

# Информатика

## Аппаратные средства вычислительной техники и ассемблеры

Часть 2

Гирик Алексей Валерьевич

Университет ИТМО  
2022

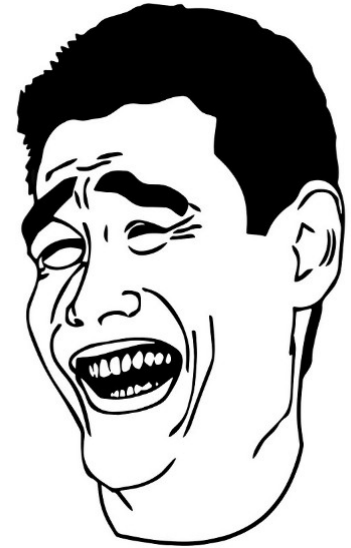
# Материалы курса

- Презентации, материалы к лекциям, литература, задания на лабораторные работы
  - [shorturl.at/jqRZ6](https://shorturl.at/jqRZ6)



# Небольшой оффтоп

- ✓ **BOOLEAN** – 1-битный логический;
- ✓ **INT32** – 32-битные подписанные числа;
- ✓ **INT64** – 64-битные подписанные числа;
- ✓ **INT96** – 96-битные подписанные числа;
- ✓ **FLOAT** – IEEE 32-битные значения с плавающей точкой;
- ✓ **DOUBLE** – IEEE 64-битные значения с плавающей точкой;
- ✓ **BYTE\_ARRAY** – произвольно длинные байтовые массивы.



Данные в таблице

A	B	C
A1	B1	C1
A2	B2	C2
A3	B3	C3

Текстовый файл CSV

A1	B1	C1	A2	B2	C2	A3	B3	C3
----	----	----	----	----	----	----	----	----

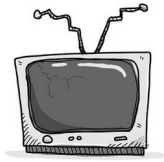
Файл в формате Parquet

A1	A2	A3	B1	B2	B3	C1	C2	C3
----	----	----	----	----	----	----	----	----

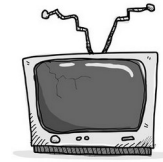
Представление данных в формате Apache Parquet

<https://www.bigdataschool.ru/wiki/parquet>

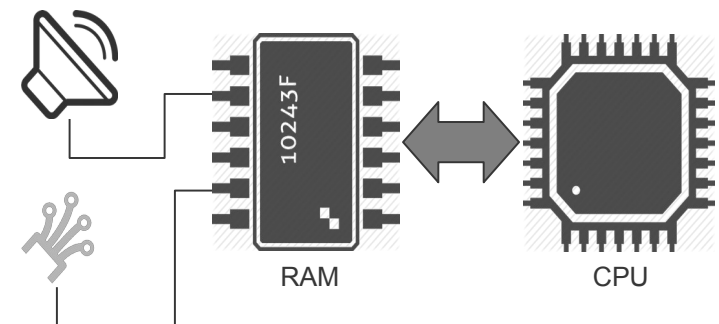
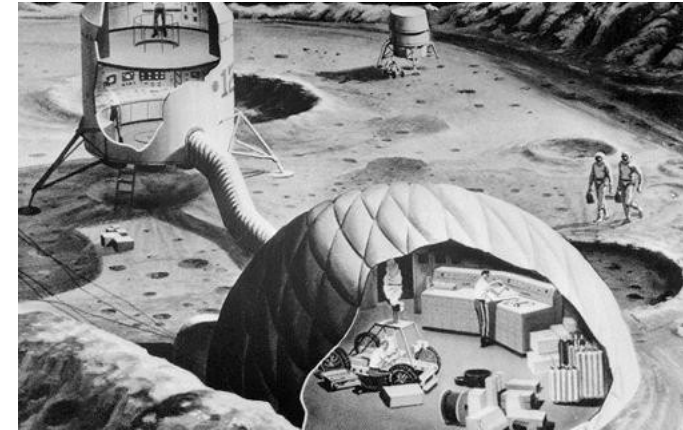
Очень простой ассемблер для  
очень простого процессора



