Информатика

Аппаратные средства вычислительной техники и ассемблеры

Часть 1

Гирик Алексей Валерьевич

Университет ИТМО
2022

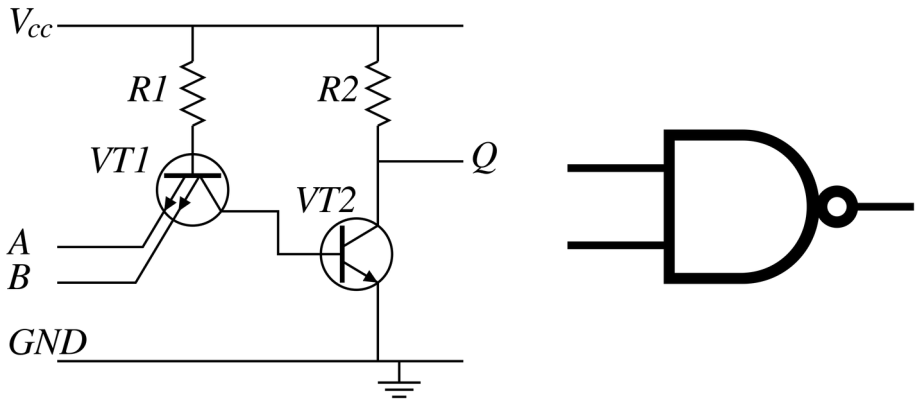
Материалы курса

- Презентации, материалы к лекциям, литература, задания на лабораторные работы
  - shorturl.at/jqRZ6

Процессоры

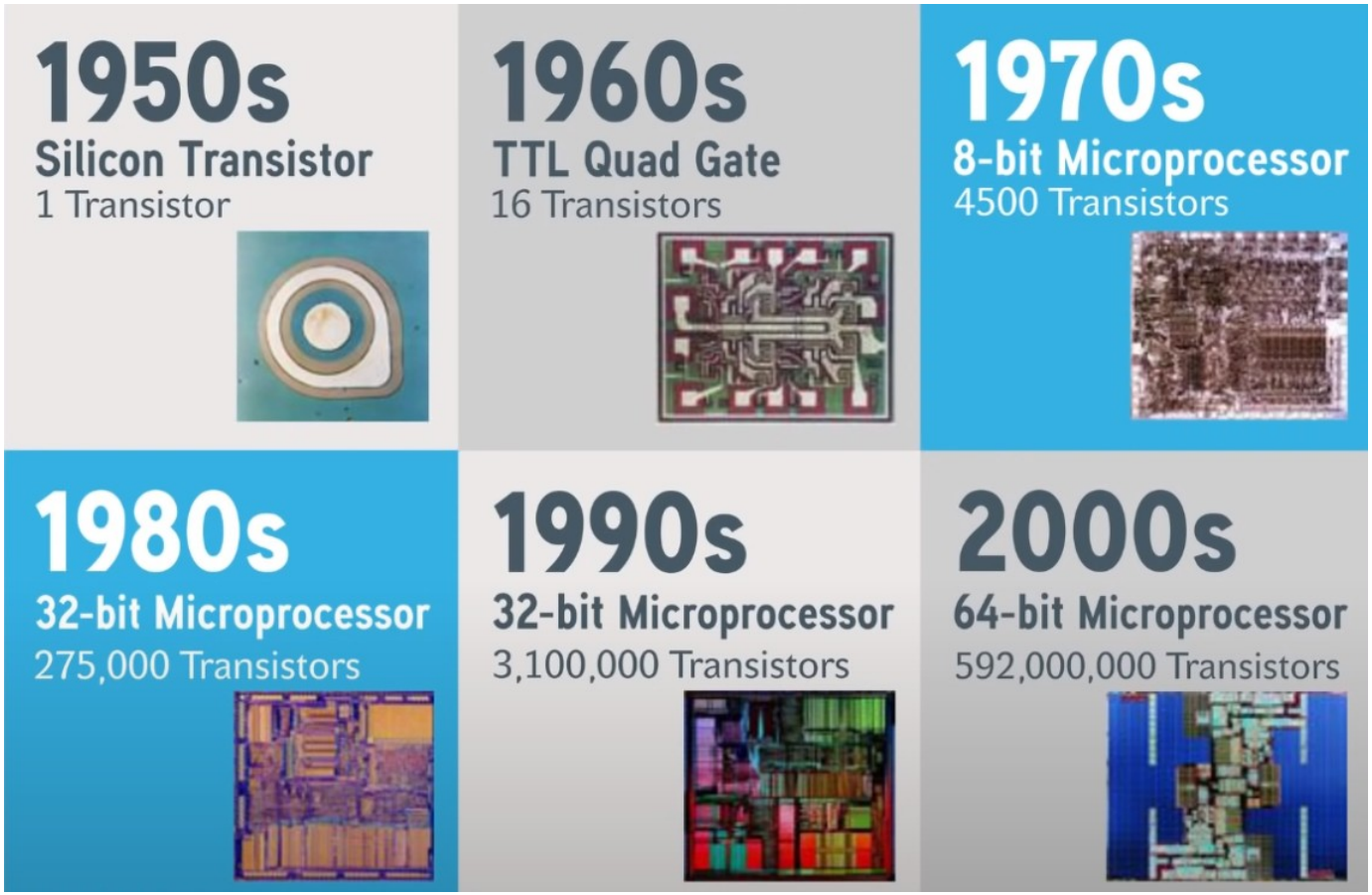
Что собой представляет процессор (с точки зрения схемотехники)?

- если утрировать, то
  - любой процессор – просто огромная *логическая схема*, составленная из *логических элементов* (сумматоров, компараторов, триггеров, регистров)
  - логические элементы в свою очередь построены на основе транзисторов, инверторов, диодов, ...



Элемент И-НЕ

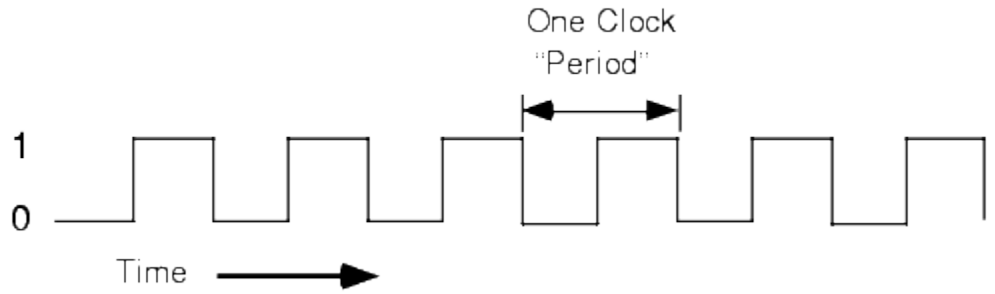
More, please?



"Закон Мура мертв"
https://www.youtube.com/watch?v=MTROT945xuo

Как работает процессор?

- если упрощенно, то
  - каждый раз, когда на вход поступает *тактирующий импульс*, в процессоре переключается состояние логических элементов
  - частота тактирующих импульсов (*тактовая частота*) определяет, как много переключений – и в конечном счете операций – в секунду может делать процессор

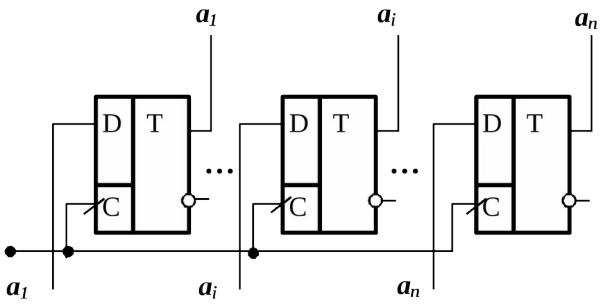


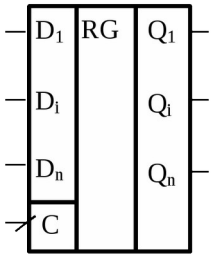
Что собой представляет процессор?

