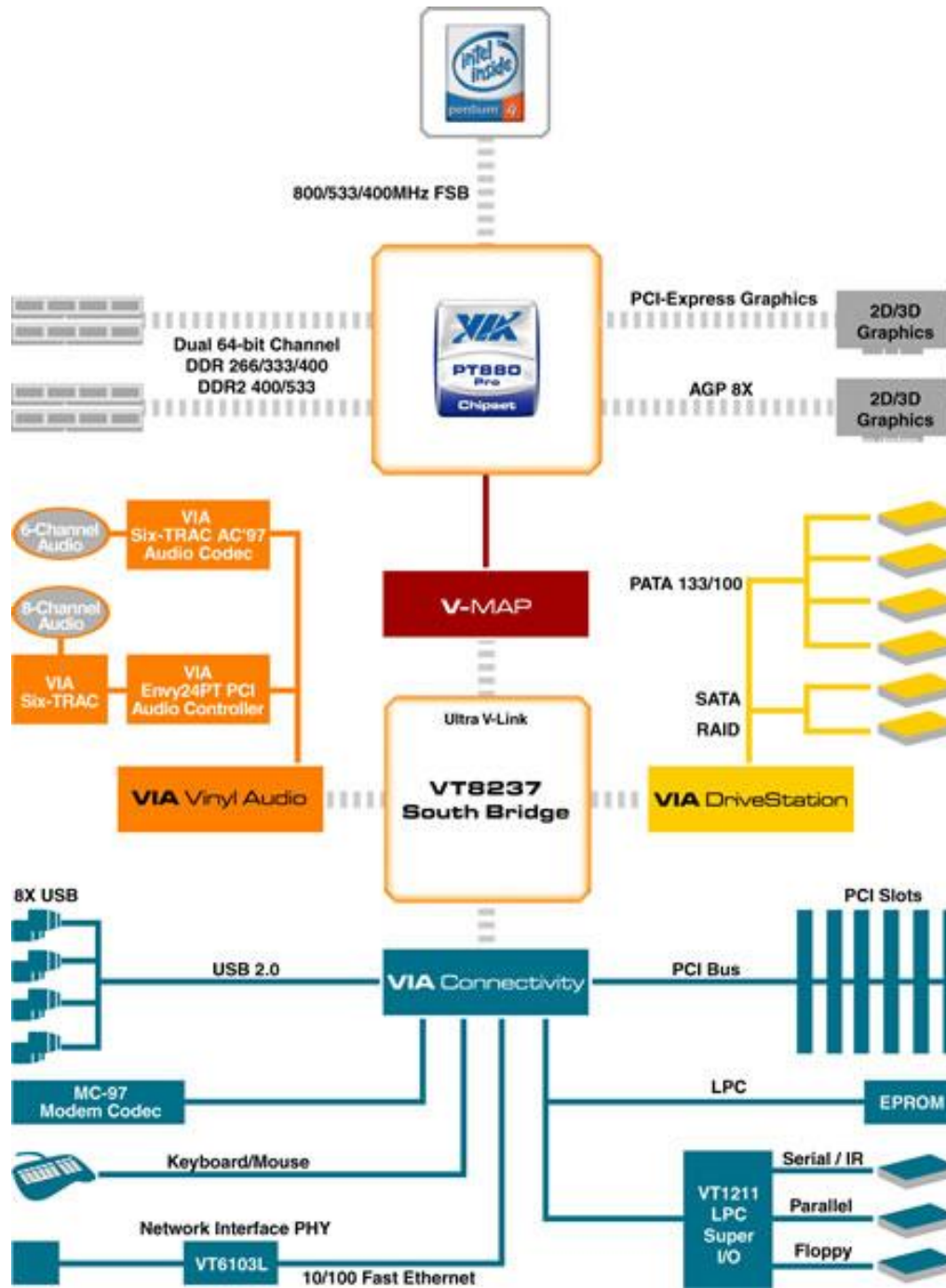
# Главные компоненты компьютера

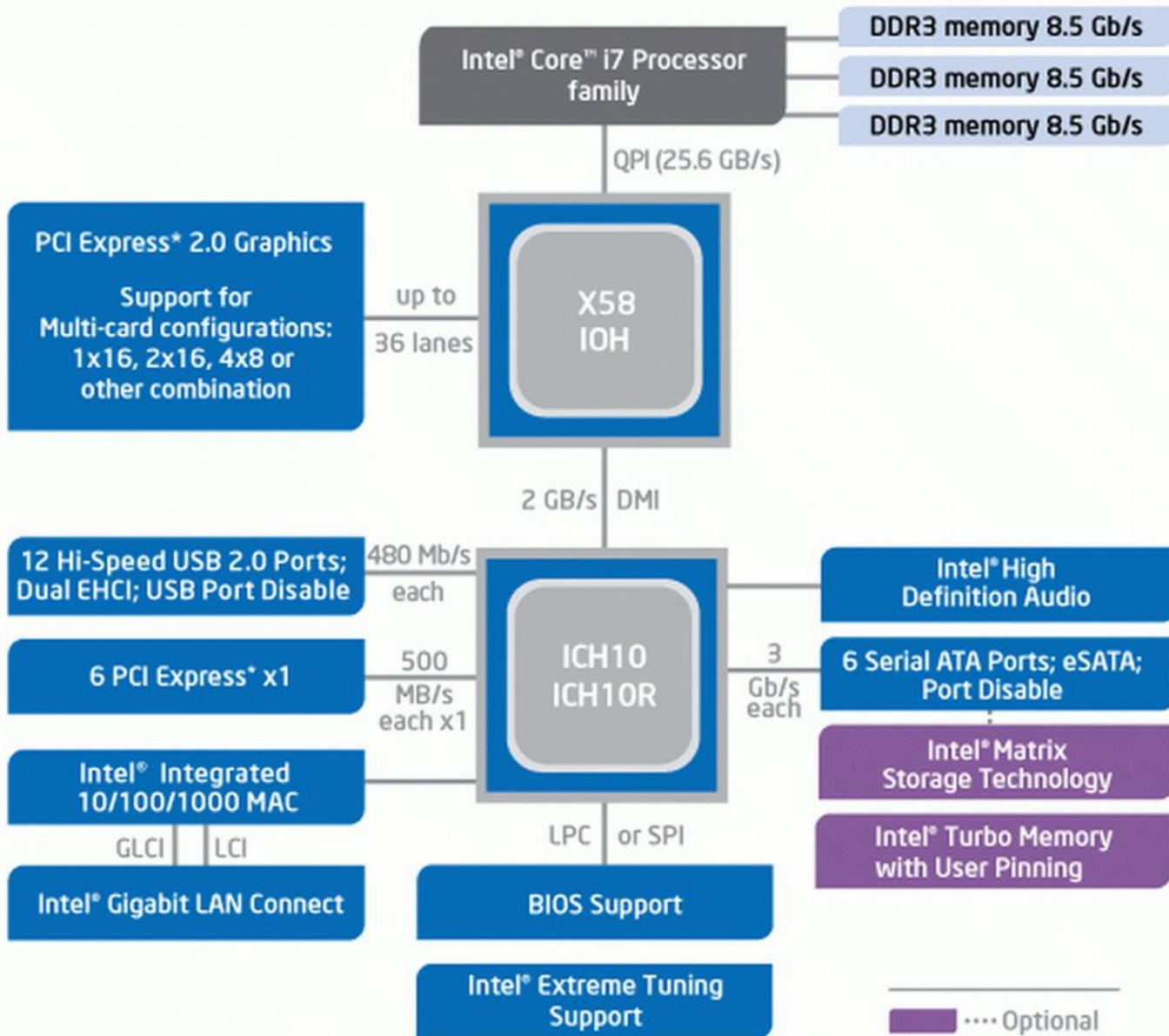
**Процессор (*Central Processing Unit*)** – выполняет команды, производит вычисления

**Оперативная память (*Random Access Memory*)** предназначена для хранения программных команд и результатов вычислений.

**Устройством ввода (*Input Device*)** называется любое устройство, позволяющее человеку передать информацию в компьютер.

