

# Команды процессора RISC v.s. CISC

Лекция №2

Немного организационных моментов

**OFFTOP**

# Оценивание задач

- После прохождения тестов - Code Review
- Каждая задача - 100 баллов
- Неправильные посылки штрафуются на 10 баллов
- Не снижаемый остаток - 20 баллов
- Срок сдачи - 2 недели. На первой неделе Code Review проводится в лояльном формате, на второй - более строго

# Как нагадить в ejudge

- system("sudo rm -rf /")
- system("rm -rf \$HOME")
- DDoS на веб-морду
- ....
- фантазия ФИВТов безгранична!

# Как с этим бороться

- Сервер ejudge - это виртуалка
- Еженедельно делается бэкап образа
- Внутри виртуалки мы не заморачиваемся с безопасностью

*но при этом:*

- Снаружи стоит nginx в качестве proxy и ведет подробный лог

# 273 УК РФ

1. Создание, распространение или использование компьютерных программ либо иной компьютерной информации, заведомо предназначенных для несанкционированного уничтожения, блокирования, модификации, копирования компьютерной информации или нейтрализации средств защиты компьютерной информации, -  
наказываются ограничением свободы на срок до четырех лет, либо принудительными работами на срок до четырех лет, либо лишением свободы на тот же срок со штрафом в размере до двухсот тысяч рублей или в размере заработной платы или иного дохода осужденного за период до восемнадцати месяцев.
2. Деяния, предусмотренные частью первой настоящей статьи, совершенные группой лиц по предварительному сговору или организованной группой либо лицом с использованием своего служебного положения, а равно причинившие крупный ущерб или совершенные из корыстной заинтересованности, -  
наказываются ограничением свободы на срок до четырех лет, либо принудительными работами на срок до пяти лет с лишением права занимать определенные должности или заниматься определенной деятельностью на срок до трех лет или без такового, либо лишением свободы на срок до пяти лет со штрафом в размере от ста тысяч до двухсот тысяч рублей или в размере заработной платы или иного дохода осужденного за период от двух до трех лет или без такового и с лишением права занимать определенные должности или заниматься определенной деятельностью на срок до трех лет или без такового.

Теперь уже содержательная часть лекции

# **КОМАНДЫ ПРОЦЕССОРА**

# С точки зрения пользователя (на примере i386)

Регистры	
%eax	%esi
%ebx	%edi
%ecx	%ebp
%edx	%esp

Флаги		
ZF	CF	SF
Указатель на текущую команду		
PC (32 бит)		