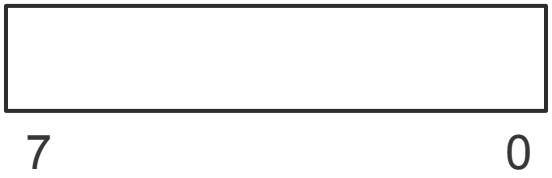
- если утрировать, то
  - любой процессор – просто "числодробилка" (number cruncher)
  - процессор хранит числа, с которыми выполняет операции, в *регистрах*

Что собой представляет регистр?

- если утрировать, то
  - регистр – ячейка памяти внутри процессора
  - наиболее часто на программном уровне приходится иметь дело с
    - регистрами общего назначения (РОНами)
    - регистром команд
    - регистром флагов

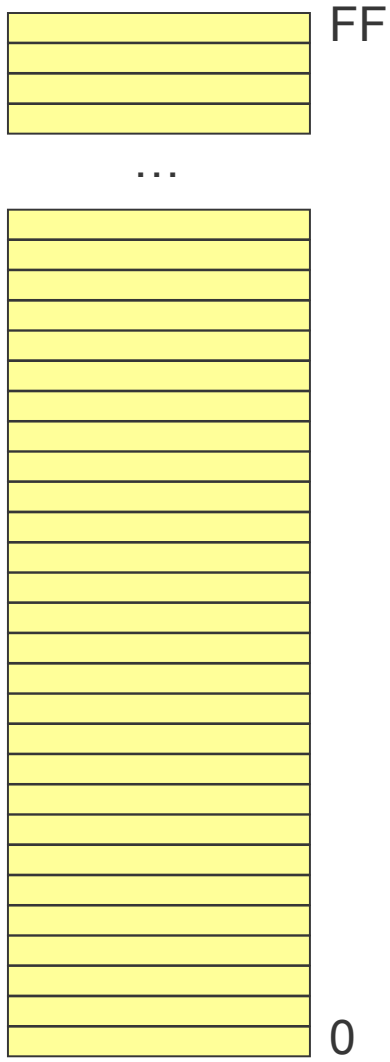






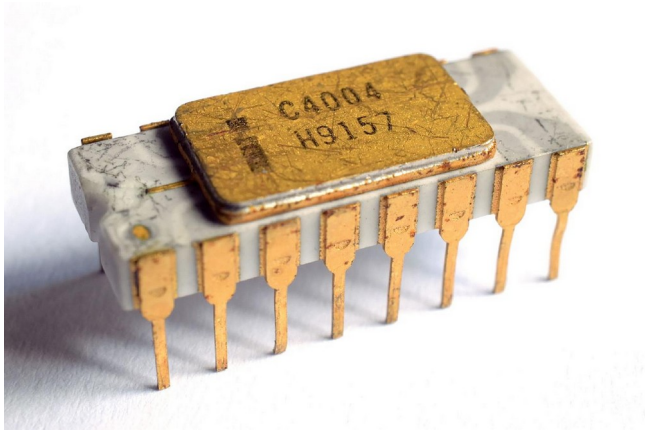
Что собой представляет память?

- если говорить упрощенно, то
  - память – это массив ячеек (обычно байтов), каждый байт имеет порядковый номер, который называется адресом
  - исполнение любой программы сводится к изменению состояния памяти от некоторого начального к некоторому конечному



Разрядность

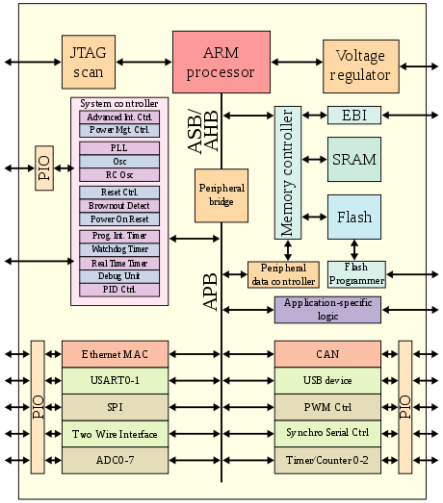
- разрядность процессора определяет
  - размер **машинного слова**
    - числа какой разрядности могут участвовать в основных арифметических/логических/... операциях
  - размер **регистров общего назначения**
  - разрядность шины данных
    - но это не всегда равная разрядности процессора величина, гораздо чаще кратная (x2, x4 и т.д.)
  - объем непосредственно адресуемой памяти

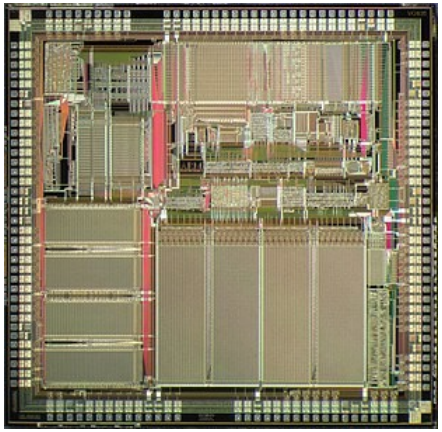


Intel 4004, 1970

Система команд и архитектуры

- **система команд** процессора определяет операции, которые можно выполнять над данными и с помощью которых можно управлять поведением процессора
- **архитектура** процессора – набор свойств и качеств процессора, определяющий разрядность, систему команд и другие свойства
- **микроархитектура** процессора – реализация архитектуры с использованием конкретного техпроцесса и микроархитектурных решений
  - https://www.ixbt.com/cpu/cpu-microarchitecture-part-1.shtml

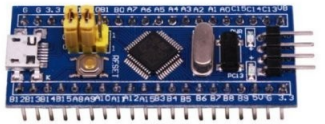




Система команд и архитектуры

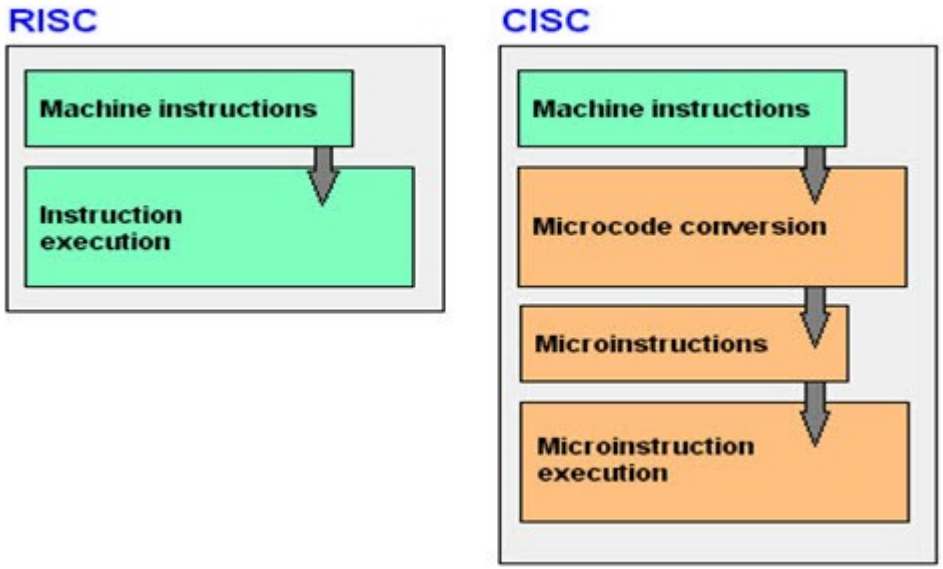
- **CISC**
  - Complex Instruction Set Computer
    - IBM System 360, 370, z/Arcitecture
    - DEC PDP-11, VAX, ...
    - Intel x86, **x86-64**
- **RISC**
  - Reduced Instruction Set Computer
    - 88000
    - PowerPC
    - SPARC
    - MIPS
    - **ARM**
    - **RISC-V**
- ...
- **OISC**
  - One Instruction Set Computer