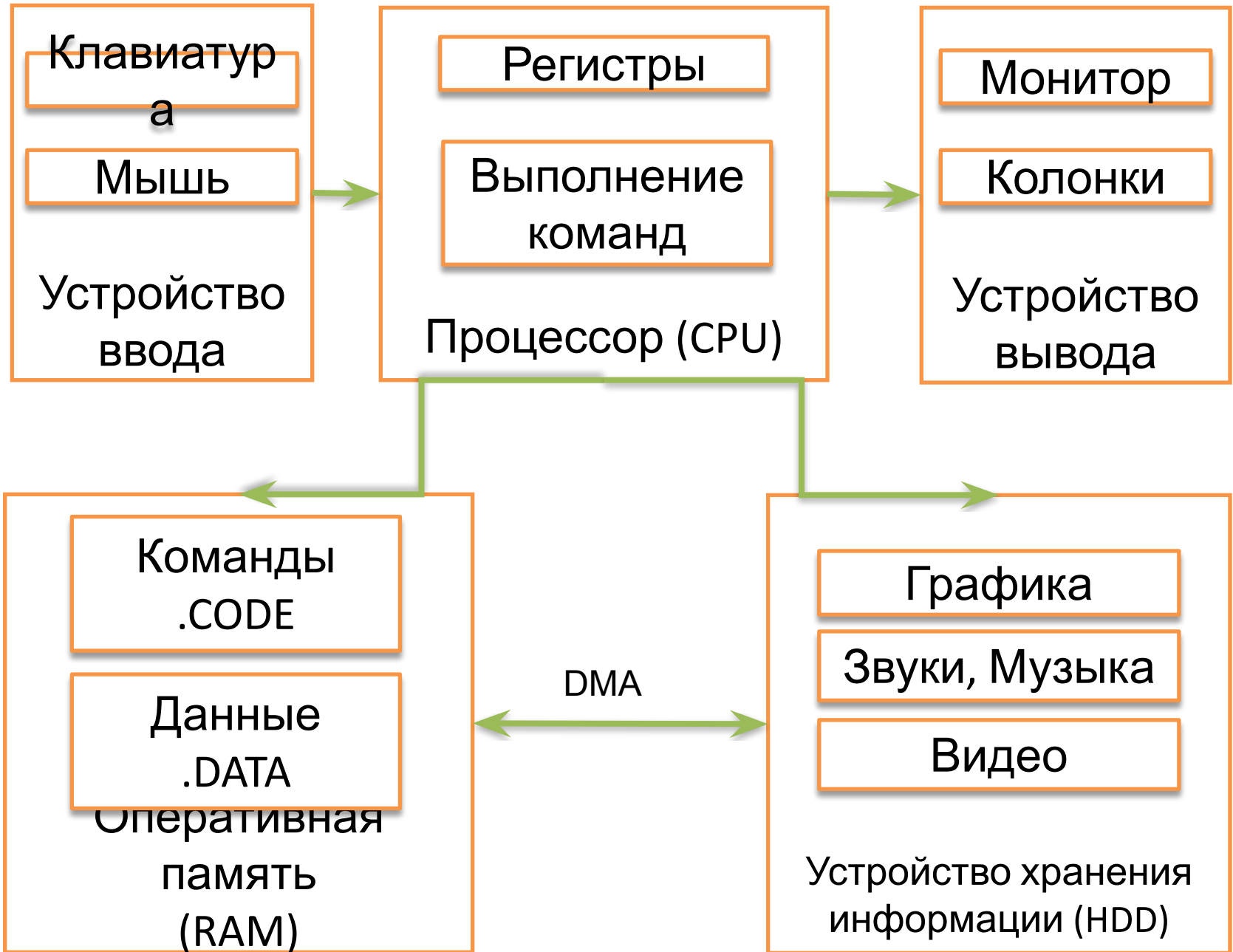
**Устройством вывода (*Output Device*)** называется любое устройство, с помощью которого компьютер выдаёт информацию человеку.