# Команды x86

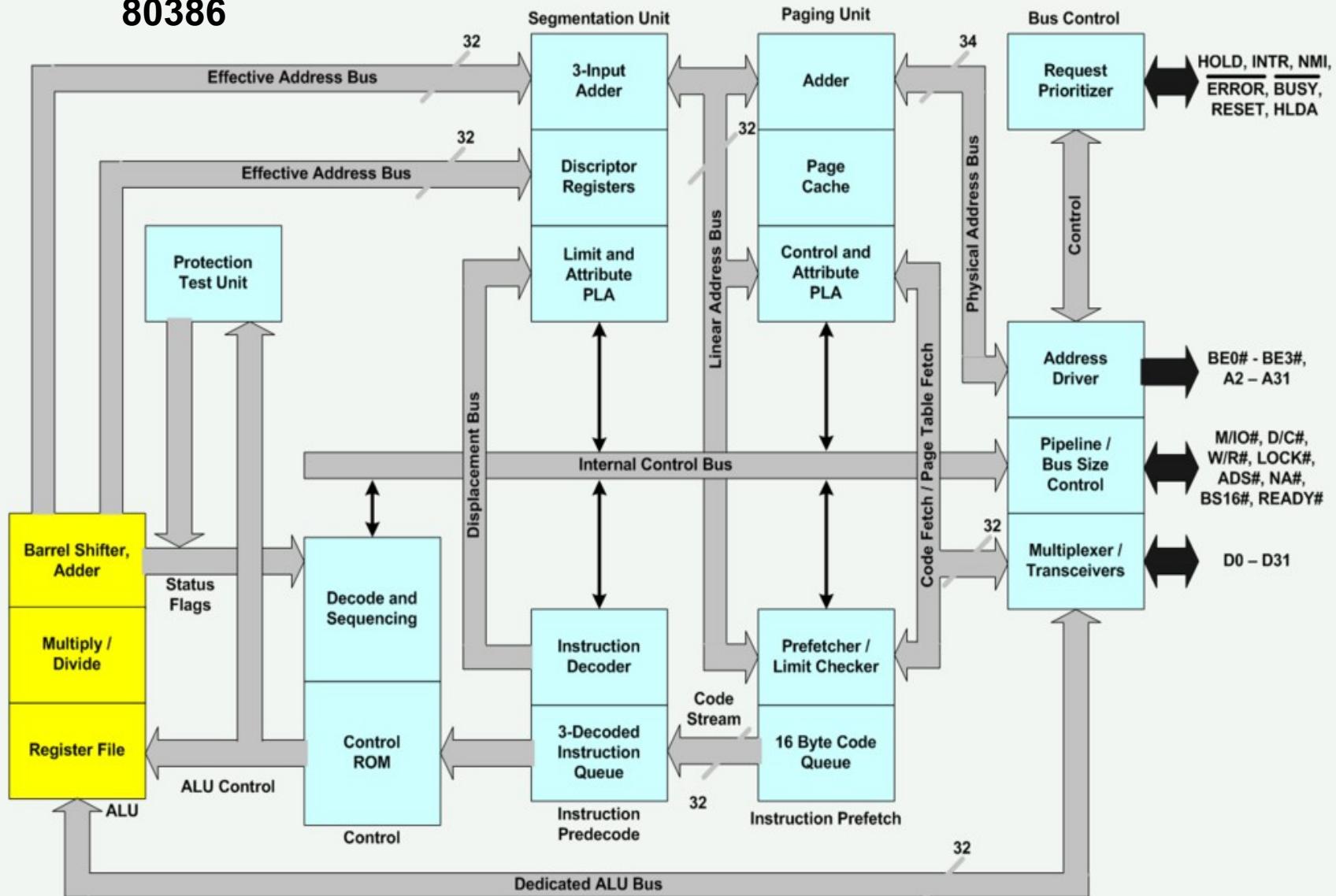
Байты	0	1	2	3	4	5
nop	0x00					
halt	0x10					
rrmovl <i>rA, rB</i>	0x20	<i>rA rB</i>				
irmovl <i>V, rB</i>	0x30	0x8 <i>rB</i>				<i>V</i>
call <i>Dest</i>	0x80					<i>Dest</i>
pushl <i>rA</i>	0xA0	<i>rA 0x8</i>				
popl <i>rA</i>	0xB0	<i>rA 0x8</i>				

Команды кодируются переменным количеством байт

# Какие команды выполняет процессор

- Арифметические
- Управляющие

# 80386



PLA: Programmable Logic Array

# Представление структур программы в виде простых инструкций

**while (true)**

{

<statement 1>

. . . .

<statement N>

}

**Loop\_Start:**

<statement 1>

. . . .

<statement N>

**br %Loop\_Start**

# Представление структур программы в виде простых инструкций

```
while (<cond>)
{
    <statement 1>
    . . .
    <statement N>
}
```

**Loop\_Start:**

%cond = . . .  
**br i1 %cond,**  
  └ **label %Loop\_Body,**  
  └ **label %Loop\_End**

**Loop\_Body:**

<statement 1>  
. . .  
<statement N>  
**br %Loop\_Start**

**Loop\_End:**

. . .

# Представление структур программы в виде простых инструкций

```
for (<init>;<cond>;<incr>)
{
    <statement 1>
    . . .
    <statement N>
}
```

<init statements>  
**Loop\_Start:**  
%cond = . . .  
**br i1 %cond,**  
  └ **label %Loop\_Body,**  
  └ **label %Loop\_End**  
**Loop\_Body:**  
<statement 1>  
. . .  
<statement N>  
<increment statement>  
**br %Loop\_Start**  
**Loop\_End:**  
<cleanup statements>  
. . .

