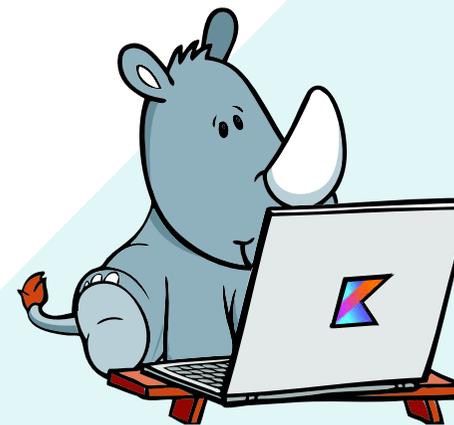


# SMARTRHINO 2019



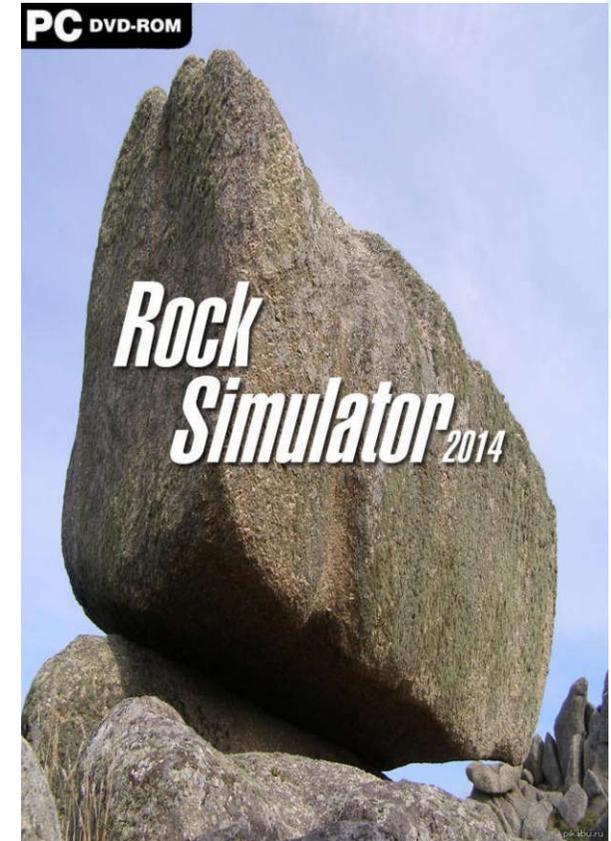
WAIT WHILE EMULATING...

**ОСНОВЫ И ПРИНЦИПЫ  
ПОСТРОЕНИЯ ЭМУЛЯТОРА**

Автор: АЛЕКСЕЙ ГЛАДКИХ

# Что это такое для пользователя?

The  
**SIMS4**



Подражание чему-либо ... например, реальным людям или ...  
камню =)

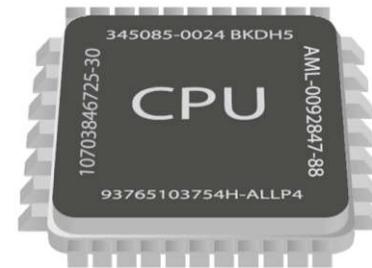
## А для разработчика?

**Эмуляция** (англ. *emulation*) в вычислительной технике — комплекс программных, аппаратных средств или их сочетание, предназначенное для копирования (или *эмулирования*) функций одной вычислительной системы (*гостя*) на другой, отличной от первой, вычислительной системе (*хосте*) таким образом, чтобы эмулированное поведение как можно ближе соответствовало поведению оригинальной системы (*гостя*).



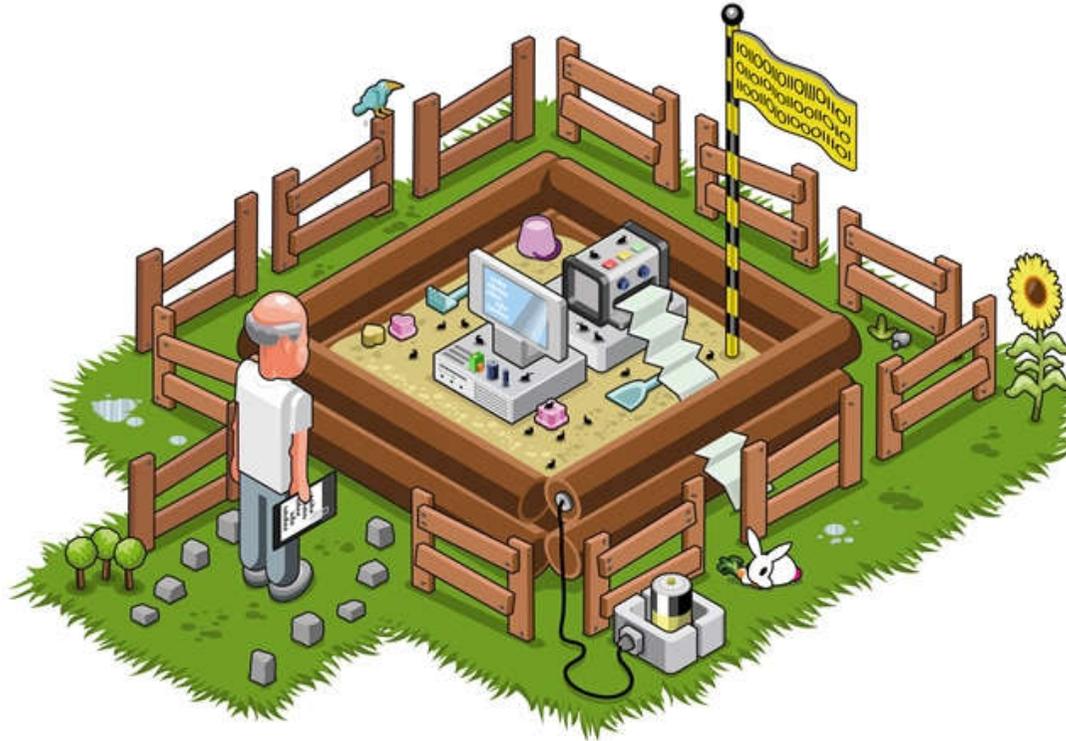
## Для чего это нужно?

- Запуск того, что не поддерживается данной платформой (запуск программ от Windows на Linux, старых видеоигр и т.п.).
- Разработка и отладка прошивок аппаратных систем (BIOS, прошивка будильника, контроллера управления машиной или ядерным реактором).
- Проектирование и отладка (верификация) интегральных микросхем (ЦПУ)
- Изучение и детальное исследование того, что еще не создано или создано слишком давно... 😊 или просто не подходит к друг другу...