# Previously on...



- 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

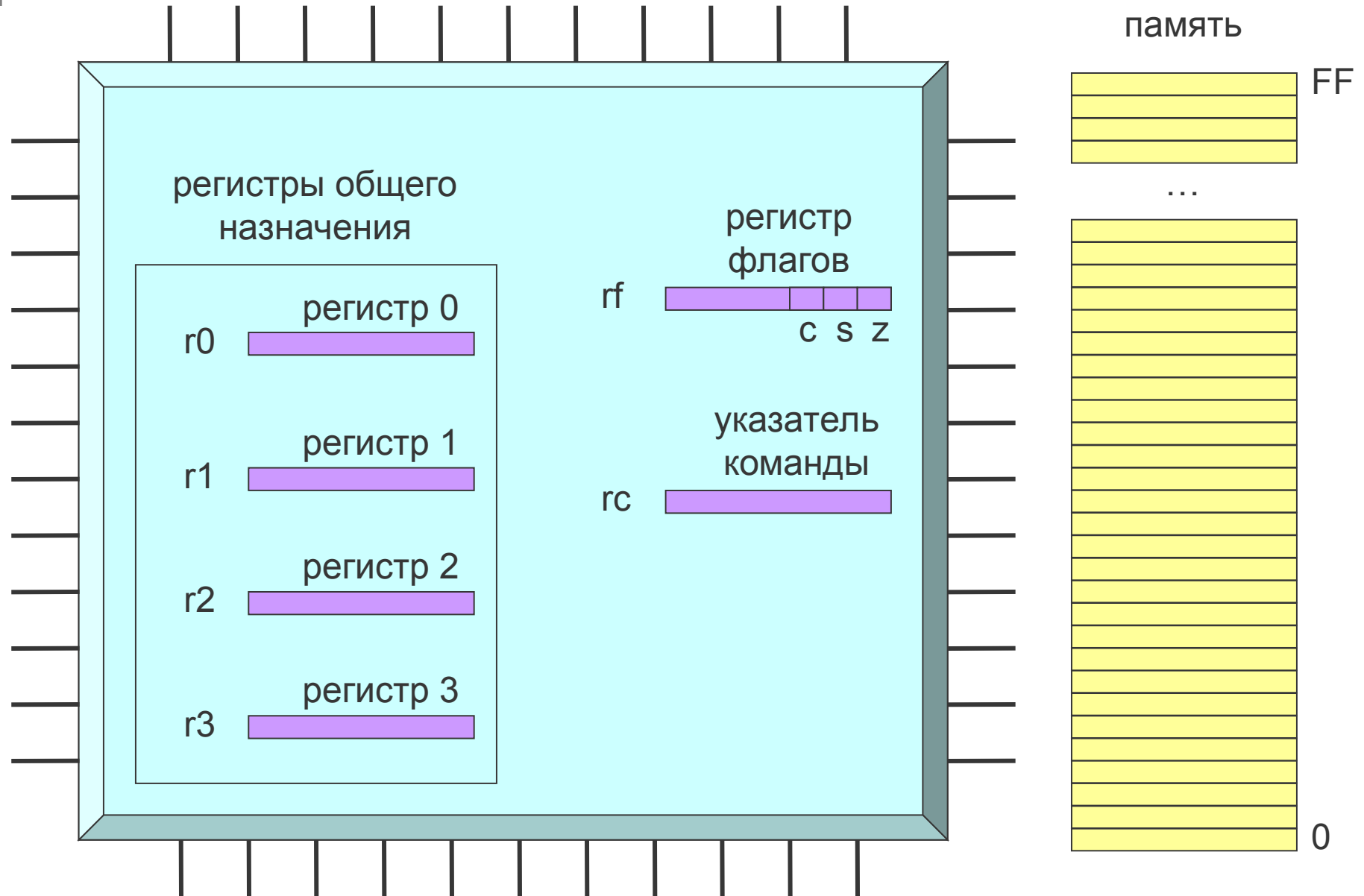


условная схема  
системы контроля  
содержания кислорода

# Очень простой процессор

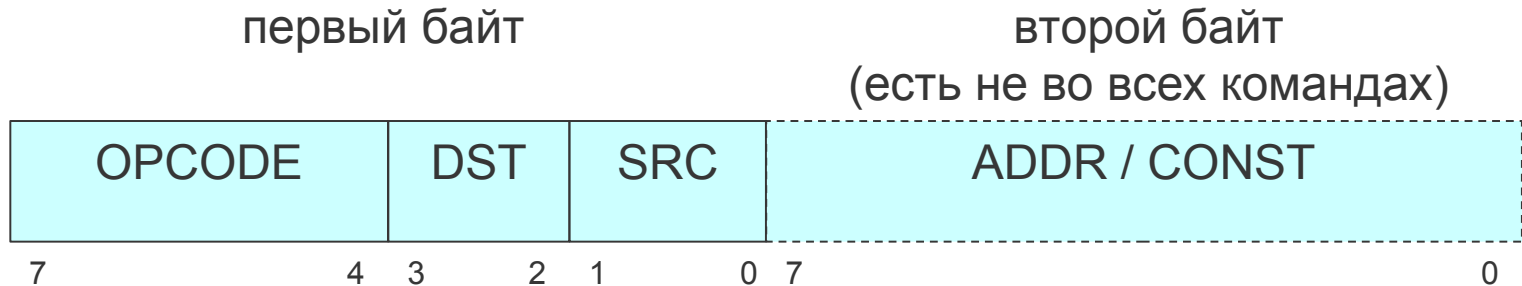
- разрядность
- объем непосредственно адресуемой памяти
- количество регистров и их назначение
- значения регистров после включения питания
- стартовый адрес
- система команд
  - коды операций
  - типы адресации
- представление чисел
  - разрядность
  - знак

# Очень простой процессор



# Очень простая система команд

- структура команды





# От программирования в двоичных кодах к мнемонической записи

**машинный код**  
**(битовое представление)**

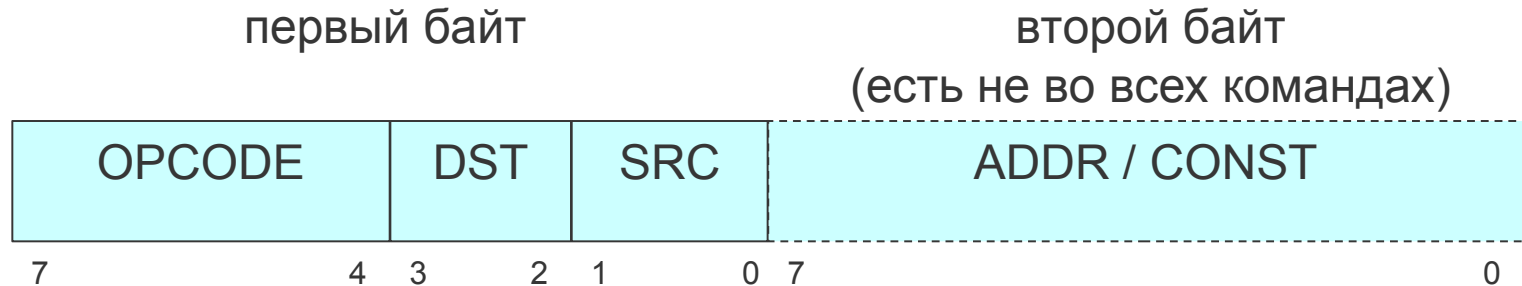
00110000 00000010  
00000100  
00100001 11111111

