Процессоры. Часть 1

Лекция №2
АКОС 2019-2020

Немного организационных моментов

OFFTOP

HTTP://EJUDGE64.ATP-FIVT.ORG

Как нагадить в ejudge

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

Как с этим бороться

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

но при этом:

- Снаружи стоит nginx в качестве proxy и ведет подробный лог

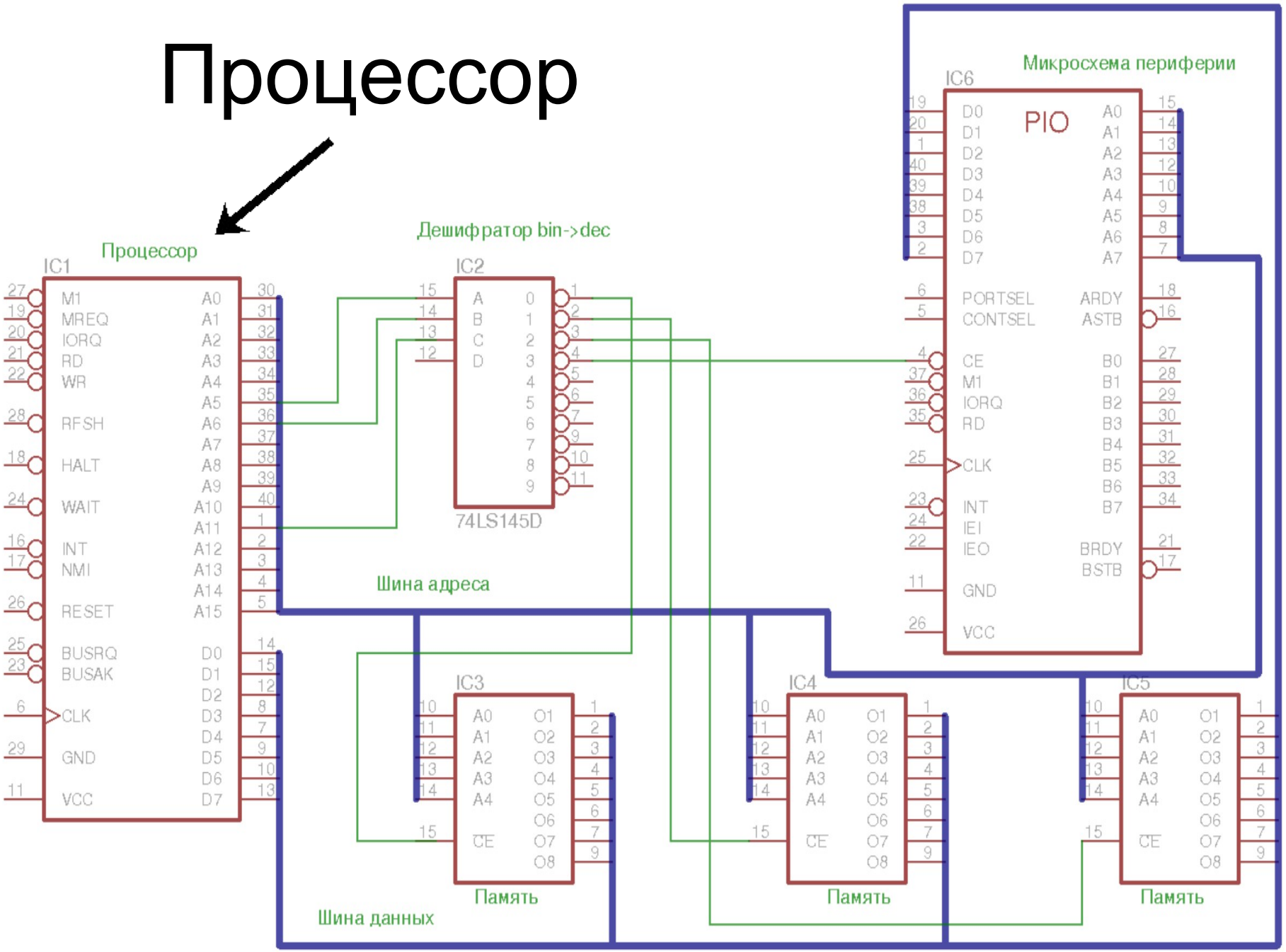
273 УК РФ

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

Процессор


Процессор
IC1
M1
MREQ
IORQ
RD
WR
RFSH
HALT
WAIT
INT
NMI
RESET
BUSRQ
BUSAK
CLK
GND
VCC
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
A15
D0
D1
D2
D3
D4
D5
D6
D7
Дешифратор bin->dec
IC2
74LS145D
Микросхема периферии
IC6
PIO
PORTSEL
CONTSEL
CE
M1
IORQ
RD
CLK
INT
IEI
IEO
GND
VCC
ARDY
ASTB
BRDY
BSTB
Шина адреса
IC3
IC4
IC5
Память
Память
Память
Шина данных