## И еще немного философии...

В настоящее время многие сложные технические объекты создаются с помощью моделирования, симуляции и эмуляции...



Человек создает себе песочницу, в которой изучает существующий мир и создает новый...

Пример: высокотехнологичное устройство,  
которое может нанести вред в случае ошибок  
разработки и проектирования



# «Пищевая цепочка» сложного «устройства»

Chevrolet Camaro



Блок управления двигателем



ДВС

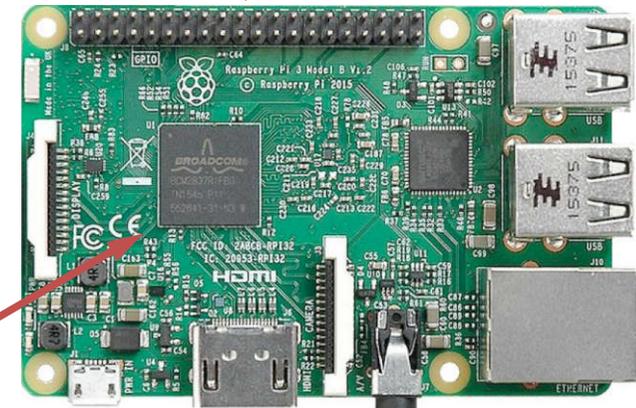
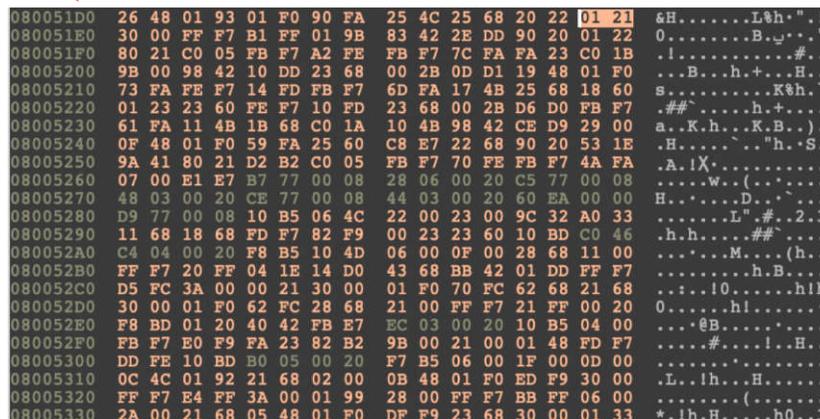


Электрические сигналы

Исходный код на языке Си

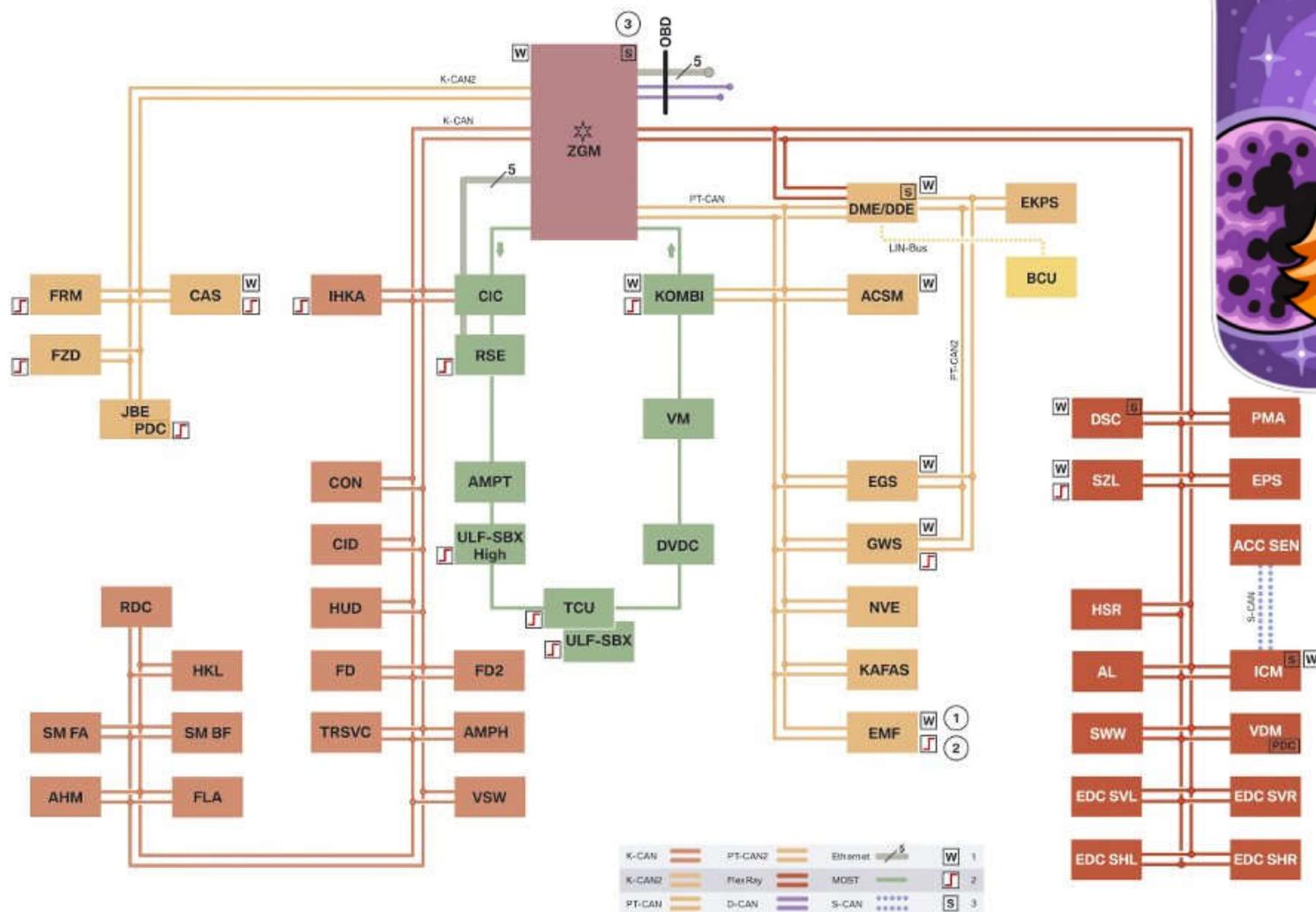
```
for(;;)
{
    int uhtSCAcquisitionValue = tsc_acquisition_value(&htsc, TSC_GROUP2_IDX, 32);
    printf("uhtSCAcquisitionValue = %d\r\n", uhtSCAcquisitionValue);
    if (uhtSCAcquisitionValue < threshold) {
        HAL_GPIO_WritePin(LED_BTN_GPIO_Port, LED_BTN_Pin, GPIO_PIN_SET);
        press_time = HAL_GetTick() - press_start;
        if (press_time > 1000) {
            if (is_auth == 0) {
                printf("AUTH BTN\r\n");
                osThreadSuspendAll();
                auth_start = HAL_GetTick();
                prv_auth = is_auth;
                is_auth = 1;
                osThreadResumeAll();
            }
        } else {
```