**мнемоническая запись**  
**(язык ассемблера)**

R0 ← 2  
R1 ← R0  
@0xFF ← R1

# Очень простая система команд

## ■ структура команды



## ■ коды операций

### Команды пересылки данных

1. Из регистра в регистр
2. Из памяти в регистр
3. Из регистра в память
4. Число в регистр

### Арифметические команды

5. Сложение
6. Вычитание
7. Умножение
8. Деление (беззнаковое)

### Команды сравнения

9. Сравнить два регистра

### Команды передачи управления

10. Переход, если  $rf.c = 1$
11. Переход, если  $rf.s = 1$
12. Переход, если  $rf.z = 1$
13. Безусловный переход

### Дополнительные команды

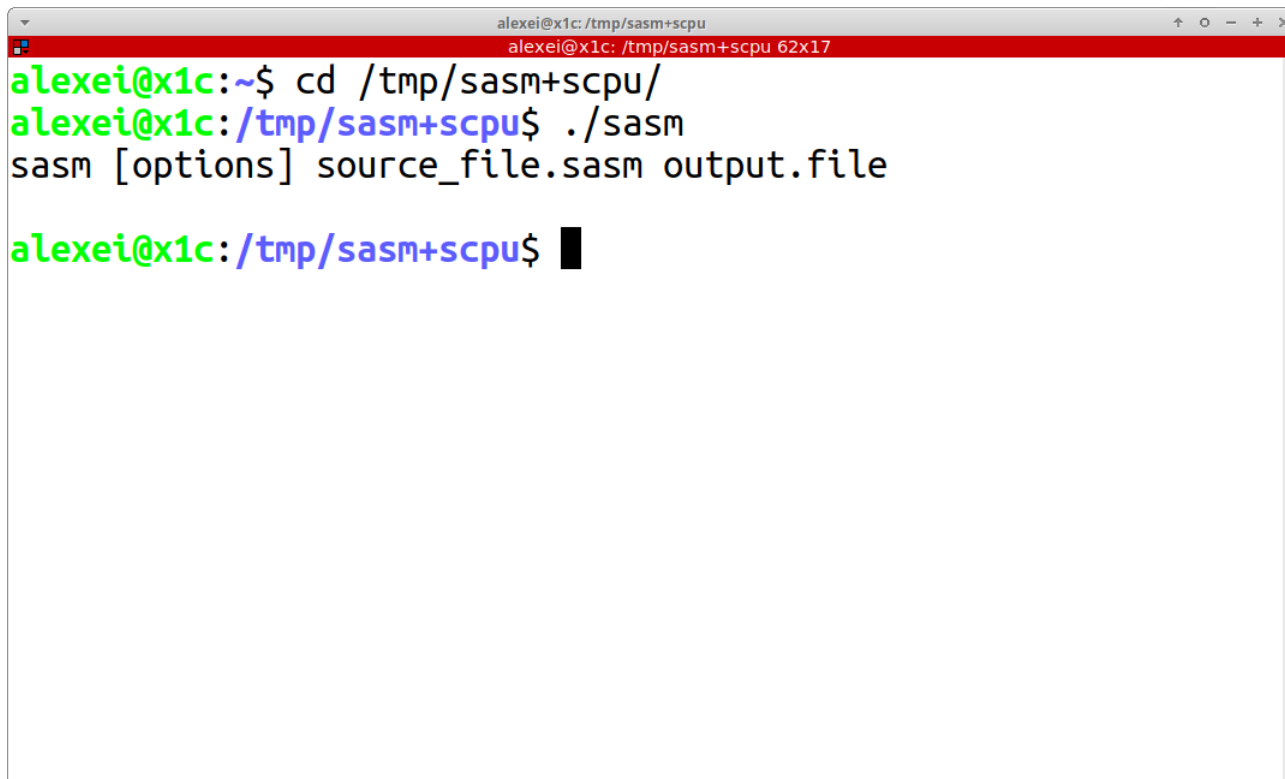
14. Случайное число в регистр
15. Инкремент числа в регистре

# Очень простая система команд

	мнемоника	битовое представление
1	Rdd ← Rss	0000ddss
2	Rdd ← @aaaaaaaa	0001dd?? aaaaaaaaa
3	@aaaaaaaa ← Rss	0010??ss aaaaaaaaa
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???? aaaaaaaaa
11	RC ← @aaaaaaaa (S)	1010???? aaaaaaaaa
12	RC ← @aaaaaaaa (Z)	1011???? aaaaaaaaa
13	RC ← @aaaaaaaa	1100???? aaaaaaaaa
14	Rdd ← ?	1101dd??
15	Rdd ← Rss++	1110ddss
16		

# Очень простой ассемблер

- sasm
  - консольная программа – запускается из командной строки



```
alexei@x1c: /tmp/sasm+scpu
alexei@x1c: /tmp/sasm+scpu 62x17
alexei@x1c:~$ cd /tmp/sasm+scpu/
alexei@x1c: /tmp/sasm+scpu$ ./sasm
sasm [options] source_file.sasm output.file

alexei@x1c: /tmp/sasm+scpu$ █
```

# Очень простой пример

Задача: Найти сумму трех случайных чисел и записать её по адресу 0xFF.