DEC PDP-11, 1970



STM32F103C8T6
Development Board, 2019

Сравнение CISC и RISC



RISC VERSUS CISC

RISC	CISC
An instruction set architecture that is designed to perform a smaller number of computer instructions so that it can operate at a higher speed	A full set of computer instructions that intends to provide the necessary capabilities in an efficient way
Stands for Reduced Instruction Set Computer	Stands for Complex Instruction Set Computer
Utilizes a small, highly optimized set of instructions	Utilizes a large, specialized and a complex set of instructions
More machine oriented	More programmer oriented
Simple and requires one clock cycle to execute instructions	Complex and requires multiple clock cycles to execute an instruction
More registers	Fewer registers
Instructions have simple, fixed formats with few addressing modes	Instructions have variable formats with several complex addressing modes
Has simple instructions - the program length is long	Has complex instructions - the program length is short
Requires more RAM	Requires a minimum amount of RAM
Used in Hardwired Control Unit; used in applications such as mobile phones and tablets	Used in Microprogrammed Control Unit; used in applications such as desktop computer and laptops

Visit www.PEDIAA.com

Как оценить производительность процессора?

$$\frac{T}{P} =$$

Как оценить производительность процессора?

- Пусть есть некоторый процессор, который выполняет программы, состоящие из инструкций, причем каждая инструкция выполняется за некоторое количество тактов (циклов)

- Введем обозначения:
  - P – общее количество выполненных программ
  - I – общее количество инструкций в программах
  - T – общее время выполнения программ
  - C – общее количество циклов, затраченное на выполнение программ

Очень сложная и непонятная формула

коэффициент производительности процессора

?

$$\frac{T}{P} = \frac{T}{P} \cdot 1 = \frac{T}{P} \cdot \frac{I}{C} \cdot \frac{C}{I} = \frac{T}{C} \cdot \frac{I}{P} \cdot \frac{C}{I}$$

? ?

Кому вообще нужны эти формулы?..

коэффициент производительности процессора

время одного такта (можно уменьшить, подняв частоту)

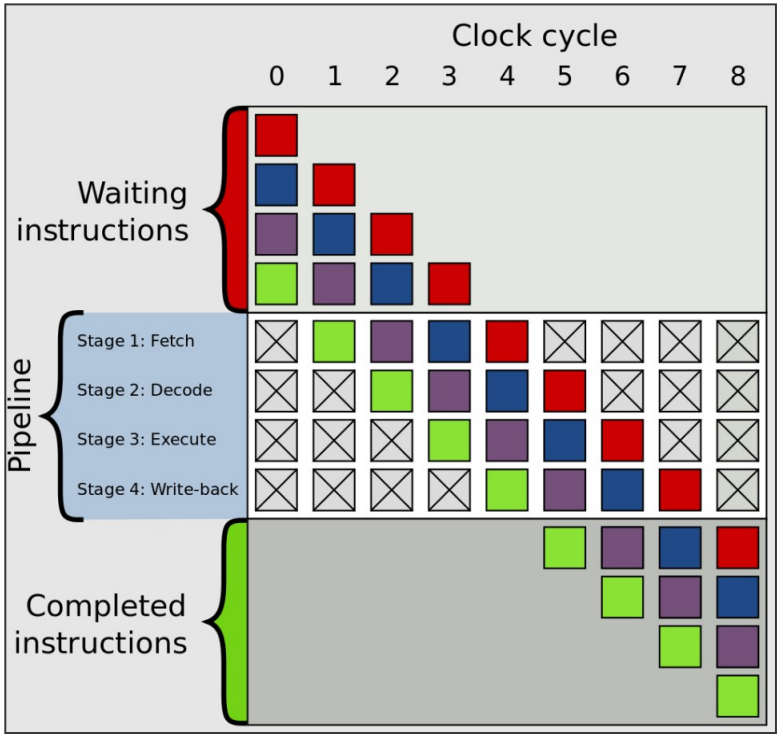
$$\frac{T}{P} = \frac{T}{P} \cdot 1 = \frac{T}{P} \cdot \frac{I}{C} \cdot \frac{C}{I} = \frac{T}{C} \cdot \frac{I}{P} \cdot \frac{C}{I}$$

среднее количество инструкций в программе (можно уменьшить, добавляя в систему команд "сложные" команды)

среднее количество тактов на одну команду (можно уменьшить, упрощая команды)

Исполнение команд

- В современных процессорах применяются разнообразные архитектурные решения для увеличения производительности:
  - конвейеризация (instruction pipelining)
  - предвыборка кода (prefetching)
  - предсказание ветвлений (branch prediction)
  - суперскалярность (superscalar architecture)

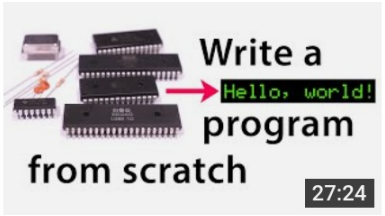


Конвейер из четырех стадий

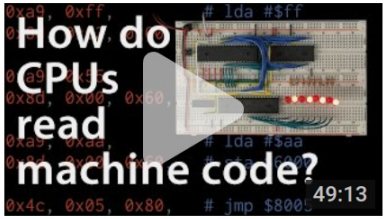
Пример работы с 8-разрядным процессором

- "Hello, world" на 8-разрядном микропроцессоре MOS 6502

- Ben Eater
  - https://eater.net/6502
    - https://www.youtube.com/watch?v=LnzuMJLZRdU
    - https://www.youtube.com/watch?v=yl8vPW5hydQ