Прошивка: что видит ЦПУ



Основная плата блока управления 😊

# «Сложнааа...»



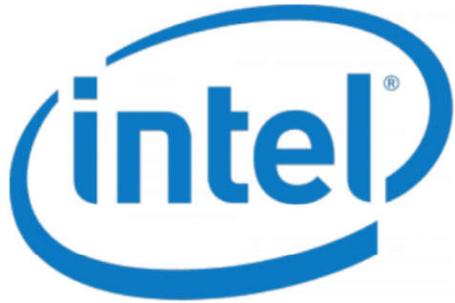
# Что здесь эмулировать?

- Целиком автомобиль
- Двигатель
- **Блок управления двигателем**
- Центральный процессор
- Контроллер USB
- И т.д.

Yes



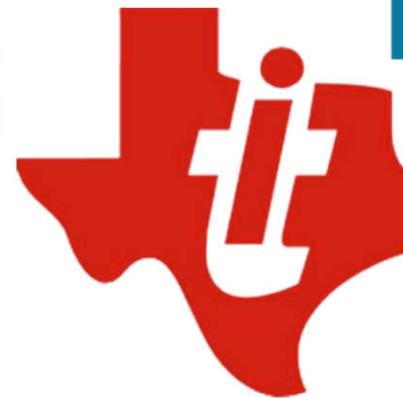
Что может быть внутри этого опасного блока?



**MIPS**  
by Imagination



**RENESAS**  
V850ES



**MSP430**

**PowerPC™**



Для запуска программы от другого процессора на x86 (PC) нужен эмулятор!

По-другому никак... ☹️

# А почему нельзя без эмулятора запустить?

- Разные процессорные архитектуры
- ОС не сможет выполнить код прошивки
  - Специфичные привилегированные инструкции
  - Различные режимы работы процессора
  - Нет прямого доступа к памяти
- Зависимость от периферии



# Возможные варианты моделирования и эмуляции

- Симулятор электрических сигналов (аналоговых или цифровых)

**ModelSim**   
**PROTEUS**

- Эмулятор выполнения инструкций

 **QEMU** 

- Виртуализация

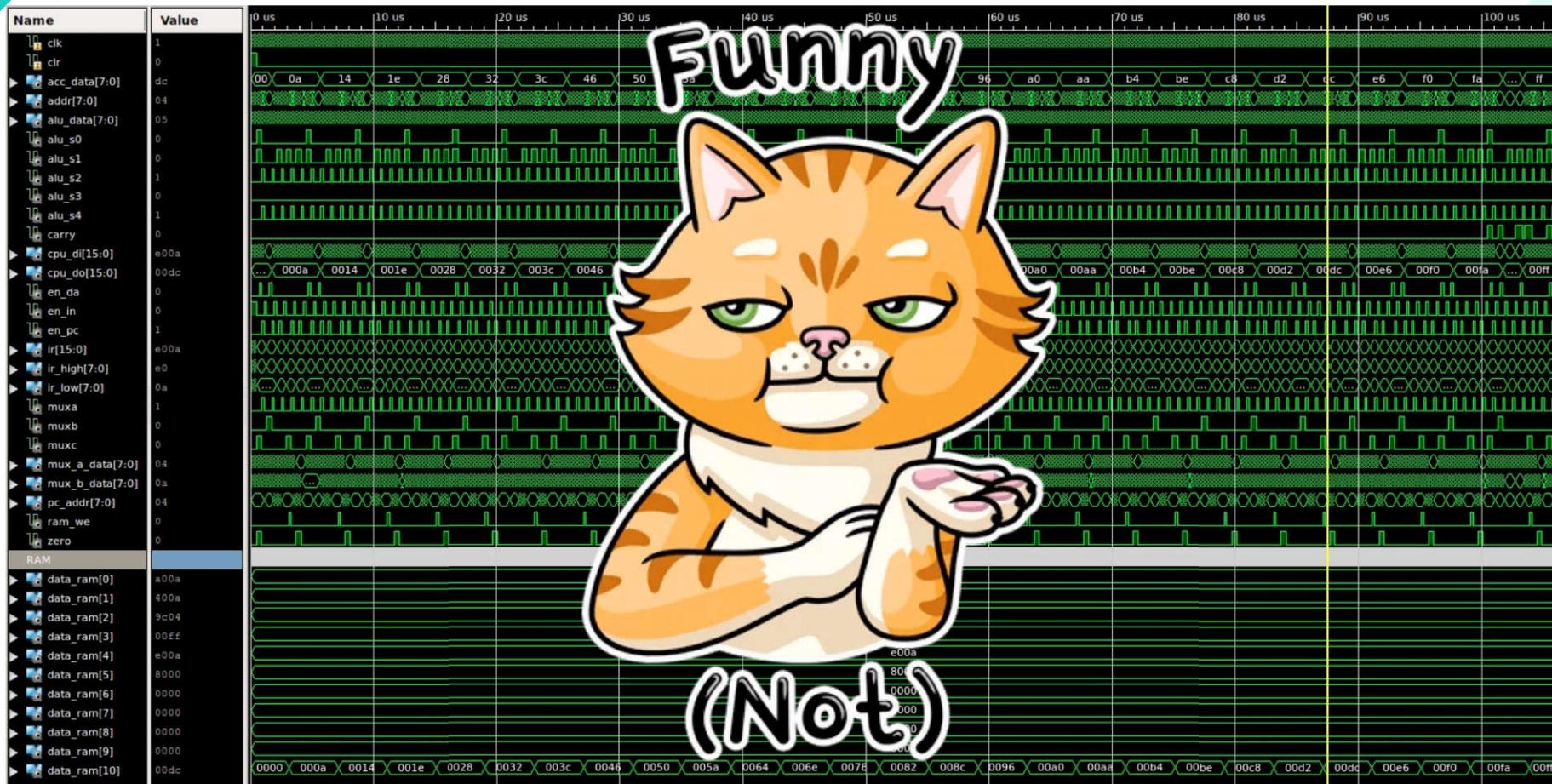
 **VirtualBox** 

SMARTRHINO  
2019

Детализация

Скорость

# Симулятор: моделируем всё, сразу и долго!



- Применимо для разработки микросхем: ЦПУ и ASIC (специального применения и назначения).
- Бессмысленно для разработки прошивки.