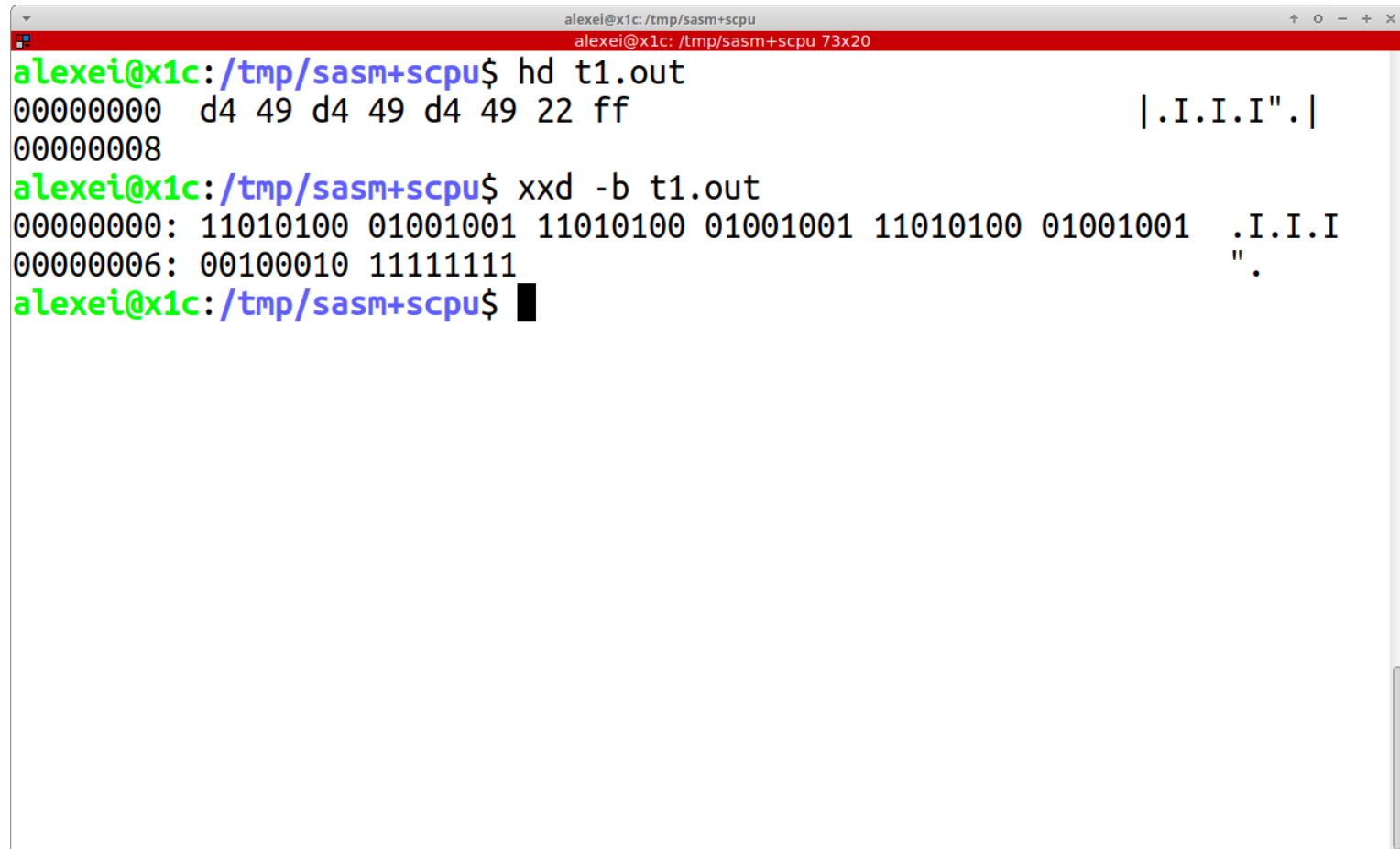
# Очень простой пример

Задача: Найти сумму трех случайных чисел и записать её по адресу 0xFF.

```
1  R01 <- ?
2  R02 <- R02 + R01
3  R01 <- ?
4  R02 <- R02 + R01
5  R01 <- ?
6  R02 <- R02 + R01
7  @0xFF <- R02
```

# Очень простой пример

Что получаем после ассемблирования?



```
alexei@x1c: /tmp/sasm+scpu
alexei@x1c: /tmp/sasm+scpu 73x20
alexei@x1c:/tmp/sasm+scpu$ hd t1.out
00000000  d4 49 d4 49 d4 49 22 ff          |.I.I.I".|
00000008
alexei@x1c:/tmp/sasm+scpu$ xxd -b t1.out
00000000: 11010100 01001001 11010100 01001001 11010100 01001001  .I.I.I
00000006: 00100010 11111111          ".
alexei@x1c:/tmp/sasm+scpu$
```

# Очень простой пример

Задача: Найти сумму трех случайных чисел и записать её по адресу 0xFF.

```
1      > R01 <- ?  
2      R02 <- R02 + R01  
  
4      @0xFF <- R02
```



# Очень простой пример

Задача: Найти сумму трех случайных чисел и записать её по адресу 0xFF.

```
1          R03 <- 3
2  LOOP:   R01 <- ?
3          R02 <- R02 + R01
4          R00 <- R00++
5          RF <- R00 ~ R03
6          RC <- @LOOP (S)
7          @0xFF <- R02
```

# Очень простой пример

Задача: Найти сумму трех случайных чисел и записать её по адресу 0xFF.

```
1          R03 <- -3
2  LOOP:   R01 <- ?
3          R02 <- R02 + R01
4          R03 <- R03++
5          RC  <- @LOOP (S)
6          @0xFF <- R02
```