"Hello, world" from scratch on a 6502 — Part 1



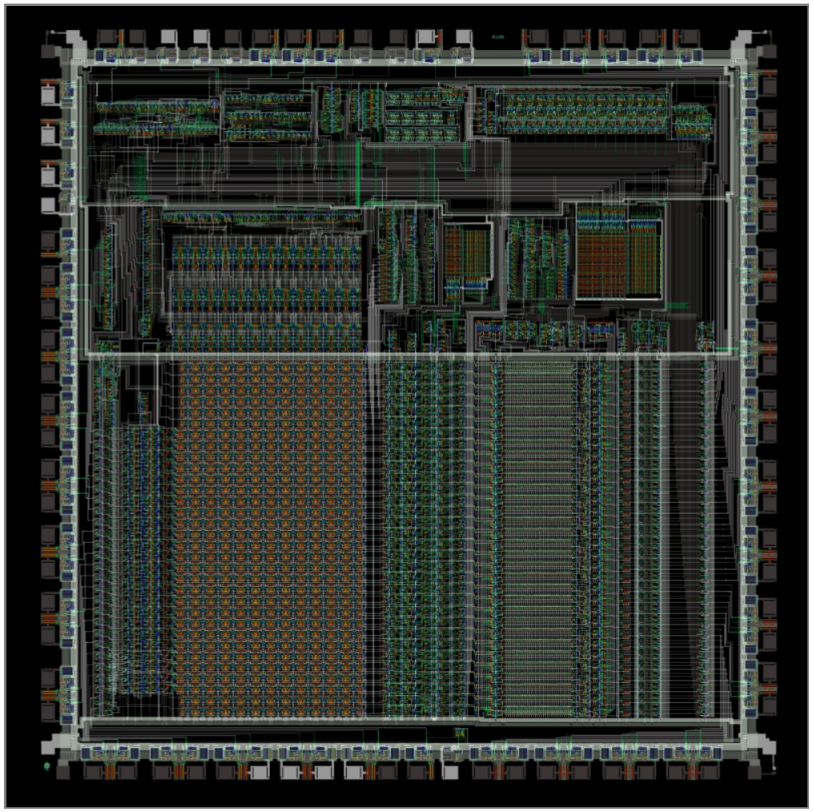
How do CPUs read machine code? — 6502 part 2

Онлайн-симулятор MOS 6502

- http://visual6502.org/JSSim/index.html

Онлайн-симулятор ARM1

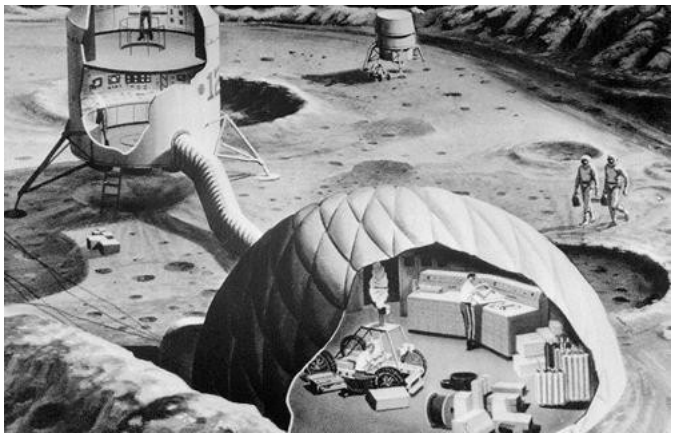
- Внутри ARM1
  - https://www.youtube.com/watch?v=rXW-CLByNDs

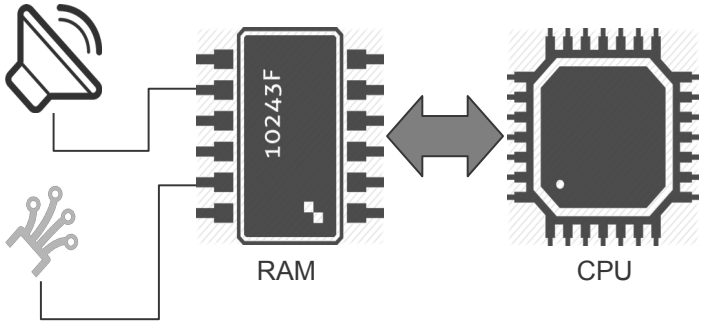


Пример проектирования архитектуры процессора

Очень простая задача

- Необходимо создать систему контроля содержания кислорода в воздухе
  - нормальное содержание по объему – 21%
  - датчик записывает в заданный адрес памяти текущее значение
  - есть возможность ввести с пульта
    - нормальное значение
    - максимально допустимое отклонение
  - сигнал тревоги включается путем записи ненулевого значения в заданную ячейку памяти



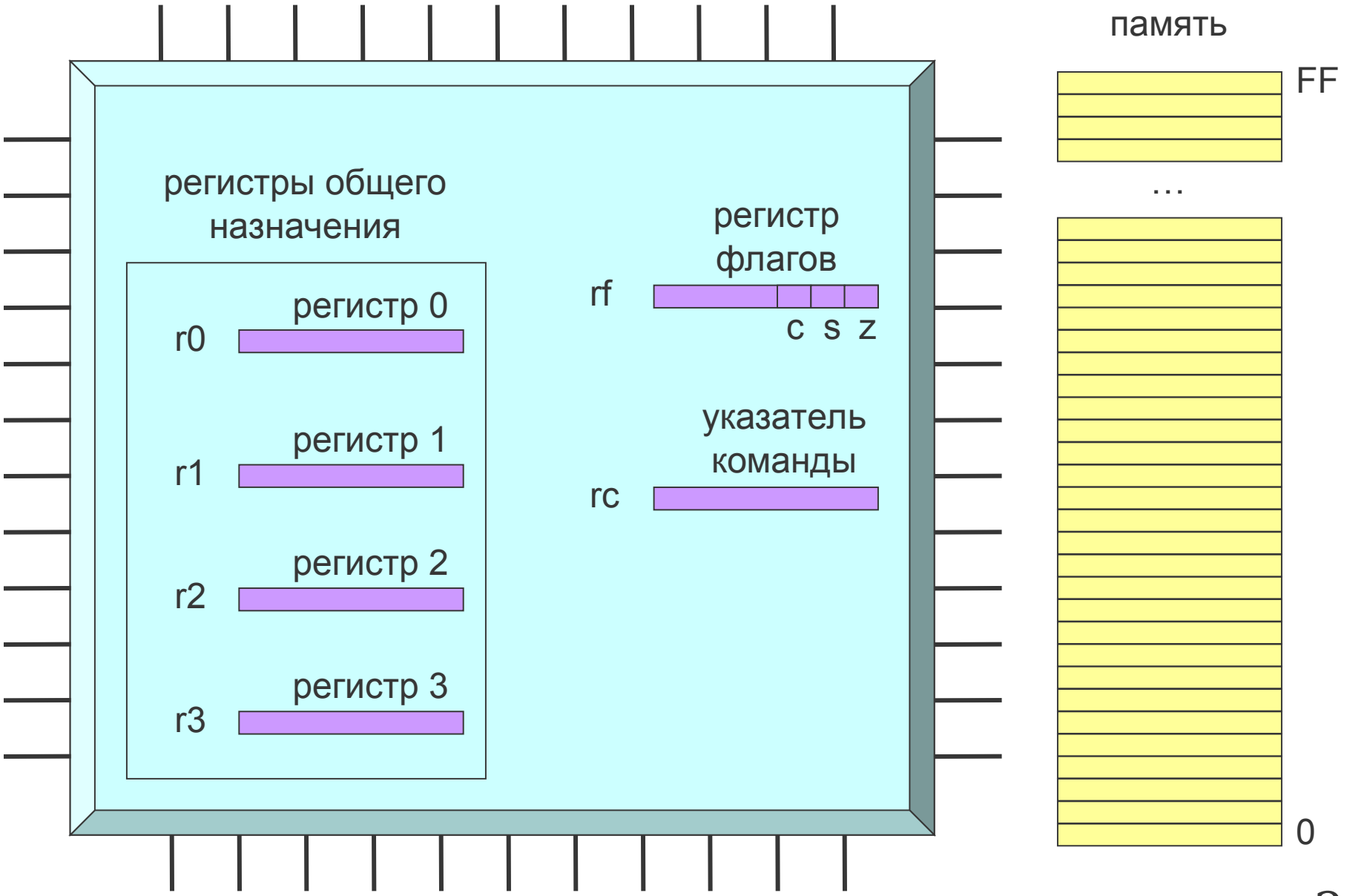


условная схема системы контроля содержания кислорода

Очень простой процессор

- разрядность
- объем непосредственно адресуемой памяти
- количество регистров и их назначение
- значения регистров после включения питания
- стартовый адрес
- система команд
  - коды операций
  - типы адресации
- типы и представление чисел
  - разрядность
  - знак

Очень простой процессор



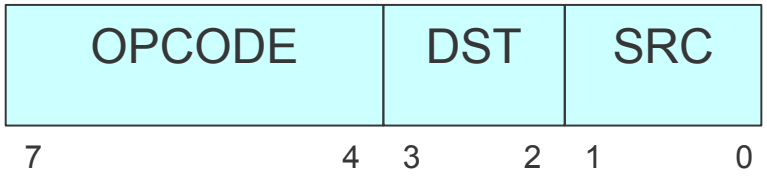
Очень простая система команд

- какие операции поддерживаются?
  - пересылки данных
  - арифметические
  - передачи управления
  - еще какие-то?