# Эмуляция: только инструкции и транзакции!

```
080051D0 26 48 01 93 01 F0 90 FA 25 4C 25 68 20 22 01 21
080051E0 30 00 FF F7 B1 FF 01 9B 83 42 2E DD 90 20 01 22
080051F0 80 21 C0 05 FB F7 A2 FE FB F7 7C FA FA 23 C0 1B
08005200 9B 00 98 42 10 DD 23 68 00 2B 0D D1 19 48 01 F0
08005210 73 FA FE F7 14 FD FB F7 6D FA 17 4B 25 68 18 60
08005220 01 23 23 60 FE F7 10 FD 23 68 00 2B D6 D0 FB F7
08005230 61 FA 11 4B 1B 68 C0 1A 10 4B 98 42 CE D9 29 00
08005240 0F 48 01 F0 59 FA 25 60 C8 E7 22 68 90 20 53 1E
08005250 9A 41 80 21 D2 B2 C0 05 FB F7 70 FE FB F7 4A FA
08005260 07 00 E1 E7 B7 77 00 08 28 06 00 20 C5 77 00 08
08005270 48 03 00 20 CE 77 00 08 44 03 00 20 60 EA 00 00
08005280 D9 77 00 08 10 B5 06 4C 22 00 23 00 9C 32 A0 33
08005290 11 68 18 68 FD F7 82 F9 00 23 23 60 10 BD C0 46
080052A0 C4 04 00 20 F8 B5 10 4D 06 00 0F 00 28 68 11 00
080052B0 FF F7 20 FF 04 1E 14 D0 43 68 BB 42 01 DD FF F7
080052C0 D5 FC 3A 00 00 21 30 00 01 F0 70 FC 62 68 21 68
080052D0 30 00 01 F0 62 FC 28 68 21 00 FF F7 21 FF 00 20
080052E0 F8 BD 01 20 40 42 FB E7 EC 03 00 20 10 B5 04 00
080052F0 FB F7 E0 F9 FA 23 82 B2 9B 00 21 00 01 48 FD F7
08005300 DD FE 10 BD B0 05 00 20 F7 B5 06 00 1F 00 0D 00
08005310 0C 4C 01 92 21 68 02 00 0B 48 01 F0 ED F9 30 00
08005320 FF F7 E4 FF 3A 00 01 99 28 00 FF F7 BB FF 06 00
08005330 2A 00 21 68 05 48 01 F0 DF F9 23 68 30 00 01 33
08005340 23 60 FE BD 4C 03 00 20 E0 75 00 08 ED 75 00 08
08005350 70 B5 05 00 0C 00 01 20 04 21 01 F0 CD FB 00 23
08005360 05 60 C4 60 43 60 84 60 70 BD 70 B5 04 00 A3 68
08005370 08 00 15 00 93 42 0D DB 23 68 62 68 99 18 2A 00
08005380 01 F0 0B FC 00 20 63 68 5B 19 63 60 A3 68 5D 1B
08005390 A5 60 70 BD 01 20 40 42 FB E7 70 B5 06 00 0D 00
080053A0 48 1C 01 21 01 F0 A8 FB 2A 00 04 00 01 00 30 00
```

Декодирование и исполнение инструкции за инструкцией  
Прямо как процессор!

```
080051DC CODE32
080051DC loc_80051DC: count
080051DC press_start = R7; int
080051DC MOVS R2, #0x20 ;
080051DE CODE16
080051DE MOVS R1, #1; group_idx
080051E0 MOVS R0, R6; htsc
080051E2 BL tsc_acquisition_value
080051E6 uhTSCAcquisitionValue = R0; int
080051E6 LDR R3, [SP,#0x20+threshold]
080051E8 CODE32
080051E8 CMP R3, uhTSCAcquisitionValue
080051EA CODE16
080051EA BLE loc_800524A

080051ED MOVS uhTSCAcquisitionValue, #0x90
080051EE MOVS R2, #1; PinState
080051F0 MOVS R1, #0x80; GPIO_Pin
080051F2 LSLS R0, R0, #0x17
080051F4 BL HAL_GPIO_WritePin
080051F8 BL HAL_GetTick
080051FC MOVS R3, #0xFA
080051FE SUBS R0, R0, press_start
08005200 press_time = R0; int
08005200 LSLS R3, R3, #2
08005202 CMP press_time, R3
08005204 BLE loc_8005228

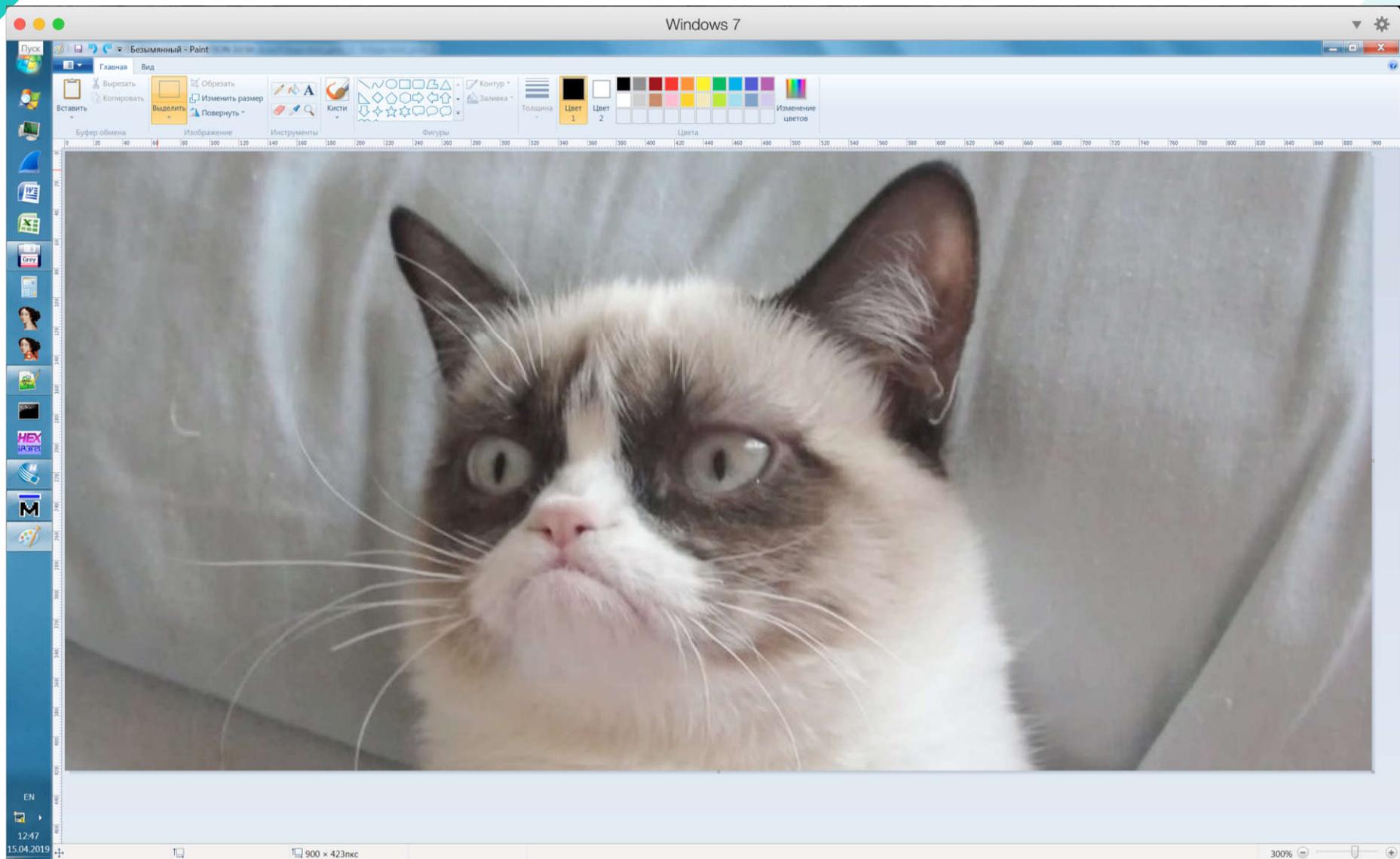
08005206 LDR R3, [R4]
08005208 CMP R3, #0
0800520A BNE loc_8005228

0800524A loc_800524A
0800524A uhTSCAcquisitionValue = R0; int
0800524C LDR R2, [R4]
0800524C MOVS uhTSCAcquisitionValue, #0x90
0800524E SUBS R3, R2, #1
08005250 SBCS R2, R3
08005252 MOVS R1, #0x80; GPIO_Pin
08005254 UXTB R2, R2
08005256 LSLS R0, R0, #0x17
08005258 BL HAL_GPIO_WritePin
0800525C BL HAL_GetTick
08005260 MOVS press_start, R0
08005262 press_start = R0; int
08005262 B loc_8005228
; End of function sensor_task_entry
08005262

0800520C LDR press_time, =aAuthBtn, "AUTH BTN\r\n"
0800520E BL tfp_printf
08005212 BL osThreadSuspendAll
08005216 BL HAL_GetTick
0800521A LDR R3, =auth_start
0800521C LDR R0, [R3]
0800521E STR R0, [R3]
08005220 MOVS R3, #1
08005222 STR R3, [R4]
08005224 BL osThreadResumeAll
```