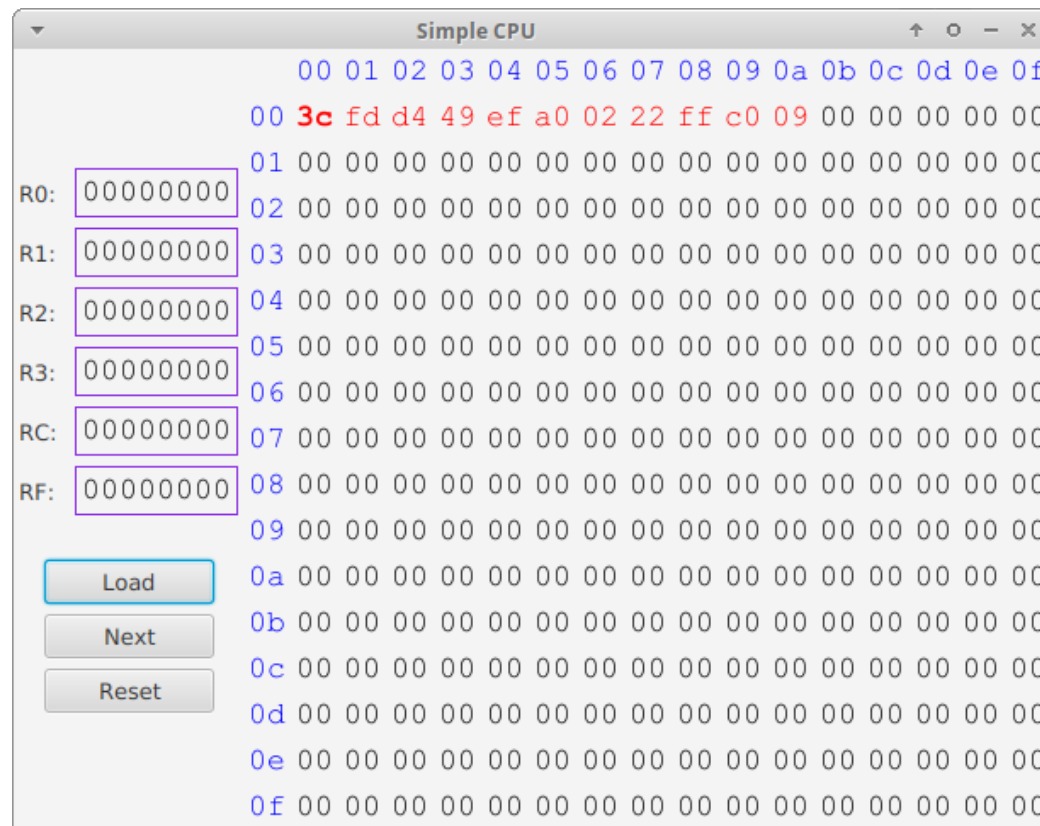
# Очень простой пример

Задача: Найти сумму трех случайных чисел и записать её по адресу 0xFF.

```
1          R03 <- -3
2  LOOP:   R01 <- ?
3          R02 <- R02 + R01
4          R03 <- R03++
5          RC  <- @LOOP (S)
6          @0xFF <- R02
7  STOP:   RC  <- @STOP
```