# Наборы команд Z80, x86 и PDP-11/VAX

## *Complex Instruction Set Computing (CISC)*

- Их много - на все случаи жизни
- Кодируются переменным числом байт
- Разные режимы адресации
- Упрощают жизнь программисту на ассемблере/машинных кодах

# Примеры команд Intel x86 со сложной логикой

## **loop Address**

1. Уменьшает значение регистра %ecx на 1
2. Если значение %ecx==0,  
то %eip+=**sizeof(loop)**,  
иначе %eip=Address

# Примеры команд Intel x86 со сложной логикой

**movl Rsrc, Offset(Rbase, Rindex, Size)**

1. Считывает значение из регистра *Rindex*
2. Умножает его на *Size*
3. Прибавляет к нему значение из регистра *Rbase*
4. Прибавляет к нему значение *Offset*
5. На шине адреса выставляет полученное значение
6. Записывает значение из регистра *Rsrc* в память

# Программирование CPU

## Совсем на низком уровне:

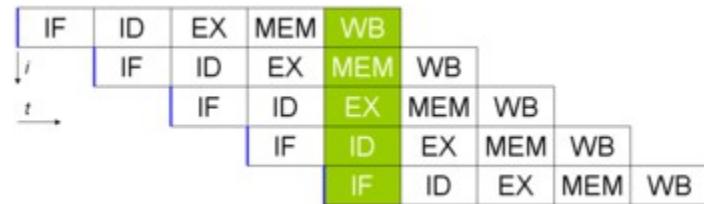
- Машинные коды
- Ассемблер
- Макро-ассемблер

## Высокоуровневые языки:

- 1966: BCPL (Basic Combined Programming Language)
- 1969: Язык Би (B)
- 1972: Язык Си (C)

# Конвейер

- Instruction Fetch
- Instruction Decode
- Execute
- Memory Access
- Register Write Back



# Проблемы конвейеризации

- Инструкции имеют разную длину в байтах
- Разные инструкции выполняются за различное число тактов
- Работа как с памятью, так и с регистрами - действуются разные блоки процессора

Процессоры AVR, ARM, MIPS, PowerPC, ...

*Reduced Instruction Set Computing  
(RISC)*

- Только простейшие инструкции фиксированной длины и (для большинства) с одинаковым временем выполнения
- Адресация только R-R

# CISC (x86)

<b>080483<b>4</b>:</b>	55	pushl %ebp
<b>080483<b>5</b>:</b>	89 e5	movl %ebp, %esp
<b>080483<b>7</b>:</b>	8b 45 0c	movl 0xc(%ebp), %eax
<b>080483<b>a</b>:</b>	03 45 08	addl 0x8(%ebp), %eax
<b>080483<b>bd</b>:</b>	01 05 64 94 04 08	addl 0x8049464, %eax
<b>080483<b>c3</b>:</b>	89 ec	movl %esp, %ebp
<b>080483<b>c5</b>:</b>	5d	popl %ebp
<b>080483<b>c6</b>:</b>	c3	ret
<b>080483<b>c7</b>:</b>	90	nop

# RISC (Atmel AVR)

<b>0000</b> :	03 e0	ldi r16, 0b00000011
<b>0002</b> :	07 bb	out DDRB, r16
<b>0004</b> :	03 e0	ldi r16, 0b00000011
<b>0006</b> :	08 bb	out PORTB, r16
<b>0008</b> :	08 b3	in r16, PORTB
<b>000a</b> :	00 95	com r16
<b>000c</b> :	03 70	andi r16, 0b00000011
<b>000e</b> :	08 bb	out PORTB, r16
<b>0010</b> :	fb cf	rjmp -4

# Обращение к памяти RISC v.s. CISC

## x86

01 05 64 94 04 08      addl 0x8049464, %eax

*6 байт в одной составной команде*

## AVR

c4 e3                  ldi 29, Low(0x1234)

d2 e1                  ldi 28, High(0x1234)

18 80                  ld r1, Y

01 0c                  add r0, r1

*8 байт в четырех простых командах*

# RISC для x86 ISA

- i486 - появление конвейера только для подмножества простых команд
- i586 (Первый Пень) - два конвейера: для простых команд и для не очень простых команд
- i686 (современная архитектура IA-32) - микроядро RISC; команды CISC предварительно транслируются во внутреннее представление самим процессором (микрокод)