- сколько нужно битов для хранения кода операции?
- какая дополнительная информация должна быть в каждой команде?

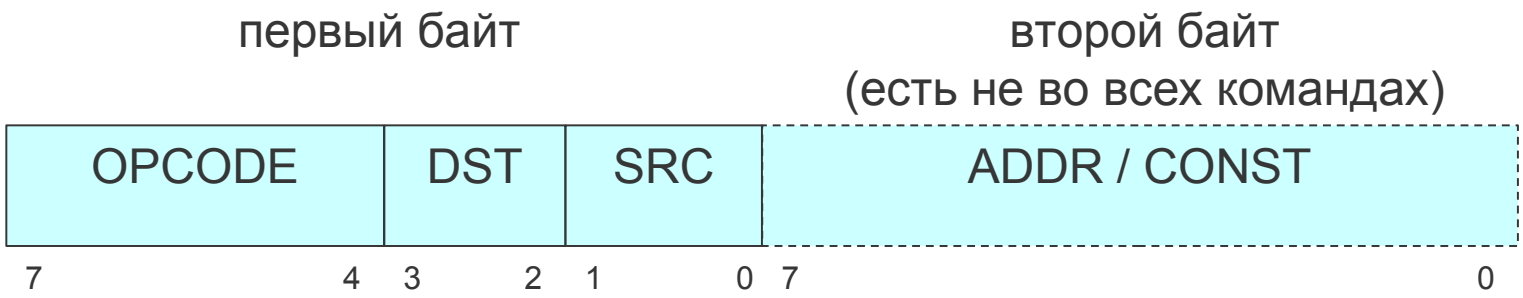
Очень простая система команд

- структура команды



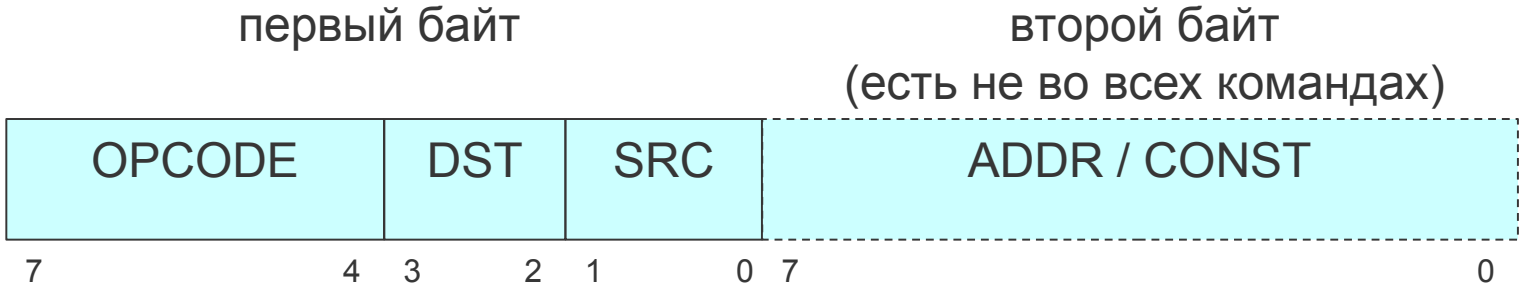
Очень простая система команд

- структура команды



Команды пересылки данных

- структура команды



- из регистра в регистр

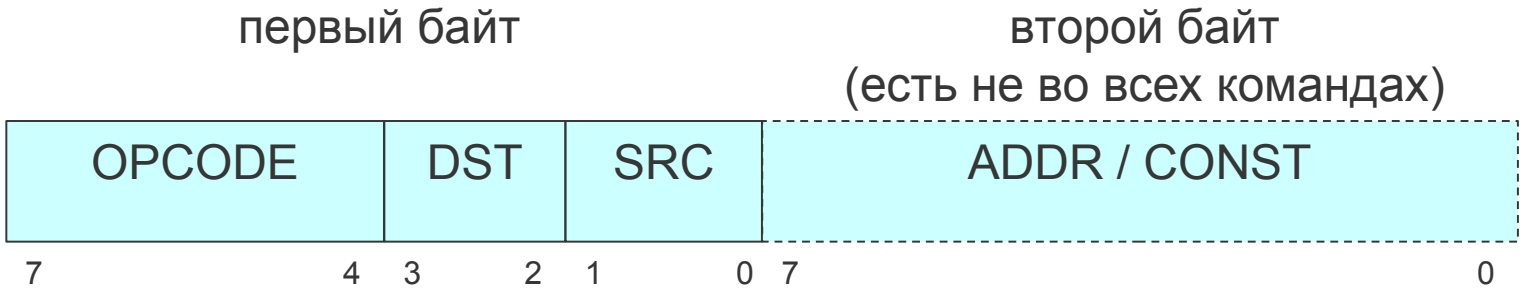
```
0000    ..  ..          -
```

например,

```
00000001 0x01     из R1 в R0
00001110 0x07     из R2 в R3
```

Команды пересылки данных

- структура команды



- из памяти в регистр

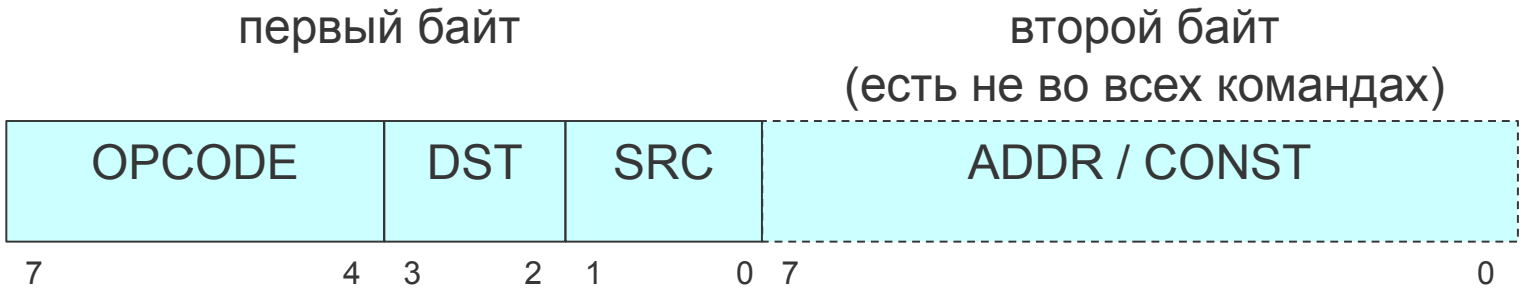
```
0001    ..   ??        ........
```

например,

```
00010000 00001010   0x0A10   из 0A в R0
00011100 11111111   0xFF1C   из FF в R3
```