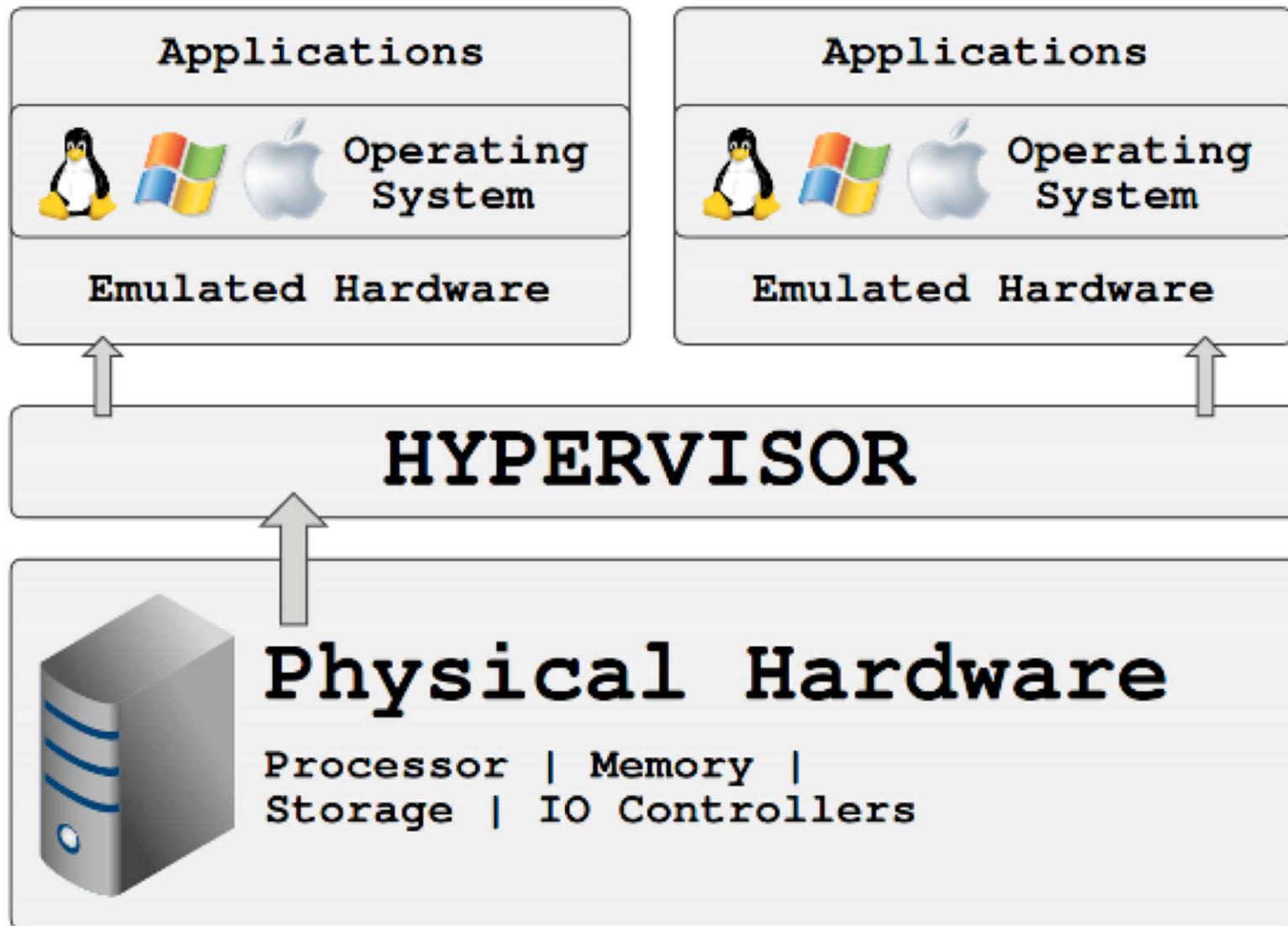
- Разработка и отладка прошивок.
- Анализ поведения устройства
- Бессмысленно для кросс-запуска приложений (медленноооо...)