# Упрощённая Архитектура компьютера



**Закон Мура - число транзисторов на кристалле  
будет удваиваться каждые 24 месяца**

## Moore's Law - 2005

Transistors  
Per Die

$10^{10}$

$10^9$

$10^8$

$10^7$

$10^6$

$10^5$

$10^4$

$10^3$

$10^2$

$10^1$

$10^0$

1960

1965

1970

1975

1980

1985

1990

1995

2000

2005

2010

1K

4K

8008

4004

8080

8086

80286

386™ Processor

486™ Processor

Pentium® Processor

Pentium® II Processor

Pentium® III Processor

Pentium® 4 Processor

Itanium™ Processor

Itanium™ 2 Processor

16K

64K

256K

1M

4M

16M

64M

128M

256M

512M

1G

2G

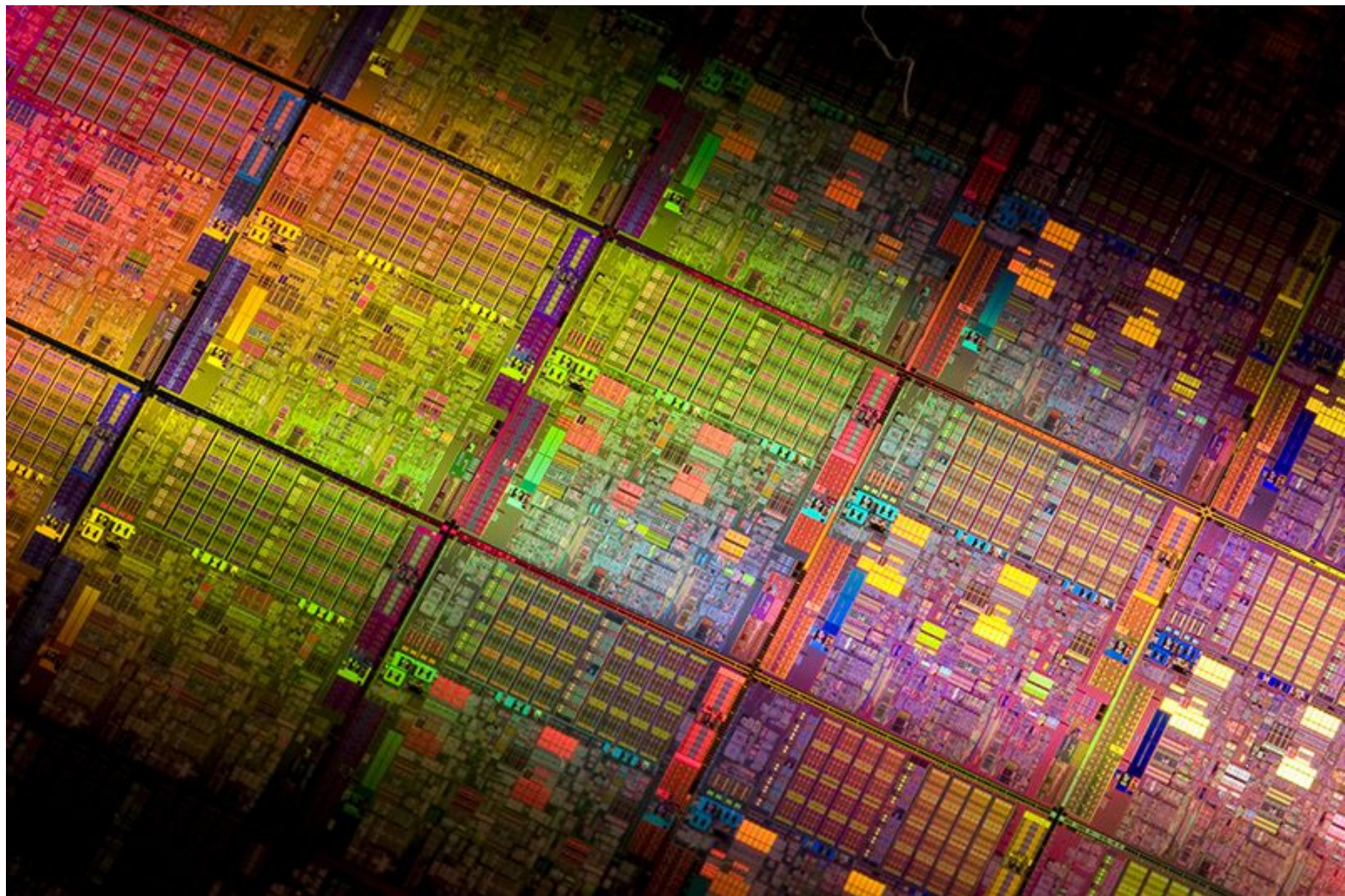
◆ 1965 Data (Moore)