Команды пересылки данных

- структура команды



- из регистра в память

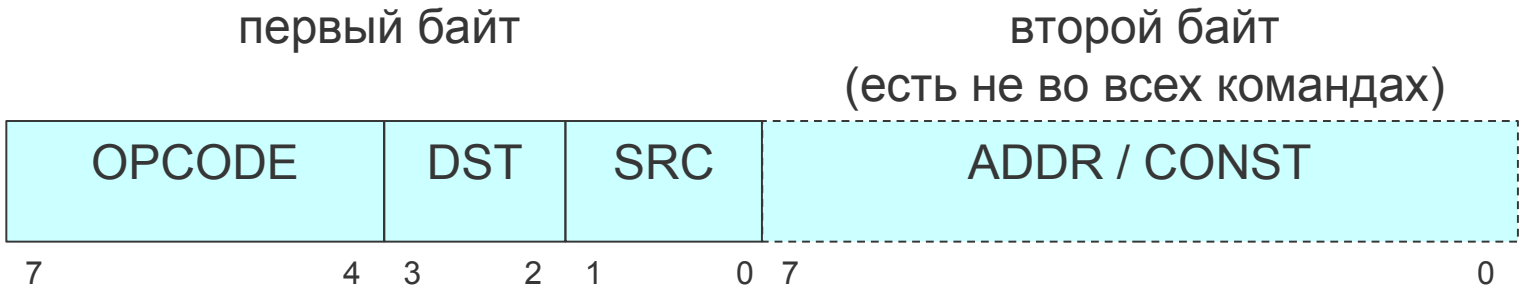
```
0010    ??  ..        ........
```

например,

```
00100001 00001010   0x0A21   из R0 в 0A
00100011 11111111   0xFF23   из R3 в FF
```

Команды пересылки данных

- структура команды



- число в регистр

```
0011    ..   ??        ........
```

например,

```
00111000 10000000   0x8038   0x80 в R2
00110100 11111110   0xFE34   0xFE в R1
```

От программирования в двоичных кодах к мнемонической записи

```
00110000 00000010 00000100 00100001 11111111
```

Сколько здесь команд?

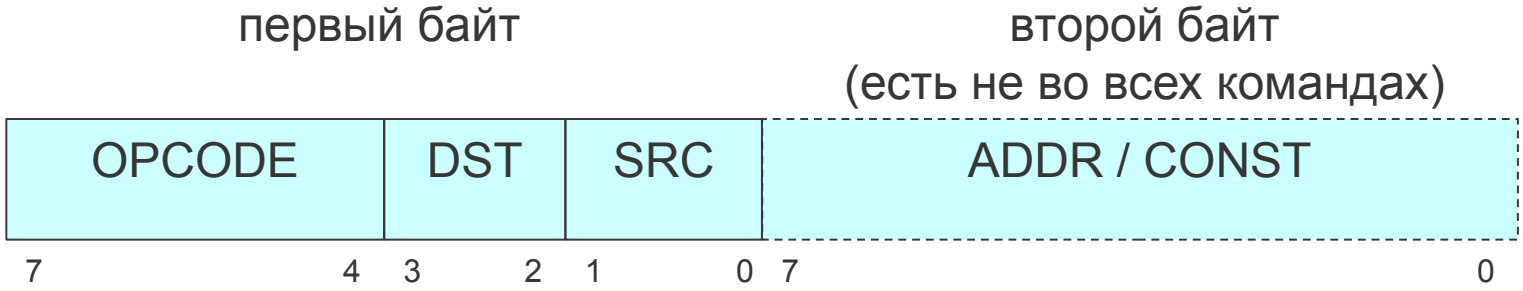
Как записать эту программу понятнее (для человека)?

От программирования в двоичных кодах к мнемонической записи

машинный код (битовое представление)	**мнемоническая запись (язык ассемблера)**
00110000 00000010	R0 <- 2
00000100	R1 <- R0
00100001 11111111	@0xFF <- R1

Арифметические команды

- структура команды



- сложение

```
0100    ..  ..      -
```

- вычитание

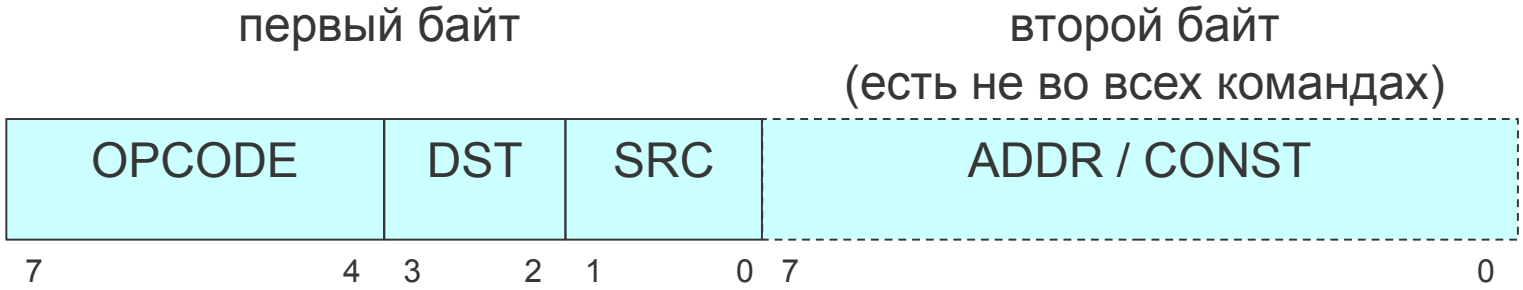
```
0101    ..  ..      -
```

например,

```
01010001 0x51      R0 <- R0 - R1
```

Арифметические команды

- структура команды



- умножение

```
0110    ..  ..          -
```

- деление

```
0111    ..  ..          -
```

например,

```
01110101 0x75       R1 <- R1 / R1
```

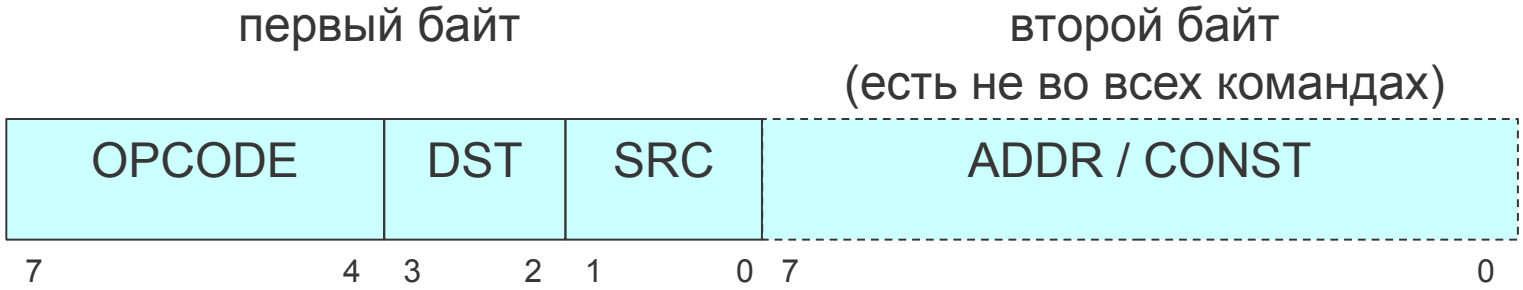
Последствия выполнения команд

```
00110000 00000010        R0 <- 2
00000100                 R1 <- R0
01000100                 R1 <- R1 + R0
00100001 11111111        @0xFF <- R1
```