# Виртуализация: только если мы одинаковые...



\*Это 7-я винда, запущенная на маке

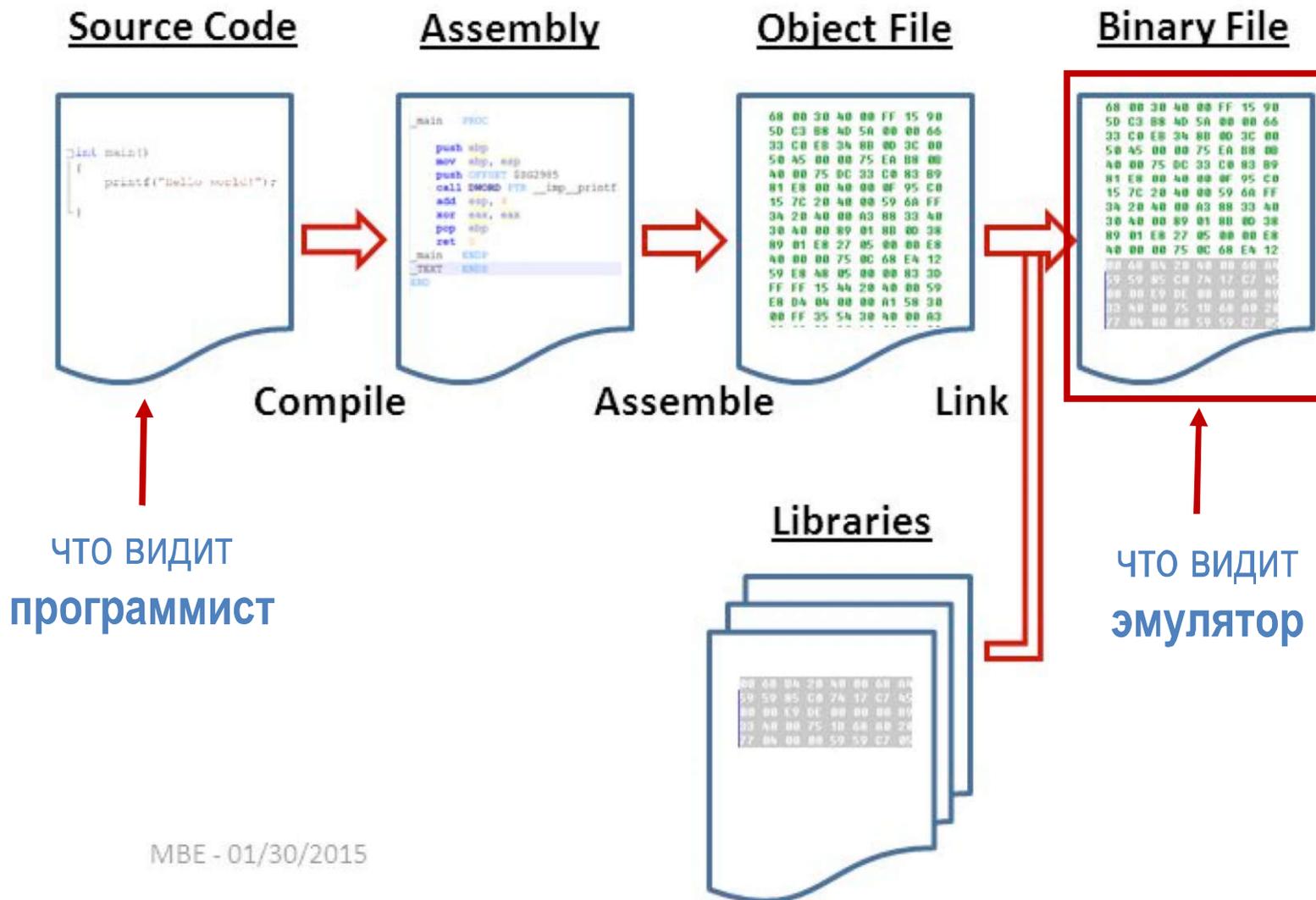
# Всё сложно...



## И так эмулятор... Что же нам нужно?

1. **Софт:** прошивка
2. **Железо:** эмулятор ядра
3. **Железо:** эмулятор периферийных модулей
4. Как собрать это все вместе...
5. **Отладчик...**

# Что у нас на входе, чтобы эмулировать?



MBE - 01/30/2015

# А что по железной части?

Ввод/вывод (управление устройствами)

ARM SOC  
(System-On-Chip)