# Современные RISC-процессоры

- PowerPC: Playstation 3, XBox 360, суперкомпы IBM
- MIPS: WiFi/Bluetooth-адAPTERы, DSP в телевизорах  
Процессоры 1890ВМ9Я (2 ядра, 1ГГц) и 1907ВМ028  
(вместо 1 ядро и смешные 150МГц, зато в космос  
летает)
- ARM: весь китайский ширпотреб (iPhone etc.)
- AVR: Arduino и его клоны + ещё много где  
встречается

# Микроконтроллеры



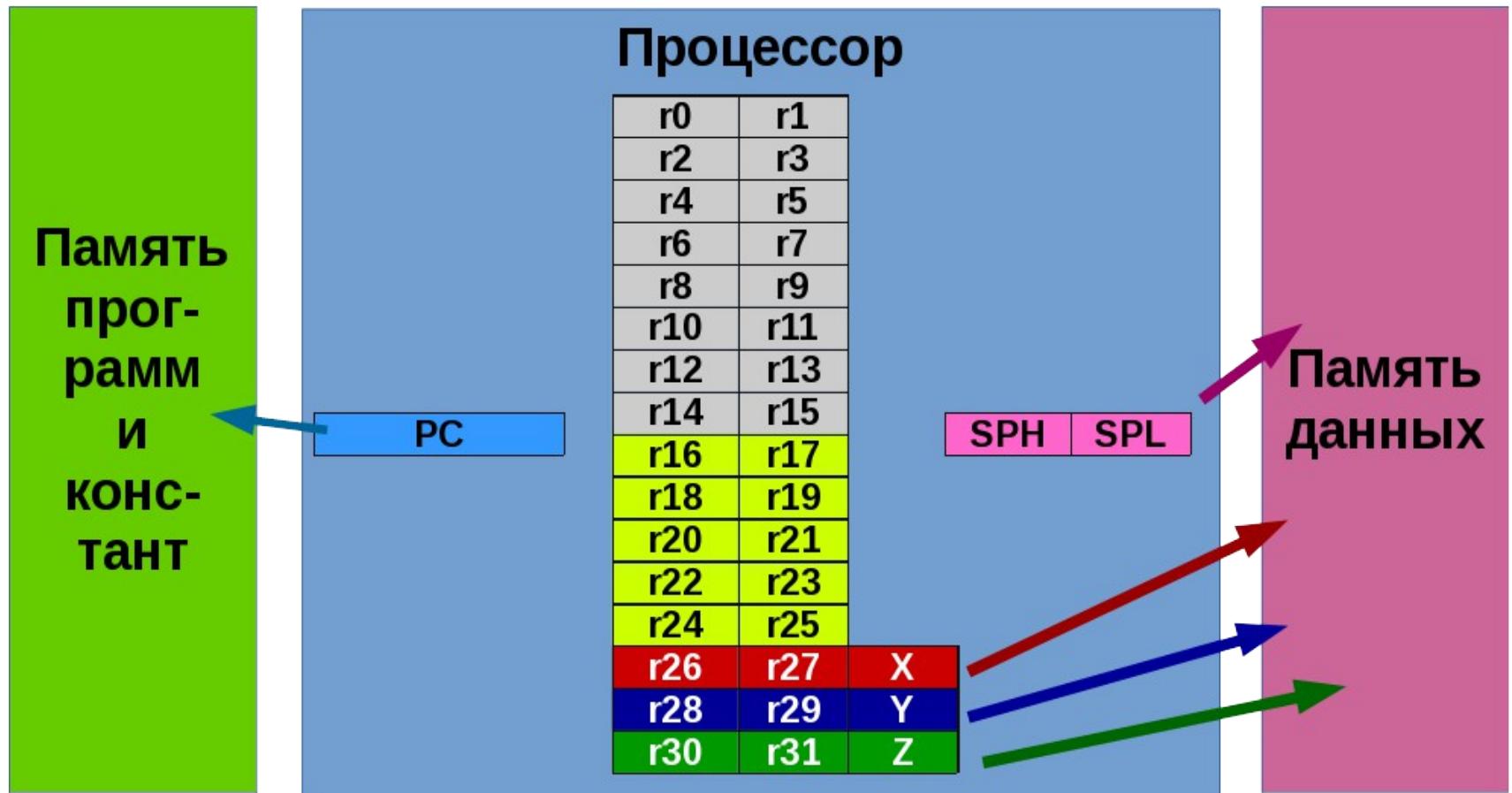
*Найдите 10 отличий на этих картинках*

# Микроконтроллеры AVR

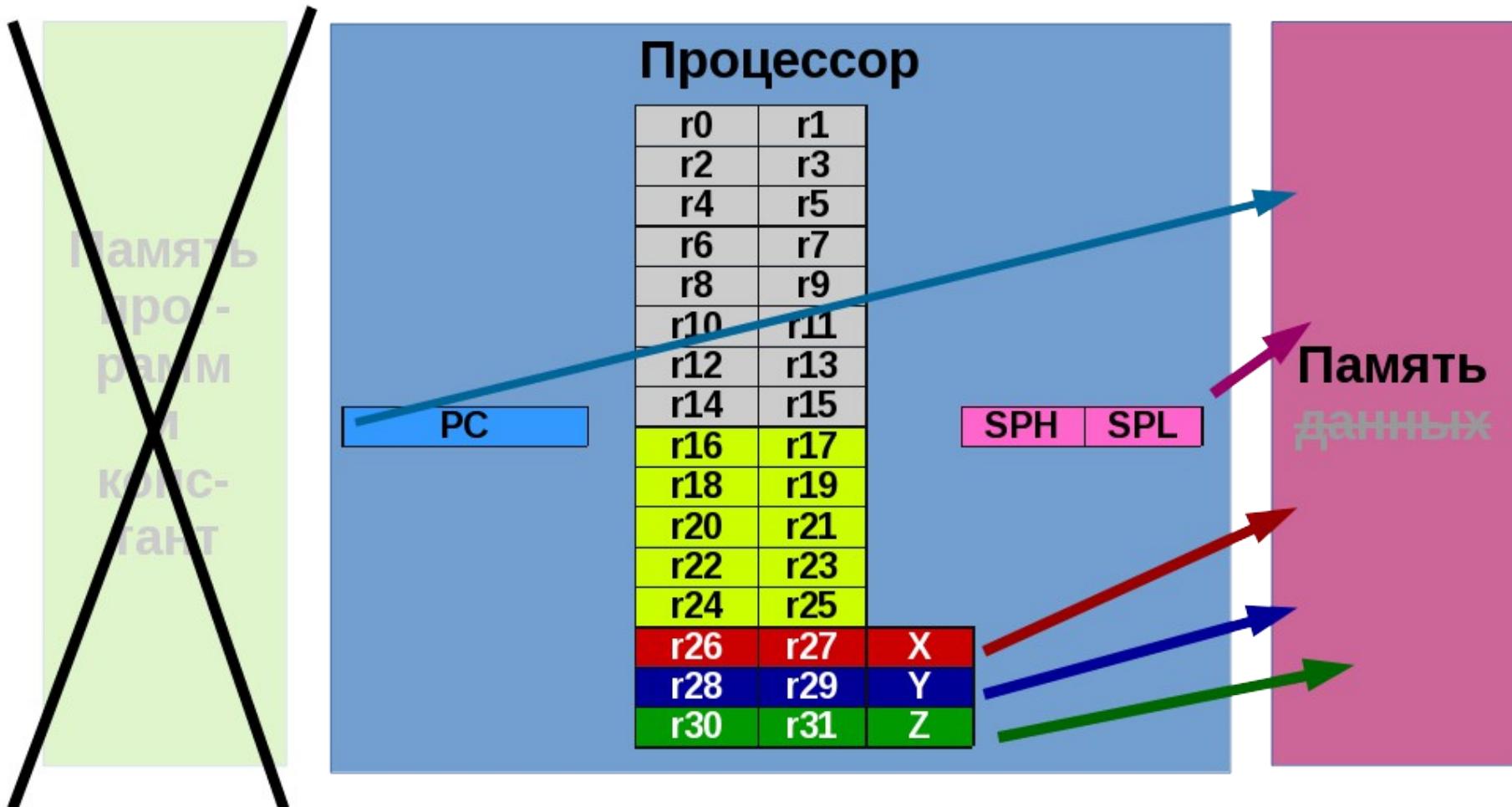
- Гарвардская архитектура,  
а не Фон-Неймана
- 8 бит
- Встроенная память ОЗУ и ПЗУ