- команды пересылки данных не изменяют содержимое регистра флагов, а арифметические – изменяют

Команда сравнения

- структура команды



- сравнить два регистра

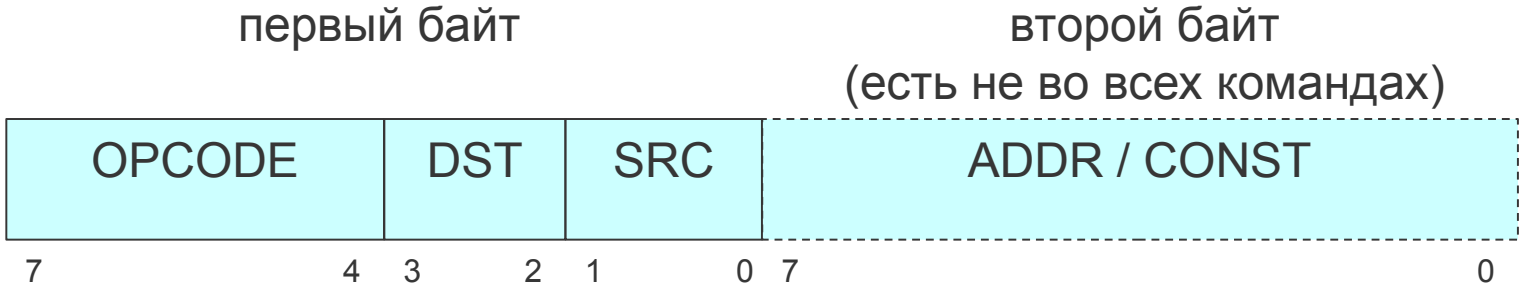
```
1000    ..  ..          -
```

сравнение реализуется как вычитание без записи результата, но с установкой флагов в регистре флагов, например,

```
10000001    0x81        RF <- R0 ~ R1
```

Команды перехода

- структура команды



- перейти, если есть перенос (RF.C = 1)

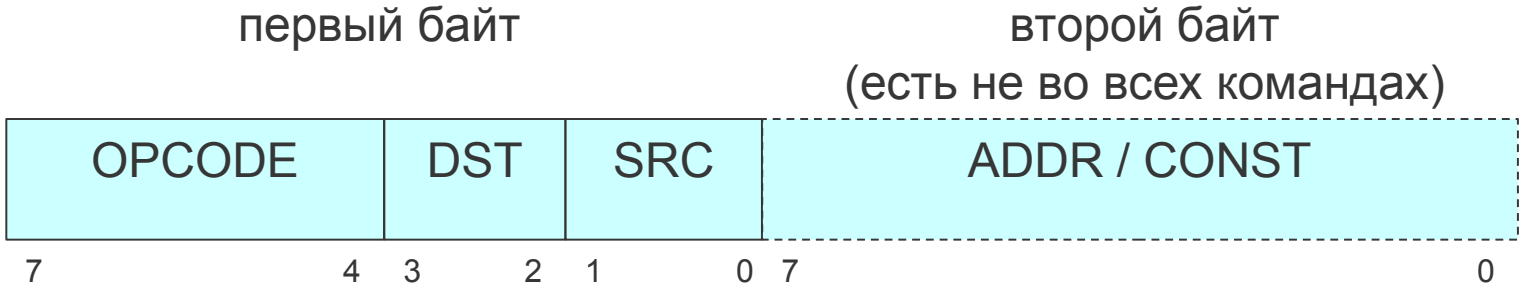
```
1001    ??  ??        ........
```

например,

```
10010000 10000000  0x8090  RC <- @0x80 (C)
```

Команды перехода

- структура команды



- перейти, если знак (RF.S = 1)

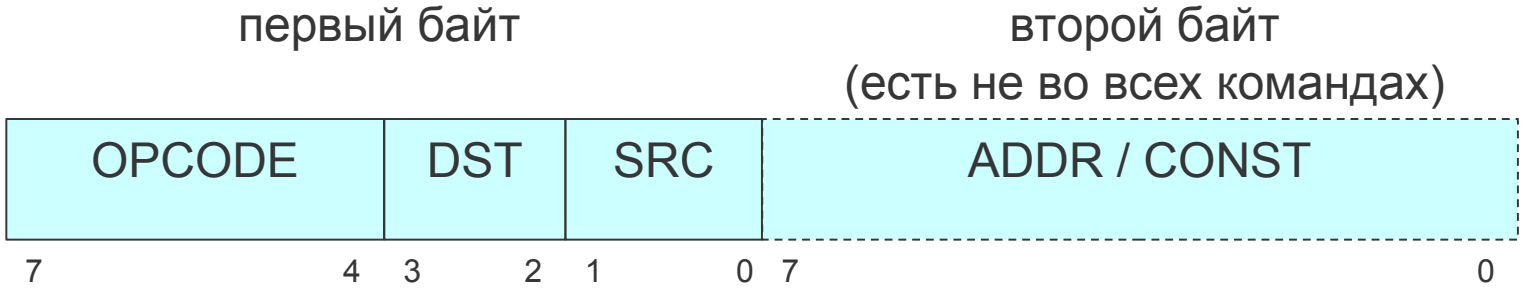
```
1010    ??  ??        ........
```

например,

```
10100000 10000000  0x80A0  RC <- @0x80 (S)
```

Команды перехода

- структура команды



- перейти, если ноль (RF.Z = 1)