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)

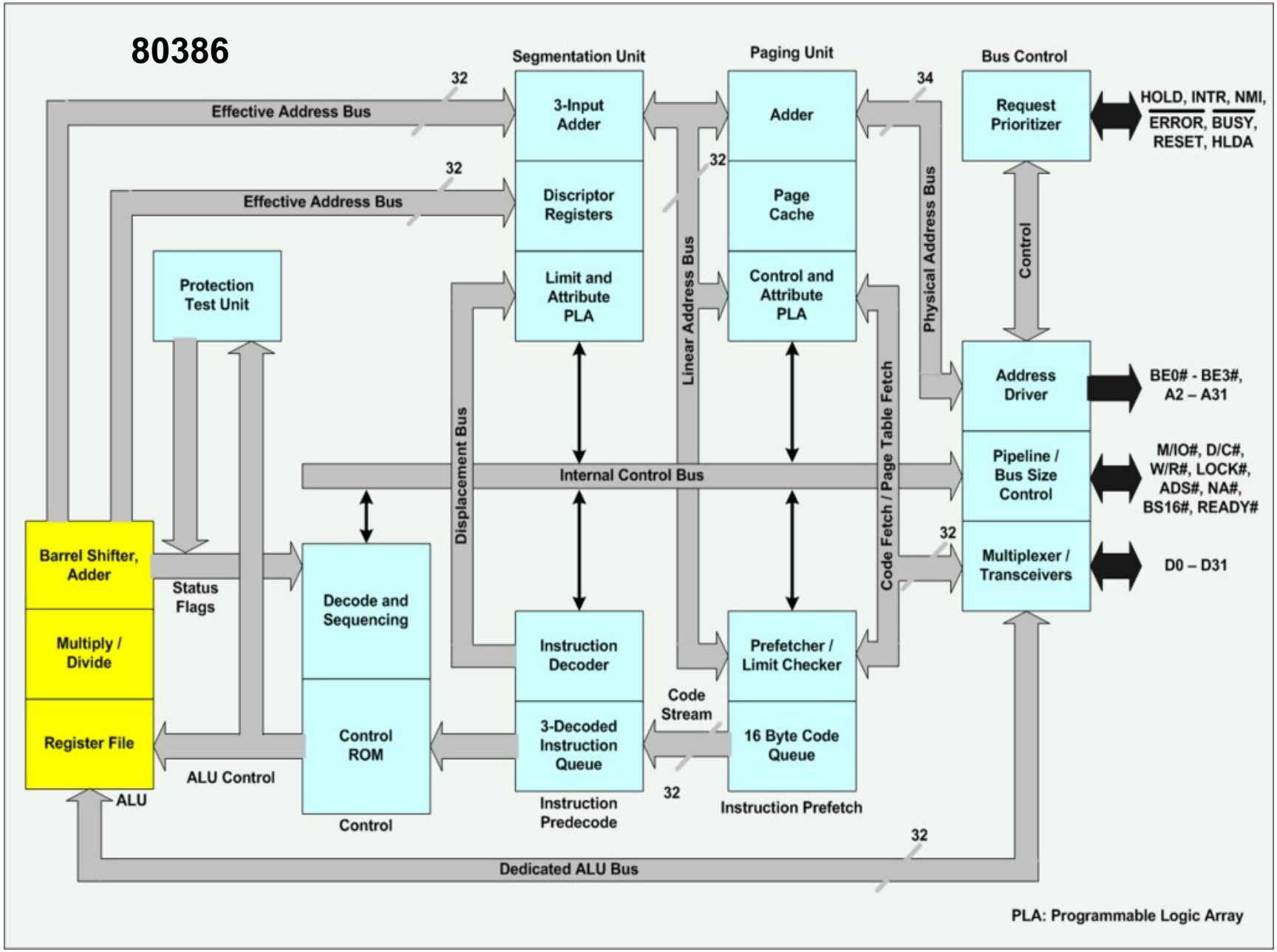
Команды x86

Байты	**0**	**1**	**2**	**3**	**4**	**5**
nop	0x00					
halt	0x10					
rrmovl *rA*, *rB*	0x20	*rA rB*				
irmovl *V*, *r*B	0x30	0x8 *rB*	*V*			
call *Des*t	0x80	*Dest*				
pushl *rA*	0xA0	rA 0x8				
popl *rA*	0xB0	rA 0x8				

Команды кодируются переменным количеством байт

Какие команды выполняет процессор

- Арифметические
- Управляющие


80386
Segmentation Unit
Paging Unit
Bus Control
32
Effective Address Bus
3-Input Adder
Adder
34
Request Prioritizer
HOLD, INTR, NMI,
ERROR, BUSY,
RESET, HLDA
32
Effective Address Bus
Discriptor Registers
32
Page Cache
Physical Address Bus
Control
Protection Test Unit
Limit and Attribute PLA
Linear Address Bus
Control and Attribute PLA
Address Driver
BE0# - BE3#,
A2 – A31
Displacement Bus
Internal Control Bus
Code Fetch / Page Table Fetch
Pipeline / Bus Size Control
M/IO#, D/C#,
W/R#, LOCK#,
ADS#, NA#,
BS16#, READY#
32
Barrel Shifter, Adder
Status Flags
Decode and Sequencing
Multiplexer / Transceivers
D0 – D31
Multiply / Divide
Instruction Decoder
Prefetcher / Limit Checker
Code Stream
Register File
Control ROM
3-Decoded Instruction Queue
16 Byte Code Queue
ALU Control
ALU
Control
Instruction Predecode
32
Instruction Prefetch
32
Dedicated ALU Bus
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

***C**omplex **I**nstruction **S**et **C**omputing (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 - задействуются разные блоки процессора

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

***R**educed **I**nstruction **S**et **C**omputing (RISC)*

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

CISC (x86)

```
080483b4:   55                      pushl %ebp
080483b5:   89 e5                   movl  %ebp, %esp
080483b7:   8b 45 0c                movl  0xc(%ebp), %eax
080483ba:   03 45 08                addl  0x8(%ebp), %eax
080483bd:   01 05 64 94 04 08       addl  0x8049464, %eax
080483c3:   89 ec                   movl  %esp, %ebp
080483c5:   5d                      popl  %ebp
080483c6:   c3                      ret
080483c7:   90                      nop
```