■ Memory

◇ Microprocessor

Source: Intel





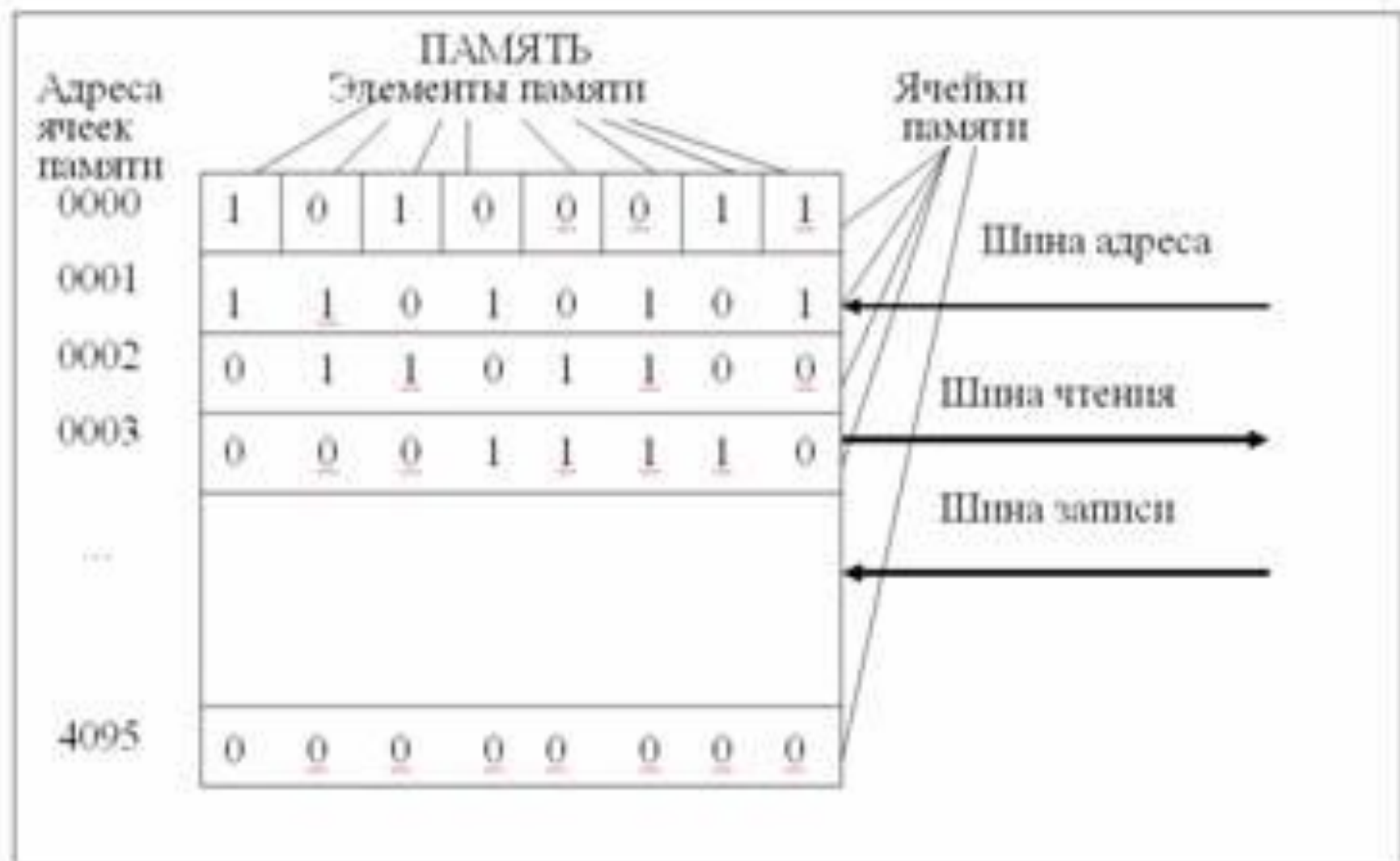


Рис 4.2 Организация оперативной памяти ЭВМ.