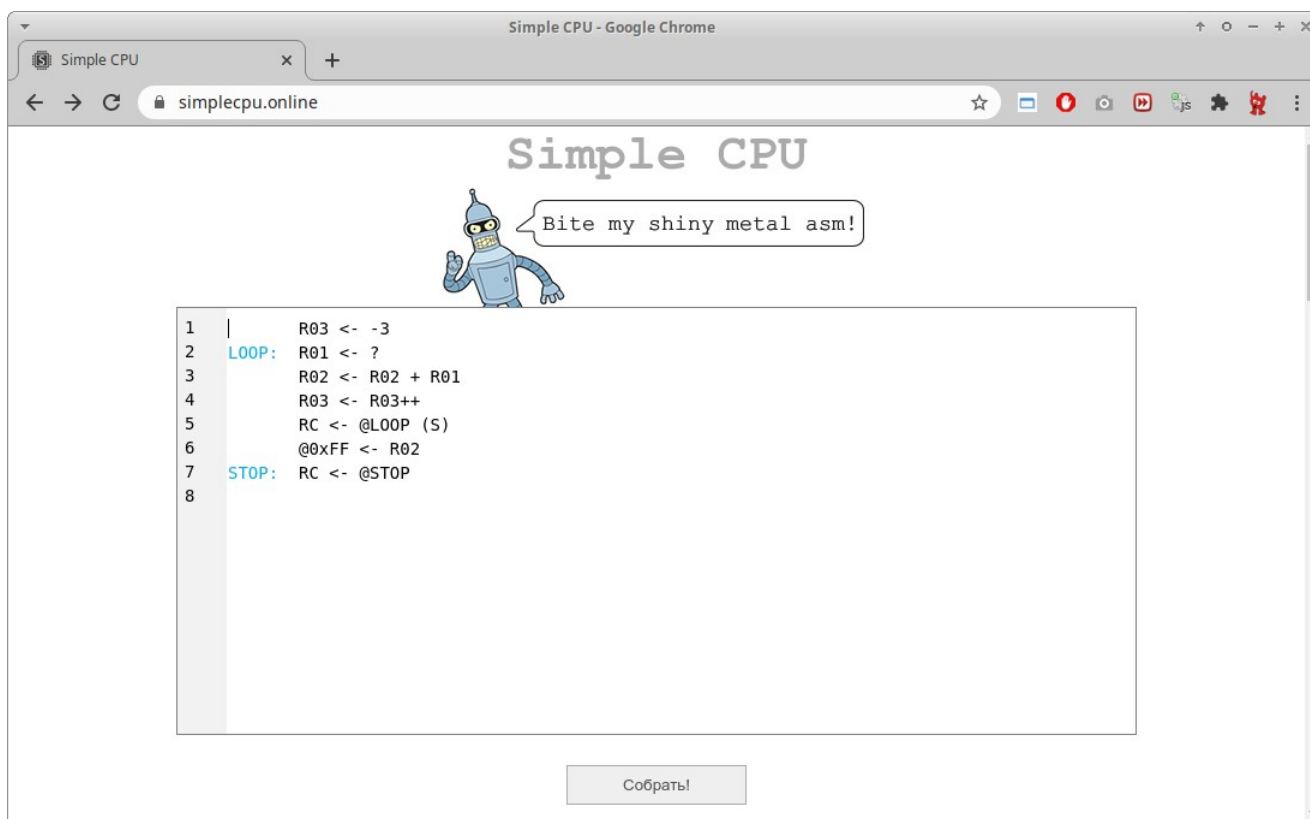
# Эмулятор простого процессора

- Simple CPU
  - эмулятор простого процессора – GUI на Java FX



# simplecpu.online – 2 в одном!

<https://simplecpu.online>



# simplecpu.online – 2 в одном!

Simple CPU - Google Chrome

simplecpu.online/scpu

## Simple CPU

I'm gonna go build my own CPU!  
With JTAG and buffers!

```
1 R03 <- -3
2 LOOP: R01 <- ?
3       R02 <- R02 + R01
4       R03 <- R03++
5       RC <- @LOOP (S)
6       @0xFF <- R02
7 STOP: RC <- @STOP
8
```

00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f

00 3c fd d4 49 ef a0 02 22 ff c0 09 00 00 00 00 00

01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

03 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

05 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

06 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

07 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

09 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

0a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

0b 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

0c 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

0d 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

0e 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

0f 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

R0: 0 0 0 0 0 0 0 0

R1: 0 0 0 0 0 0 0 0

R2: 0 0 0 0 0 0 0 0

R3: 0 0 0 0 0 0 0 0

RC: 0 0 0 0 0 0 0 0

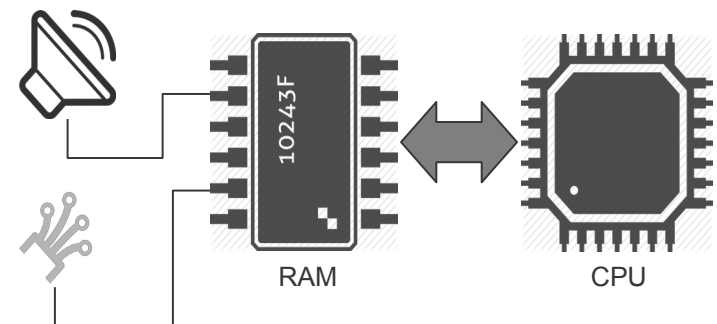
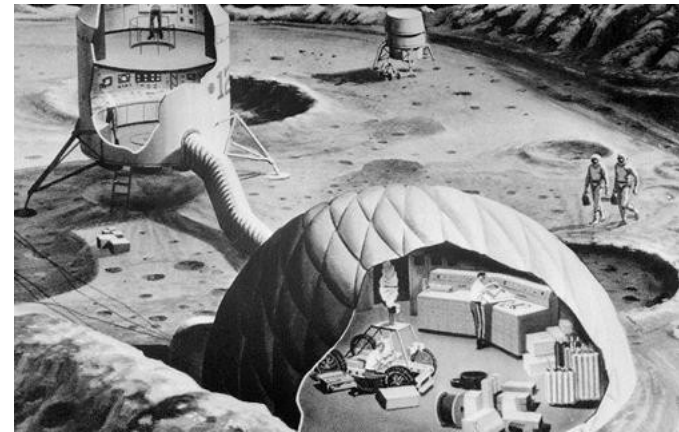
RF: 0 0 0 0 0 0 0 0

Шаг! Старт! Сброс!

Долгожданное решение очень  
простой задачи

# 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 алгоритм

00010101
00000010

FF – динамик (работает, если не равно 0)

FE – текущее значение датчика

FD – нормальное значение величины содержания  $O_2$

FC – максимально допустимое отклонение от нормы

# Очень простой алгоритм

00010101
00000010

FF – динамик (работает, если не равно 0)

FE – текущее значение датчика

FD – нормальное значение величины содержания  $O_2$

FC – максимально допустимое отклонение от нормы

1. получить текущее значение датчика
2. получить нормальное значение
3. найти разницу между текущим значением и нормальным
4. если результат (текущее отклонение) отрицательный, поменять знак результата
5. получить максимально допустимое отклонение
6. сравнить текущее и максимально допустимое отклонение
7. если текущее отклонение меньше максимально допустимого, выключить динамик и перейти к шагу 1
8. включить динамик
9. перейти к шагу 1

```
R00 <- @0xFE
R01 <- @0xFD
R00 <- R00 - R01      ; тек - норм
RC  <- @L1 (S)
RC  <- @L2
L1: R10 <- R10 - R00   ; тек < норм
R00 <- R10
R10 <- 0
L2: R11 <- @0xFC
RF  <- R00 ~ R11
RC  <- @L3 (S)
RC  <- @L3 (Z)
@0xFF <- R00
RC  <- @0
L3: @0xFF <- R10
RC  <- @0
```

# Очень простой алгоритм

00010101
00000010

FF – динамик (работает, если не равно 0)

FE – текущее значение датчика

FD – нормальное значение величины содержания  $O_2$

FC – максимально допустимое отклонение от нормы

1. получить текущее значение датчика
2. получить нормальное значение
3. найти разницу между текущим значением и нормальным

```
R00 <- @0xFE
```

```
R01 <- @0xFD
```

```
R00 <- R00 - R01 ; тек - норм
```

# Очень простой алгоритм

00010101
00000010

FF – динамик (работает, если не равно 0)

FE – текущее значение датчика

FD – нормальное значение величины содержания  $O_2$

FC – максимально допустимое отклонение от нормы

4. если результат (текущее отклонение) отрицательный, поменять знак результата

```
RC  <- @L1 (S)
```

```
RC  <- @L2
```

```
L1:  R10 <- R10 - R00 ; тек < норм
```

# Очень простой алгоритм

00010101
00000010

FF – динамик (работает, если не равно 0)