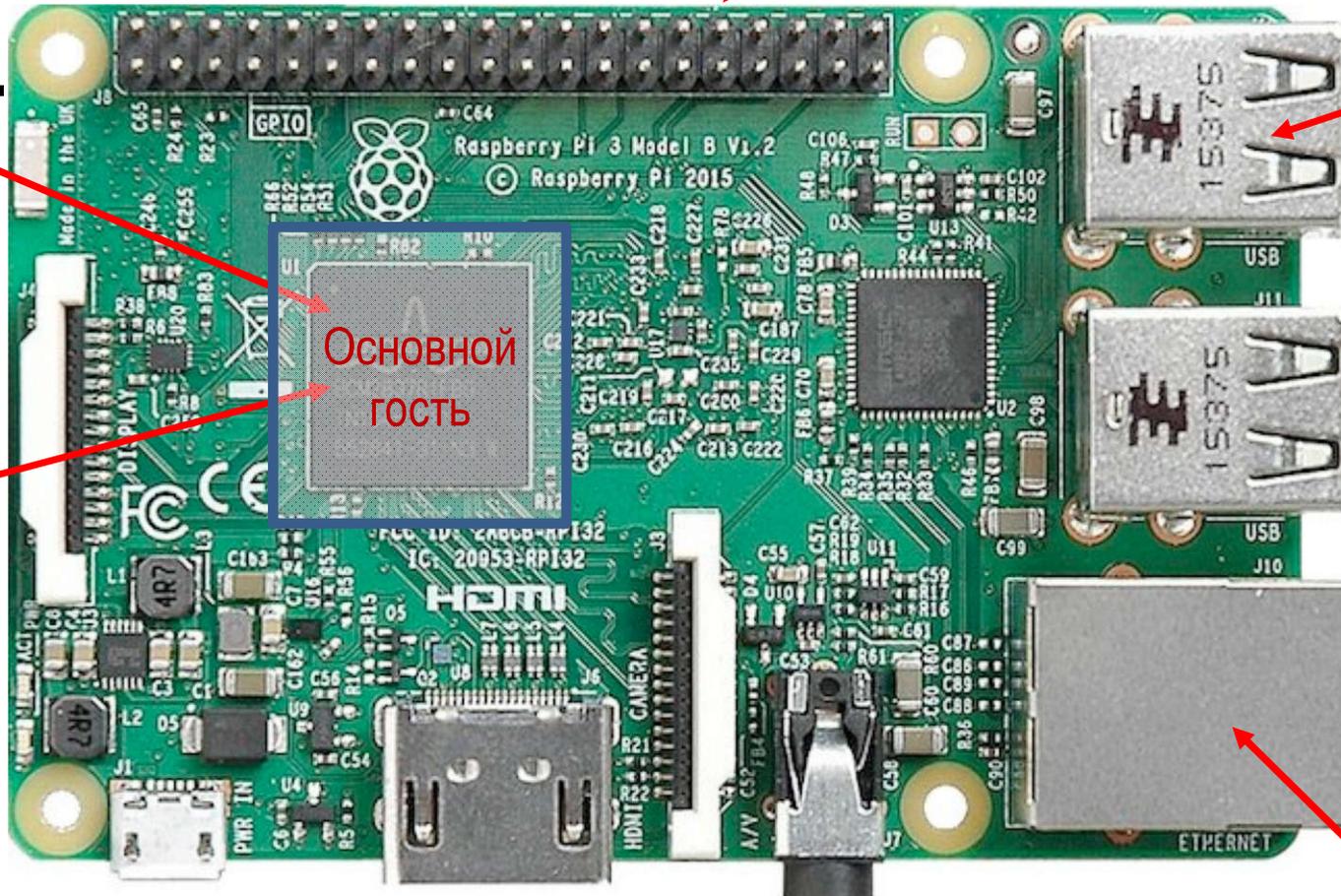
RAM

Основной  
гость

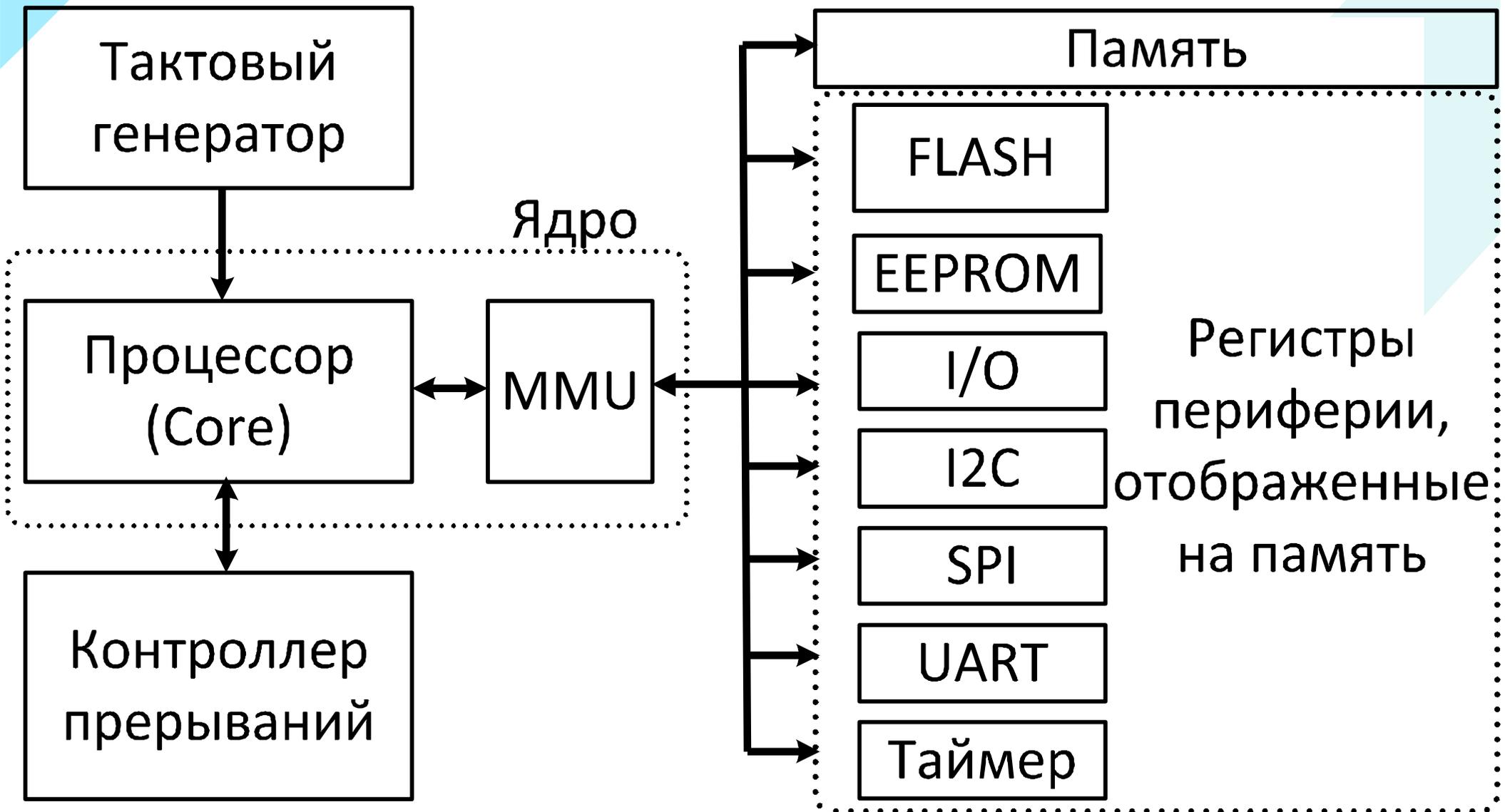
USB

Светодиоды

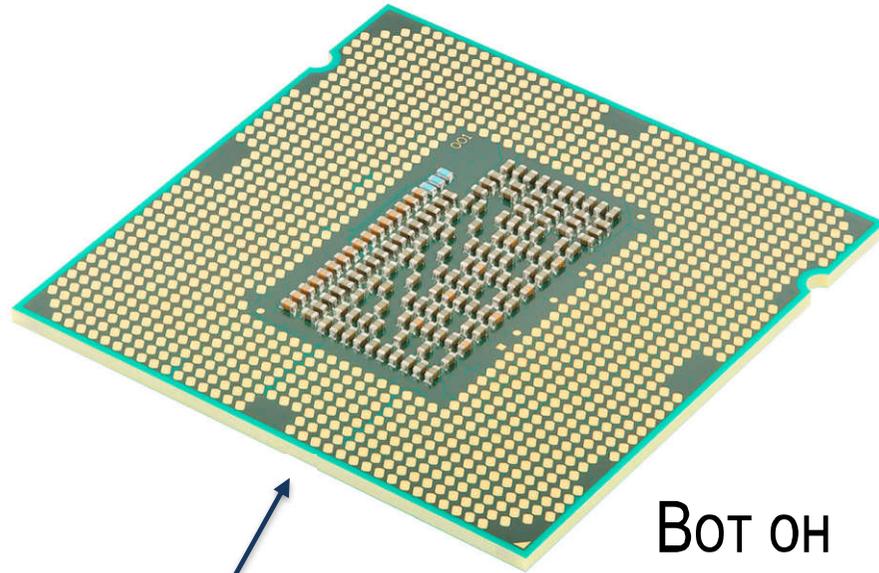
Ethernet