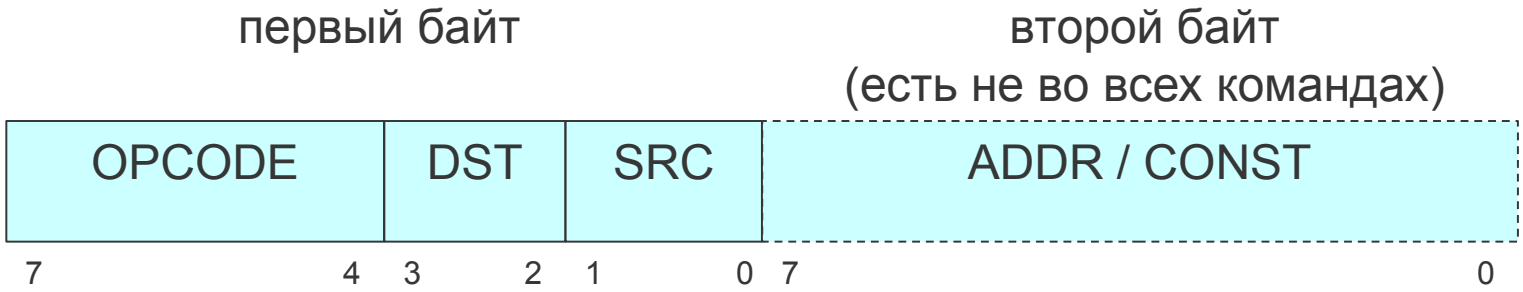
```
1011    ??  ??        ........
```

например,

```
10110000 10000000  0x80B0  RC <- @0x80 (Z)
```

Безусловный переход

- структура команды



- перейти

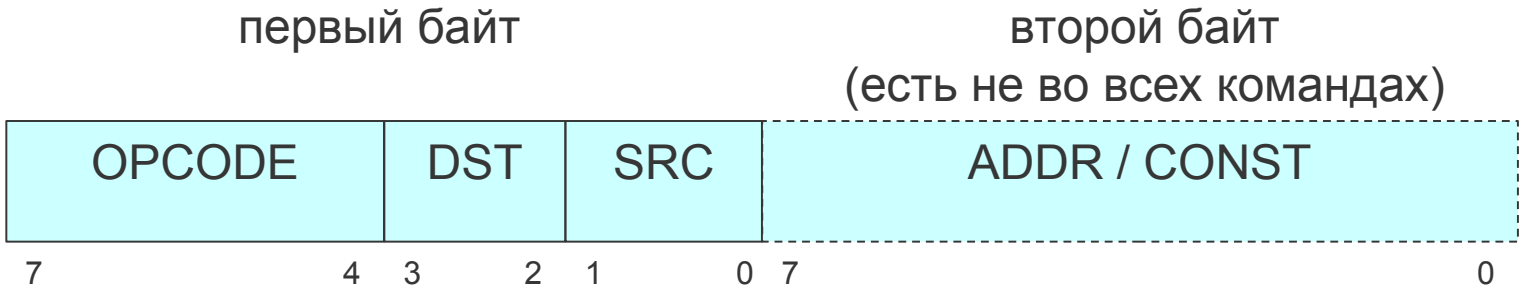
```
1100    ??  ??        ........
```

например,

```
11000000 10000000  0x80C0  RC <- @0x80
```

Дополнительные команды

- структура команды



- случайное число в регистр

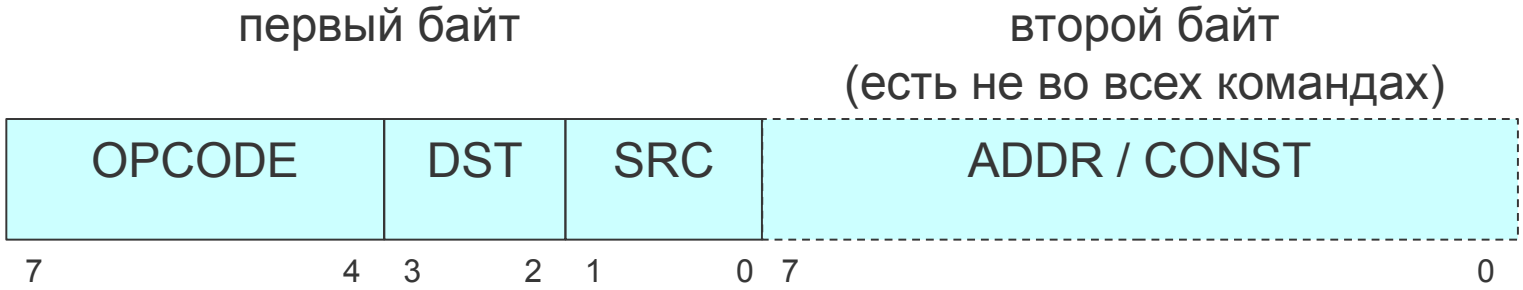
```
1101    ..  ??          -
```

например,

```
11010100 0xD4      R1 <- ?
```

Дополнительные команды

- структура команды



- инкремент

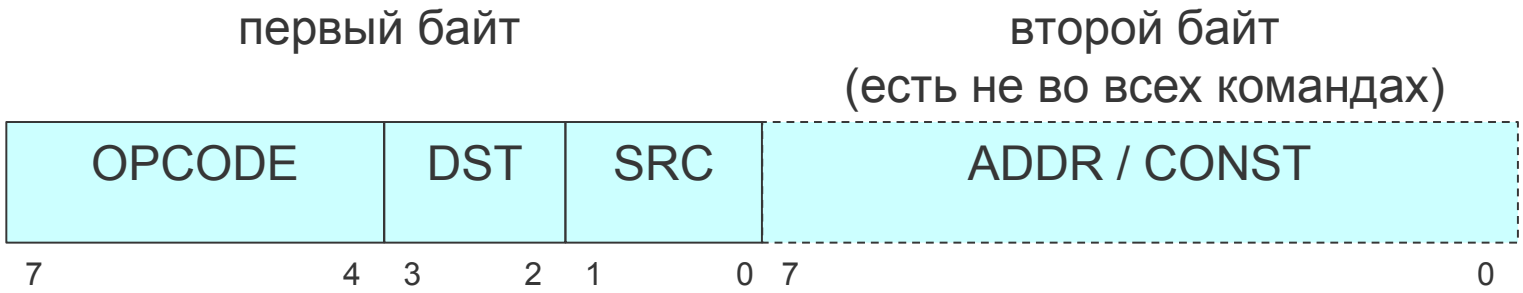
```
1110     ..   ..          -
```

например,

```
11100101 0xE1      R1 <- R1++
```

Очень простая система команд

- структура команды



- коды операций

1. **Команды пересылки данных**
   1. Из регистра в регистр
   2. Из памяти в регистр
   3. Из регистра в память
   4. Число в регистр
2. **Арифметические команды**
   5. Сложение
   6. Вычитание
   7. Умножение
   8. Деление (беззнаковое)
3. **Команды сравнения**
   9. Сравнить два регистра
4. **Команды передачи управления**
   10. Переход, если rf.c = 1
   11. Переход, если rf.s = 1
   12. Переход, если rf.z = 1
   13. Безусловный переход
5. **Дополнительные команды**
   14. Случайное число в регистр
   15. Инкремент числа в регистре

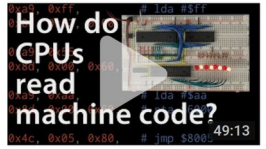
Очень простая система команд

	мнемоника	битовое представление
1	`Rdd <- Rss`	`0000ddss`
2	`Rdd <- @aaaaaaaa`	`0001dd?? aaaaaaaa`
3	`@aaaaaaaa <- Rss`	`0010??ss aaaaaaaa`
4	`Rdd <- nnnnnnnn`	`0011dd?? nnnnnnnn`
5	`Rdd <- Rdd + Rss`	`0100ddss`
6	`Rdd <- Rdd - Rss`	`0101ddss`
7	`Rdd <- Rdd * Rss`	`0110ddss`
8	`Rdd <- Rdd / Rss`	`0111ddss`
9	`RF <- Rdd ~ Rss`	`1000ddss`
10	`RC <- @aaaaaaaa (C)`	`1001???? aaaaaaaa`
11	`RC <- @aaaaaaaa (S)`	`1010???? aaaaaaaa`
12	`RC <- @aaaaaaaa (Z)`	`1011???? aaaaaaaa`
13	`RC <- @aaaaaaaa`	`1100???? aaaaaaaa`
14	`Rdd <- ?`	`1101dd??`
15	`Rdd <- Rss++`	`1110ddss`
16		

Задание к следующей лекции

- Макарова, Волков. Информатика. Учебник для вузов.
  - гл. 10
- Общеобразовательный материал
  - https://ru.wikipedia.org/wiki/Центральный_процессор
  - https://ru.wikipedia.org/wiki/Вычислительный_конвейер
- Ben Eater, серия видео про 6502
  - часть 1
    - https://www.youtube.com/watch?v=LnzuMJLZRdU
  - часть 2
    - https://www.youtube.com/watch?v=yl8vPW5hydQ
  - ...
- Внутри ARM1
  - https://www.youtube.com/watch?v=rXW-CLByNDs

"Hello, world" from scratch on a 6502 — Part 1



How do CPUs read machine code? — 6502 part 2

Материалы курса

- Презентации, материалы к лекциям, литература, задания на лабораторные работы
  - shorturl.at/jqRZ6