FE – текущее значение датчика

FD – нормальное значение величины содержания  $O_2$

FC – максимально допустимое отклонение от нормы

5. получить максимально допустимое отклонение
6. сравнить текущее и максимально допустимое отклонение

```
R00 <- R10
R10 <- 0
L2:  R11 <- @0xFC
RF   <- R00 ~ R11
```

# Очень простой алгоритм

00010101
00000010

FF – динамик (работает, если не равно 0)

FE – текущее значение датчика

FD – нормальное значение величины содержания  $O_2$

FC – максимально допустимое отклонение от нормы

7. если текущее отклонение меньше максимально допустимого, выключить динамик и перейти к шагу 1

```
RC  <- @L3 (S)
```

```
RC  <- @L3 (Z)
```

```
@0xFF <- R00
```

```
RC  <- @0
```

# Очень простой алгоритм

00010101
00000010

FF – динамик (работает, если не равно 0)

FE – текущее значение датчика

FD – нормальное значение величины содержания  $O_2$

FC – максимально допустимое отклонение от нормы

1. включить динамик
2. перейти к шагу 1

```
L3:    @0xFF <- R10
        RC    <- @0
```

# Очень простая программа

	R00 <- @0xFE	00: 00010000
		01: 11111110
	R01 <- @0xFD	02: 00010100
		03: 11111101
	R00 <- R00 - R01	04: 01010001
	RC <- @L1 (S)	05: 10100000
		06: _____
	RC <- @L2	07: 11000000
		08: _____
L1:	R10 <- R10 - R00	09: 01011000
	R00 <- R10	0A: 00000010
	R10 <- 0	0B: 00111000
		0C: 00000000
L2:	R11 <- @0xFC	0D: 00010000
		0E: 11111100
	RF <- R00 ~ R11	0F: 10000011
	RC <- @L3 (S)	10: 10100000
		11: _____
	RC <- @L3 (Z)	12: 10110000
		13: _____
	@0xFF <- R00	14: 00100000
		15: 11111111
	RC <- @0	16: 11000000
		17: 00000000
L3:	@0xFF <- R10	18: 00100010
		19: 11111111
	RC <- @0	20: 11000000
		21: 00000000



# Очень простая программа

```
R00 <- @0xFE
R01 <- @0xFD
R00 <- R00 - R01
RC <- @L1 (S)
RC <- @L2

L1:  R10 <- R10 - R00
     R00 <- R10
     R10 <- 0

L2:  R11 <- @0xFC
     RF <- R00 ~ R11
     RC <- @L3 (S)
     RC <- @L3 (Z)

     @0xFF <- R00
     RC <- @0

L3:  @0xFF <- R10
     RC <- @0
```

00: 00010000  
01: 11111110  
02: 00010100  
03: 11111101  
04: 01010001  
05: 10100000  
06: **00001001**  
07: 11000000  
08: \_\_\_\_\_  
09: 01011000  
0A: 00000010  
0B: 00111000  
0C: 00000000  
0D: 00010000  
0E: 11111100  
0F: 10000011  
10: 10100000  
11: \_\_\_\_\_  
12: 10110000  
13: \_\_\_\_\_  
14: 00100000  
15: 11111111  
16: 11000000  
17: 00000000  
18: 00100010  
19: 11111111  
20: 11000000  
21: 00000000

← reference backpatching  
(исправление ссылок)

# Предложения по усовершенствованию СИСТЕМЫ КОМАНД

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

# Задача № 1

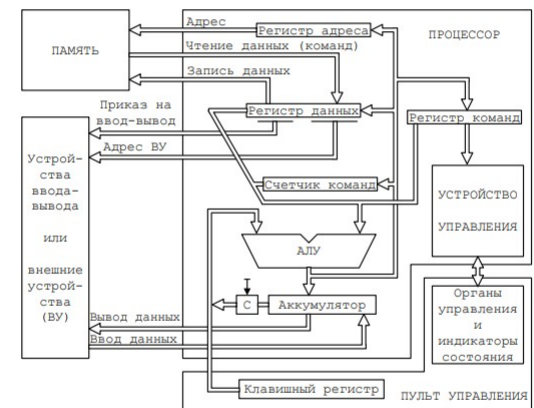
- Напишите программу на очень простом ассемблере для очень простого процессора, которая позволяет выяснить, четное ли число содержится в ячейке FF. Если в FF содержится четное число, то программа должна записать ноль в ячейку FE, иначе – ненулевое число.

# Введение в низкоуровневое программирование

- Кириллов В.В.  
Архитектура базовой ЭВМ
  - <https://books.ifmo.ru/file/pdf/761.pdf>