*По сути - "система на кристалле"*

# Гарвардская архитектура



# Архитектура Фон-Неймана



# Микроконтроллеры AVR

## Atmel ATtiny13A

Процессор 8-бит

Встроенная SRAM (64 байт)

Встроенная Flash-память (1K)

Встроенная EEPROM (64 байт)

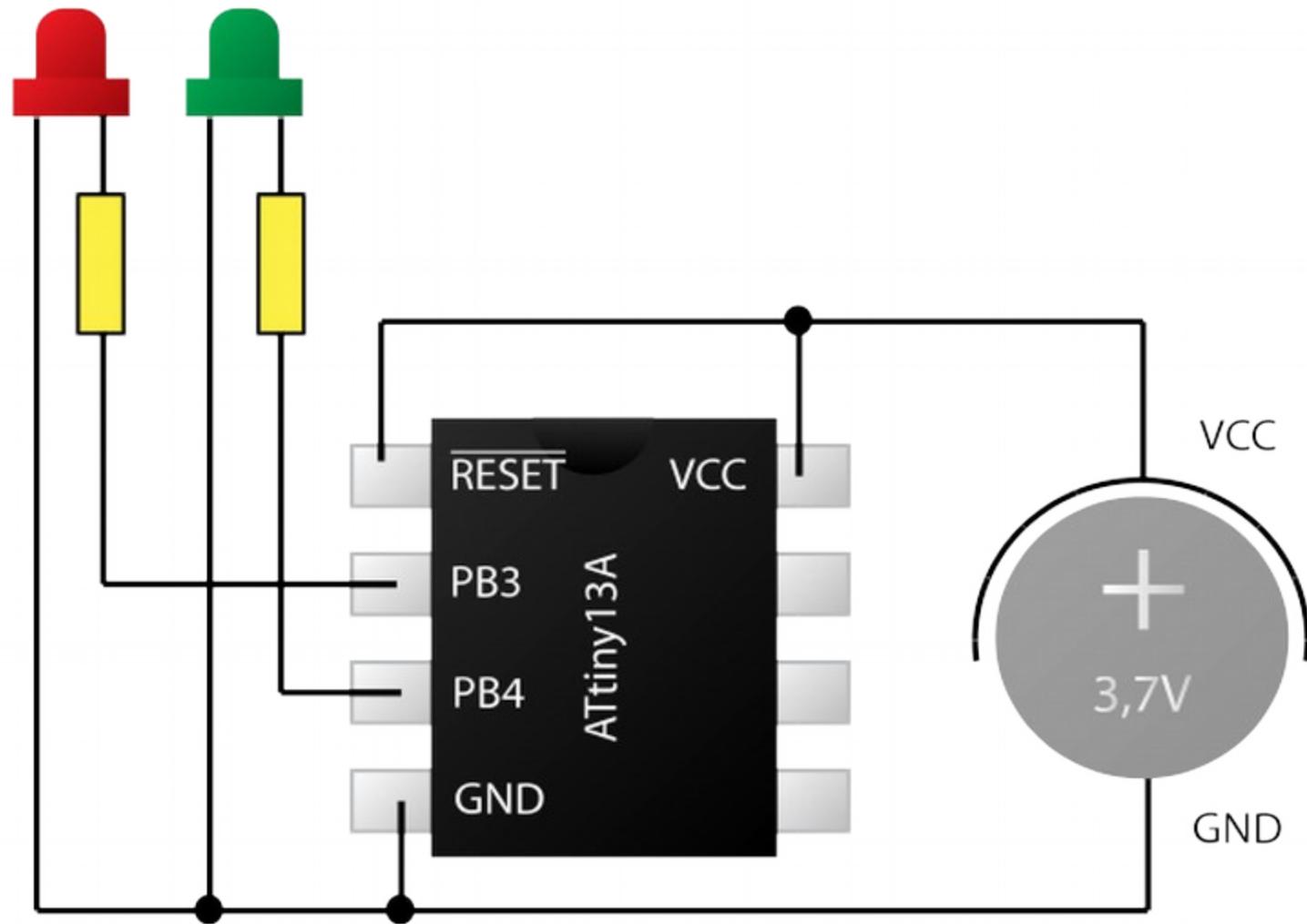
Таймер

АЦП (10 бит)

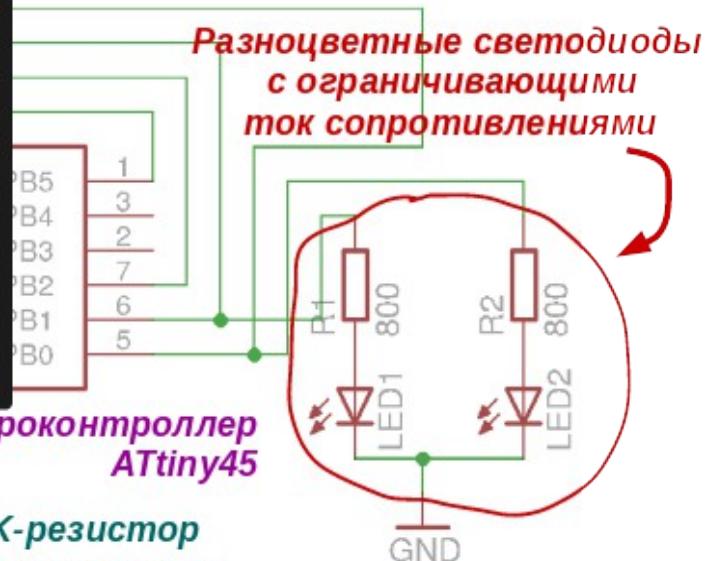
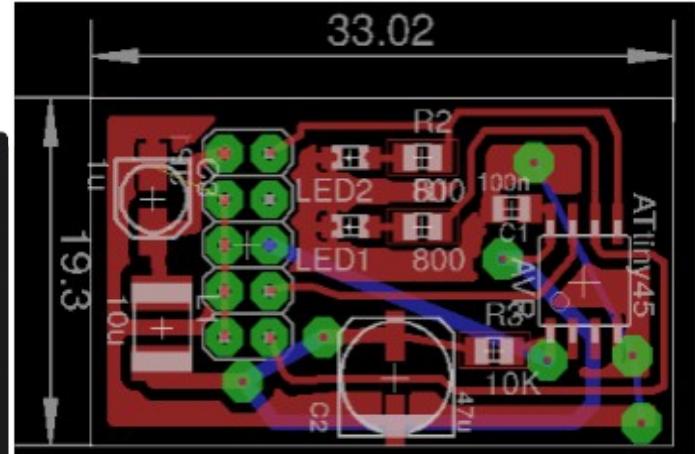
- **Всего 8 выводов у микросхемы**



# "Hello, World!" для МК



# Сделай свой компьютер методом "лазерного утюга"



# Архитектура ARM (32 бит)

- Единый набор команд для разных типов устройств:
  - Cortex-M - микроконтроллеры
  - Cortex-A - процессоры для ПК/смартов/таблеток
  - Cortex-R - промышленные процессоры
- FPU является опциональным, есть не на всех ядрах
- На некоторых ядрах есть SIMD-инструкции
- 13 32-битных регистров общего назначения
- **Инструкции условного выполнения (убрали из архитектуры AArch64)**

# Разработка для ARM

- Кросс-компиляция:  
компилятор, работающий на одной платформе, генерирует код для другой платформы (crosstool-ng, или gcc-arm из пакетной системы дистрибутива)
- Эмуляция:  
эмулируется компьютер целиком (qemu-system) или только набор команд (qemu)

## Почему ARM?

- Наследие x86 уже не так страшко благодаря OpenSource
- Большое количество самых разных производителей
- Низкое энергопотребление