RISC (Atmel AVR)

```
0000: 03 e0          ldi   r16, 0b00000011
0002: 07 bb          out   DDRB, r16
0004: 03 e0          ldi   r16, 0b00000011
0006: 08 bb          out   PORTB, r16
0008: 08 b3          in    r16, PORTB
000a: 00 95          com   r16
000c: 03 70          andi  r16, 0b00000011
000e: 08 bb          out   PORTB, r16
0010: 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-адаптеры, DSP в телевизорах Процессоры 1890ВМ9Я (2 ядра, 1ГГц) и 1907ВМ028 (вмего 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

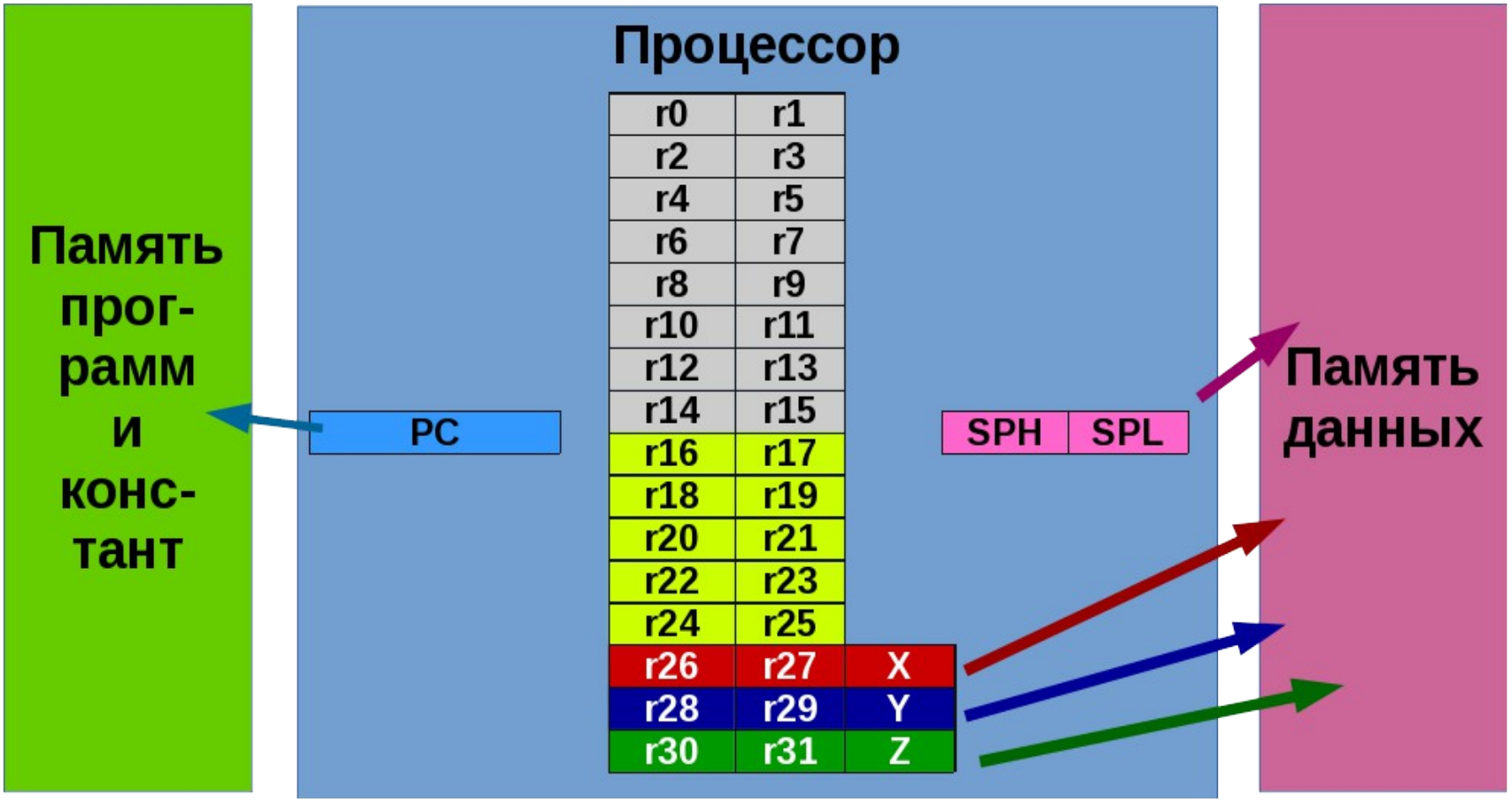
Найдите 10 отличий на этих картинках

Микроконтроллеры AVR