В.В. Кириллов

## АРХИТЕКТУРА БАЗОВОЙ ЭВМ Учебное пособие



Санкт-Петербург

2010

# Введение в методы построения процессорных архитектур

- *Harris S., Harris D.*  
*Digital Design and Computer  
Architecture*

**Digital Design and  
Computer Architecture**  
RISC-V Edition

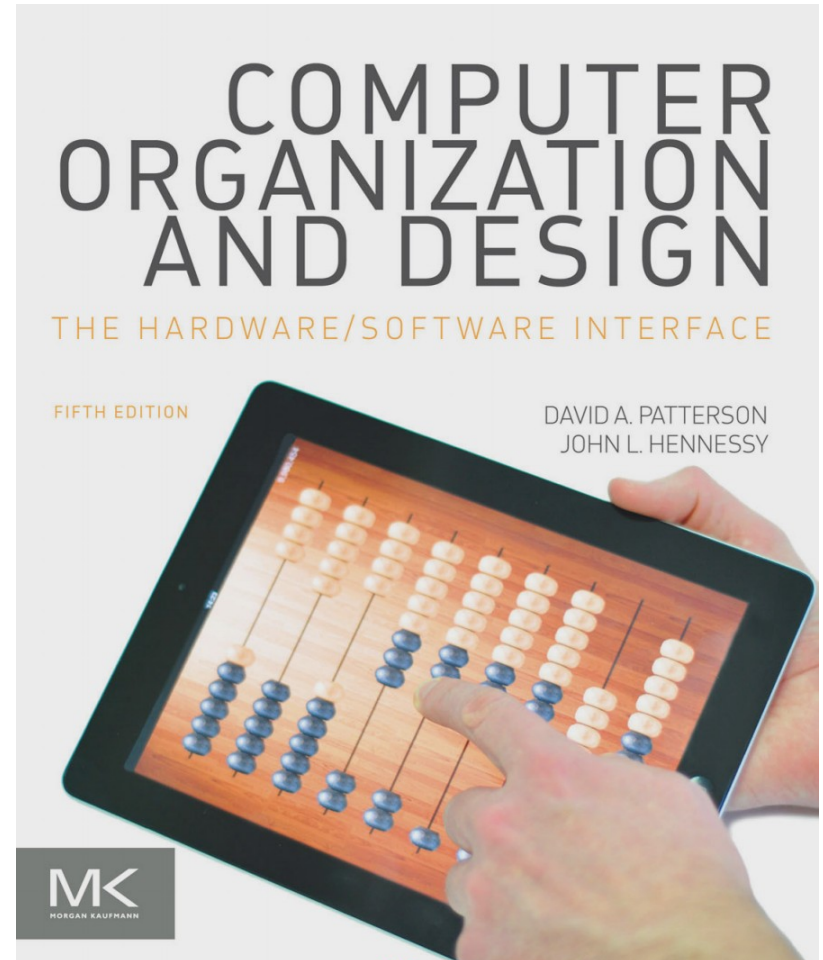


**MK**  
MORGAN KAUFMANN

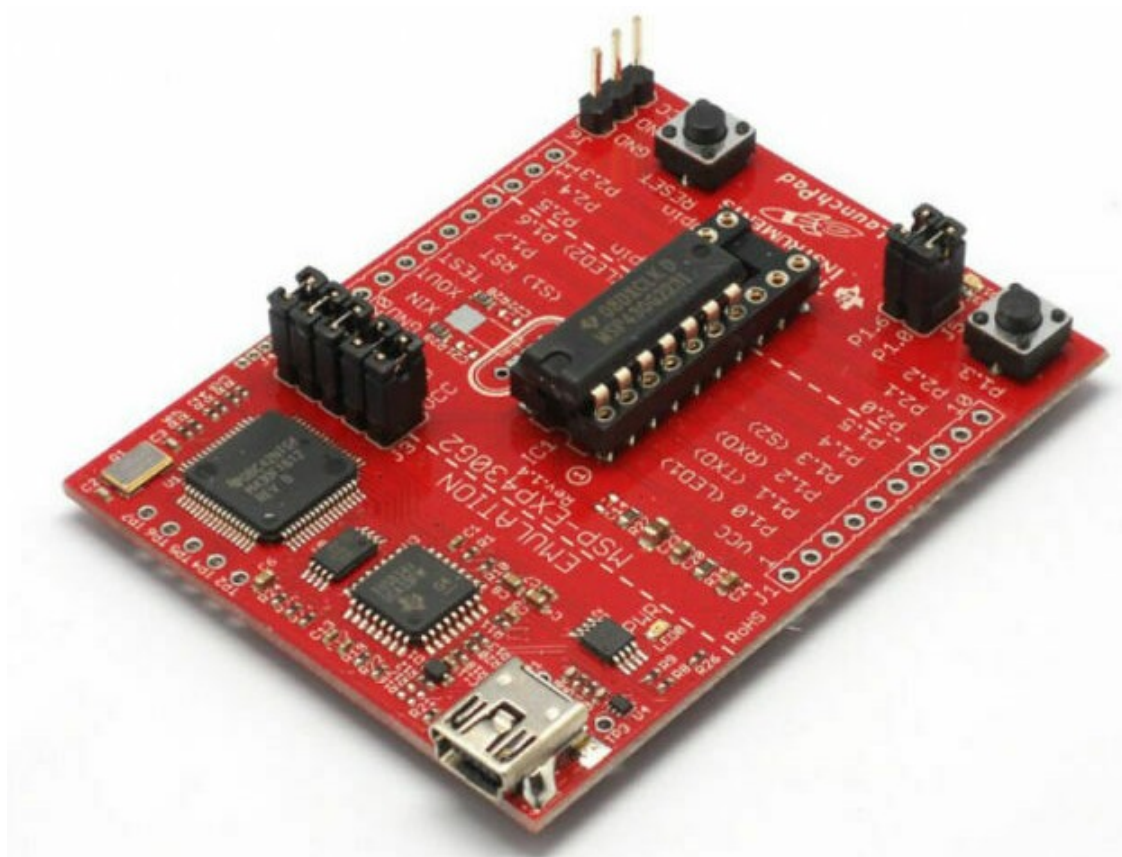
Sarah L Harris  
David Money Harris

# Введение в методы построения процессорных архитектур

- *Patterson D., Hennessy J.*  
Computer organization and  
design



# Микропроцессор TI MSP430



<http://we.easyelectronics.ru/Akay/rukovodstvo-01-pristupaya-k-rabote.html>

<http://mspsci.blogspot.com/2010/07/tutorial-01-getting-started.html>

# Лабораторная работа № 1



# Что нам понадобится для первой лабораторной работы

- очень простой процессор
- очень простой ассемблер
- очень простые варианты лабораторных
- базовые знания двоичной арифметики



# Задание к следующей лекции

- Харрис, Харрис. Цифровая схемотехника и архитектура компьютера
  - гл. 1
  - гл. 6, разделы 6.1 – 6.5
- <https://simplecpu.online>
  - Решить задачу № 1