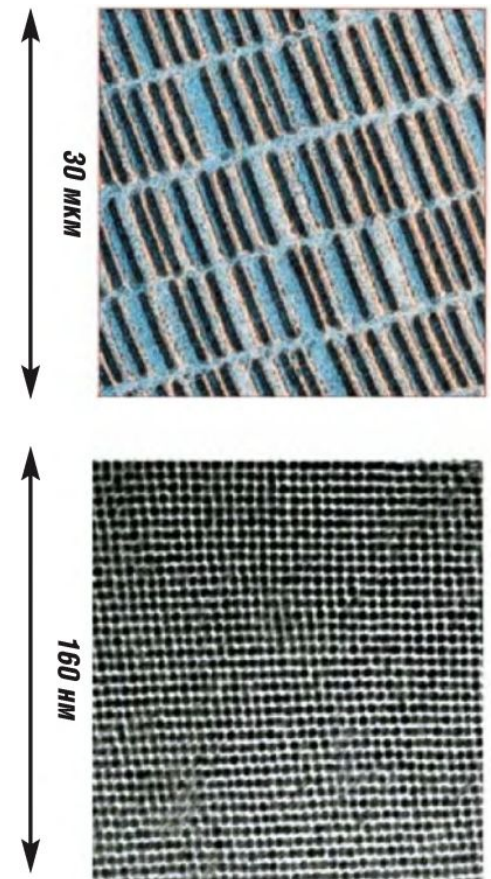
# DEKSI HARD Disk MANAGER

## PROFESSIONAL

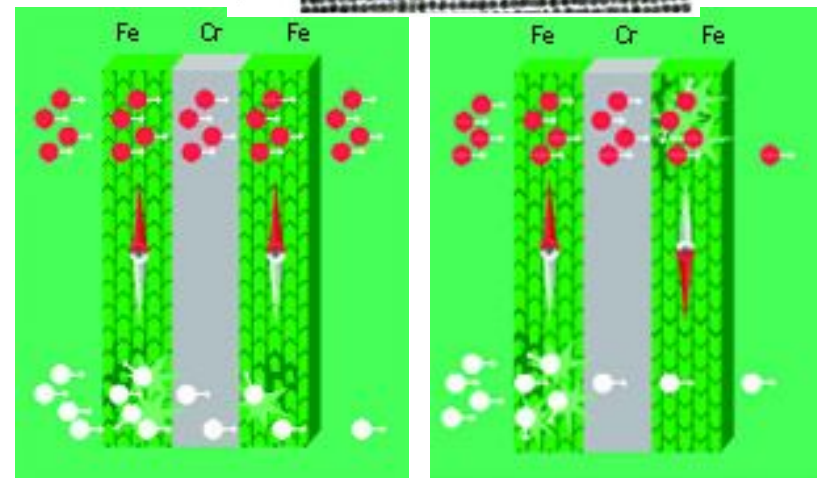
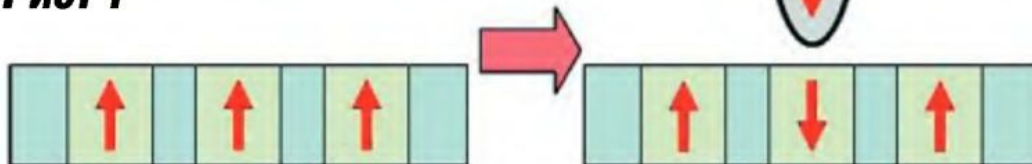


v2.90  
(3586)

**DEK**  
Software International

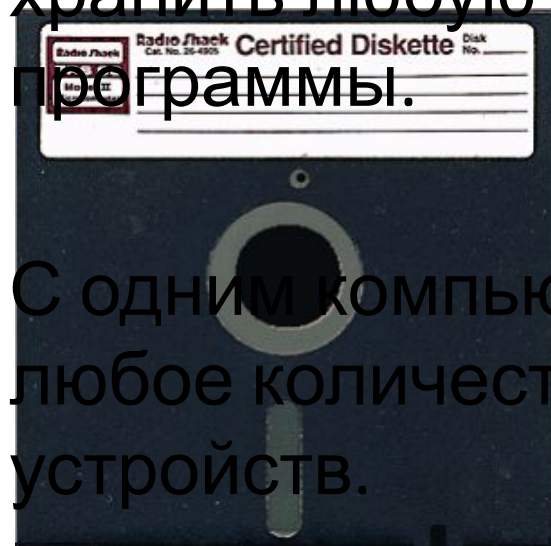


**Рис. 1**



# Устройства хранения информации

«**Внешняя память**» – это память, которая используется для постоянного хранения данных по окончании или до начала работы компьютера. Во внешнем запоминающем устройстве данные хранятся блоками, именуемыми **файлами**. Файлы могут быть произвольного размера и хранить любую информацию, в том числе и программы.



С одним компьютером может быть соединено любое количество внешних запоминающих устройств.