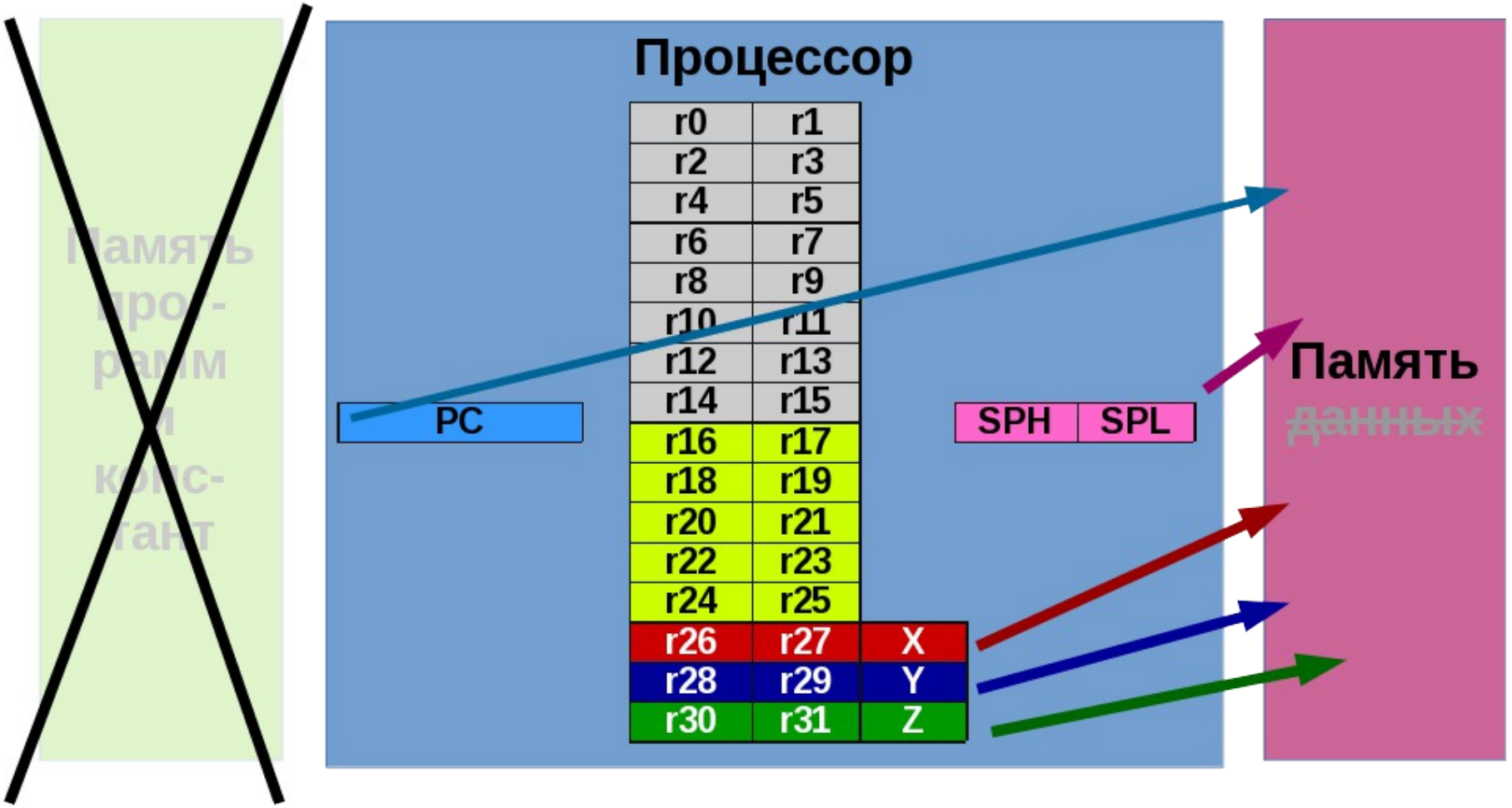
- Гарвардская архитектура,
  а не Фон-Неймана
- 8 бит
- Встроенная память ОЗУ и ПЗУ

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)

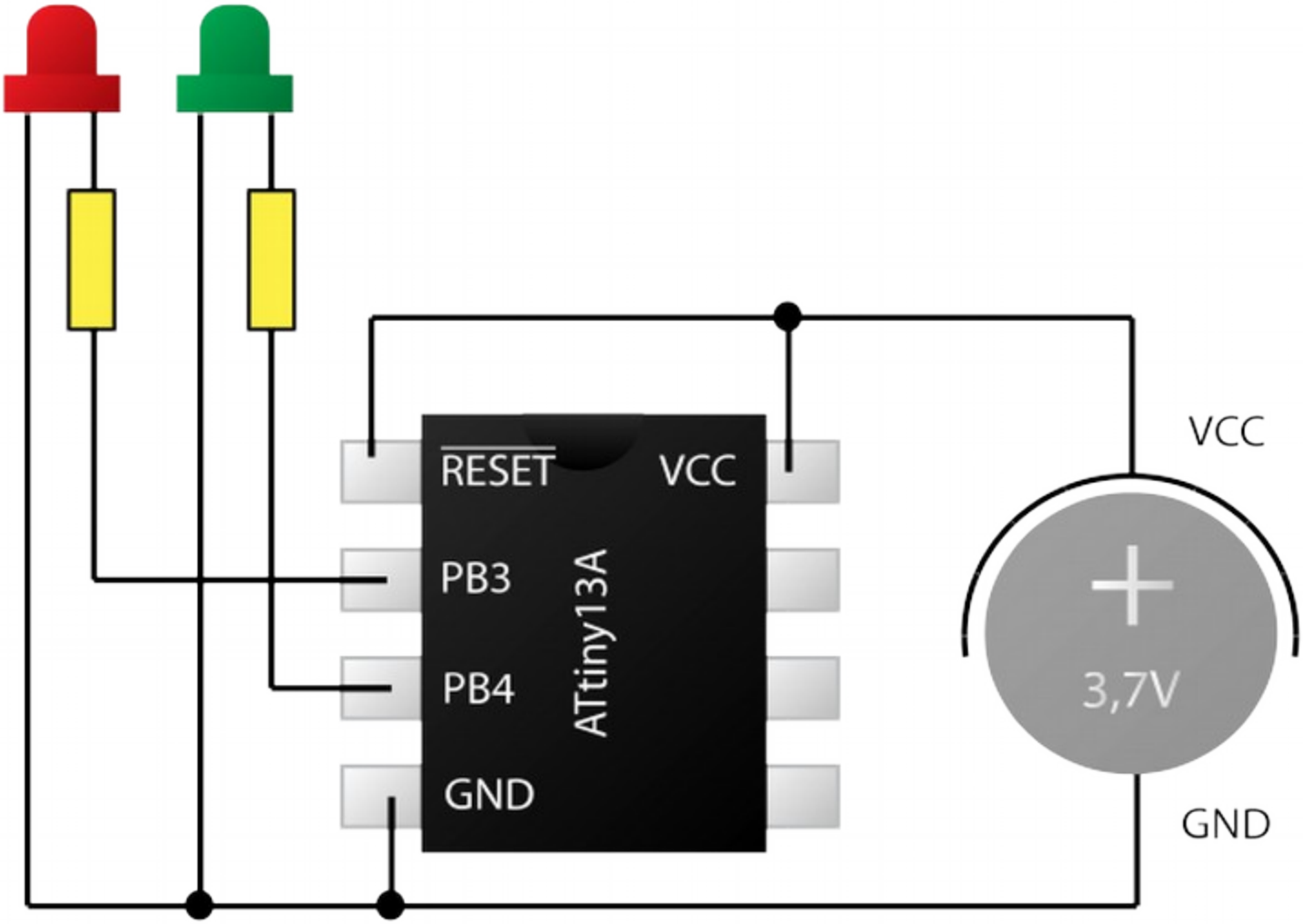
Встроенная EEPROM (64 байт)

Таймер

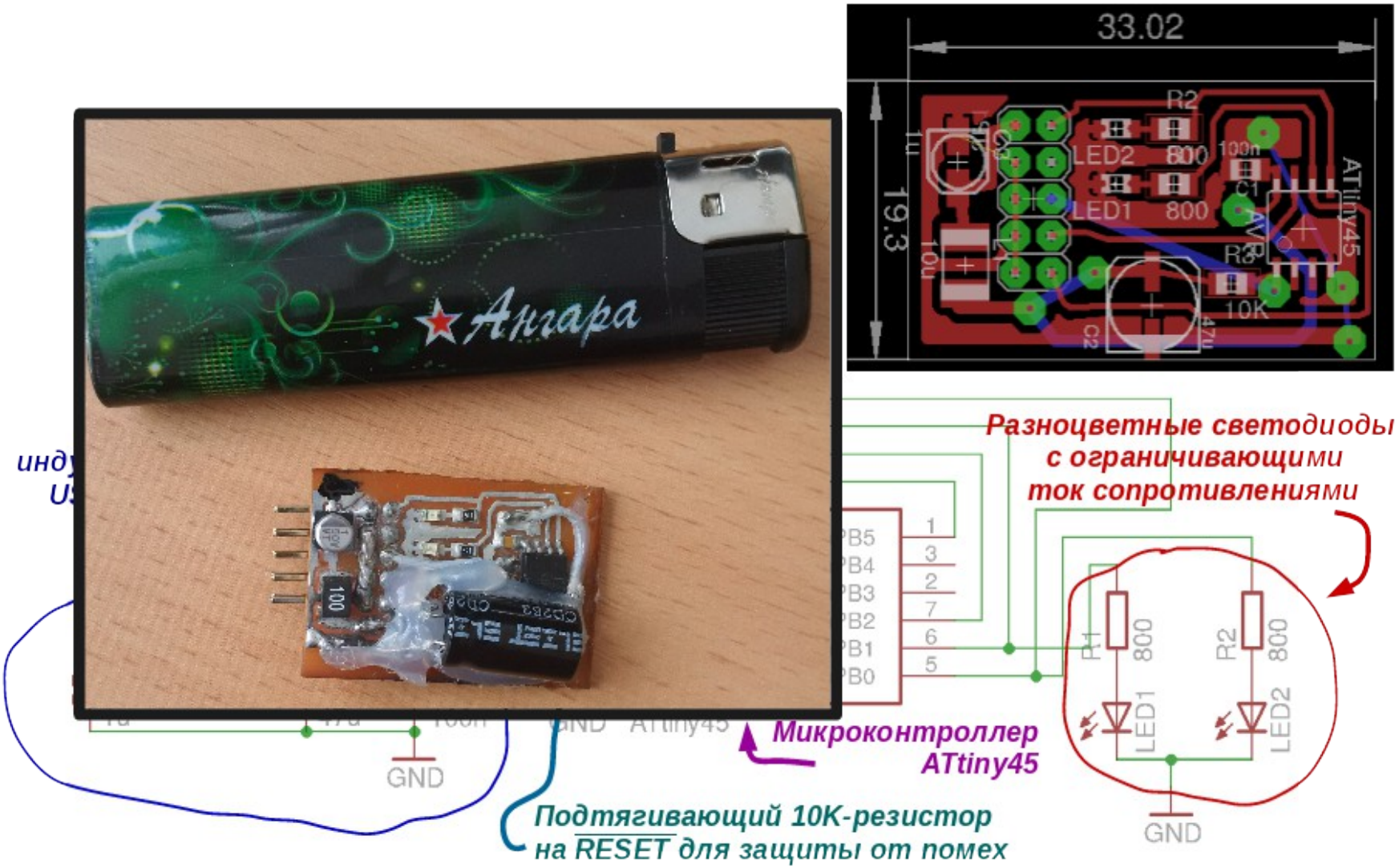
АЦП (10 бит)

- **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)**

Вещественная арифметика

IEEE 754

Представление IEEE754

Single-Precision (32 бит); B = 127		
S	E (8 бит)	M (23 бит)

Double-Precision (64 бит); B = 1023		
S	E (11 бит)	M (52 бит)

$$Value = (-1)^{S} \cdot 2^{E-B} \cdot (1 + M / (2^{23 \text{или} 52} - 1))$$

Специальные значения

S	E	M	Значение
0	0	0	+0
1	0	0	−0
0	11...11	0	+ ∞
1	11...11	0	− ∞
0	11...11	≠ 0	Signaling NaN
1	11...11	≠ 0	Quiet NaN
0	0	≠ 0	Денормализованные значения
1	0	≠ 0	

Денормализованные числа

Single-Precision (32 бит)		
S	0000 0000	M (23 бит)

Double-Precision (64 бит)		
S	000 0000 0000	M (52 бит)

$$Value = (-1)^{S} \cdot M / (2^{23 \text{или} 52} - 1)$$

Операции: умножение

$<S,E,M> = <S_1,E_1,M_1> \cdot <S_1,E_1,M_2>$

1. Вычислить

$S = S_1 \wedge S_2$

$E = E_1+E_2$

$M = 1.M_1 \cdot 1.M_2$

2. { M>>=1 ; E++ }пока переполнение M

Операции: сложение

$<S,E,M> = <S_1,E_1,M_1> \cdot <S_1,E_1,M_2>$

1. Вычислить $E_{diff}=E_1-E_2$
2. Нормализовать M_2 на E_{diff} бит
3. Значения:

   $E = E1$

   $M = M1 +/- M2$

   $S = sign(-1^{S1}M_1 + -1^{S2}M_2)$

Реализации FPU

- Расширенный набор команд (ARM VFP):
  - Дополнительные команды
  - Дополнительные 32 регистра
- Сопроцессор (x86):
  `(gcc -mfpmath=387)`
  - Команды, которые CPU делегирует FPU
  - Взаимодействие через стек
- Команды SSE (Pentium-III+, x86-64):
  `(gcc -msse -mfpmath=sse)`
  - Используются регистры xmm
  - Используются скалярные команды SSE

Точность

Floating Point

- Single Precision - 32bit ≈ 10^{37}
- Double Precision - 64bit ≈ 10^{307}

Fixed Point