# Типовая схема устройства

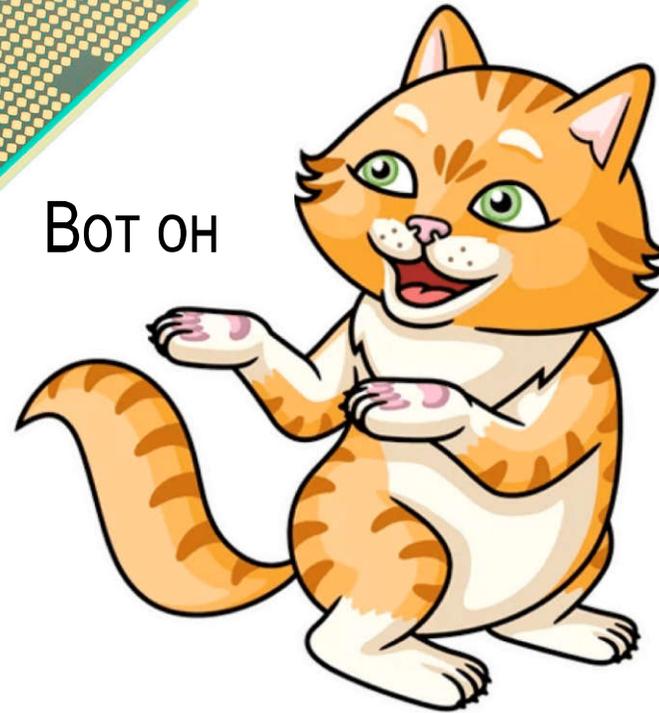


# Кто у нас основной гость или что такое ЦПУ?



Это процессор

Вот он

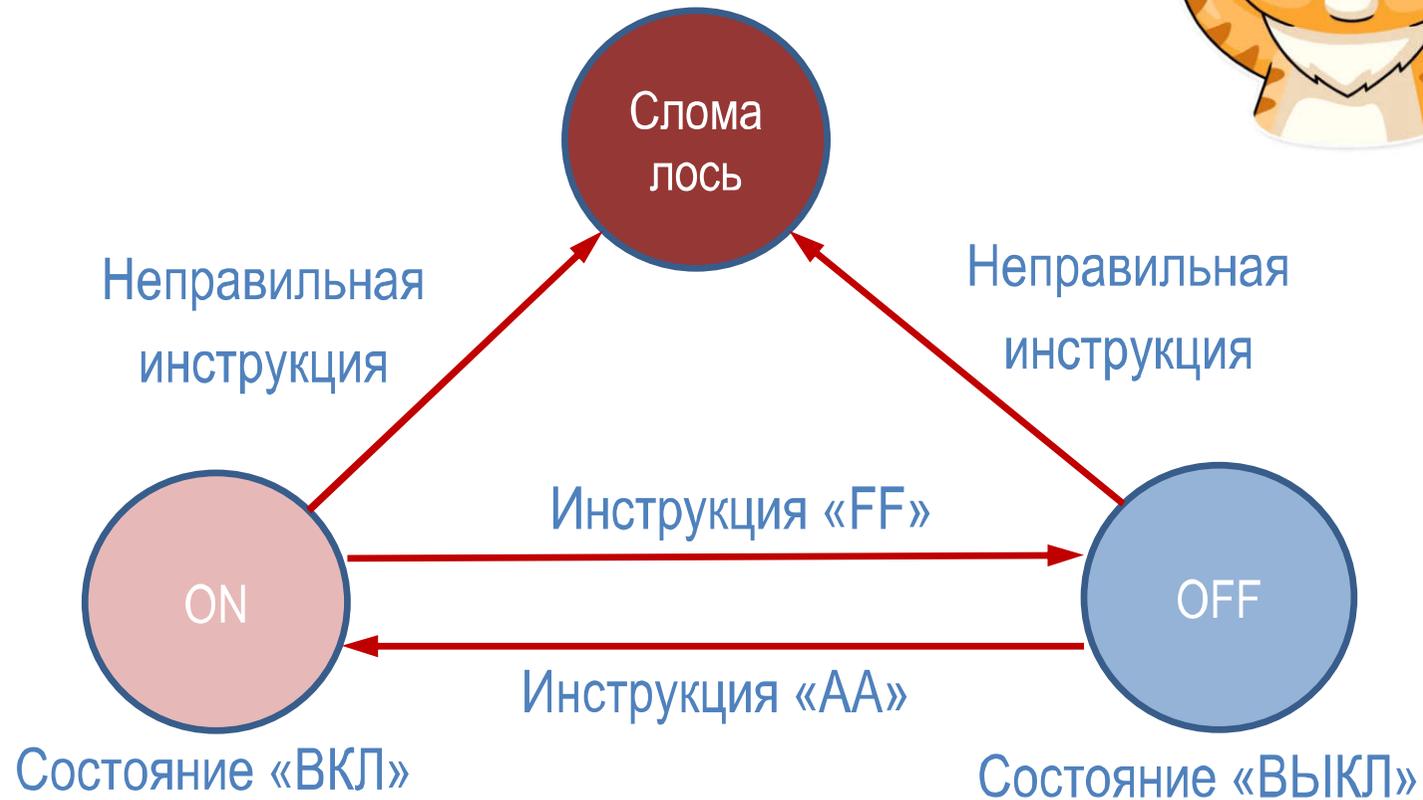


# А что внутри?... Машина состояний

LOVE

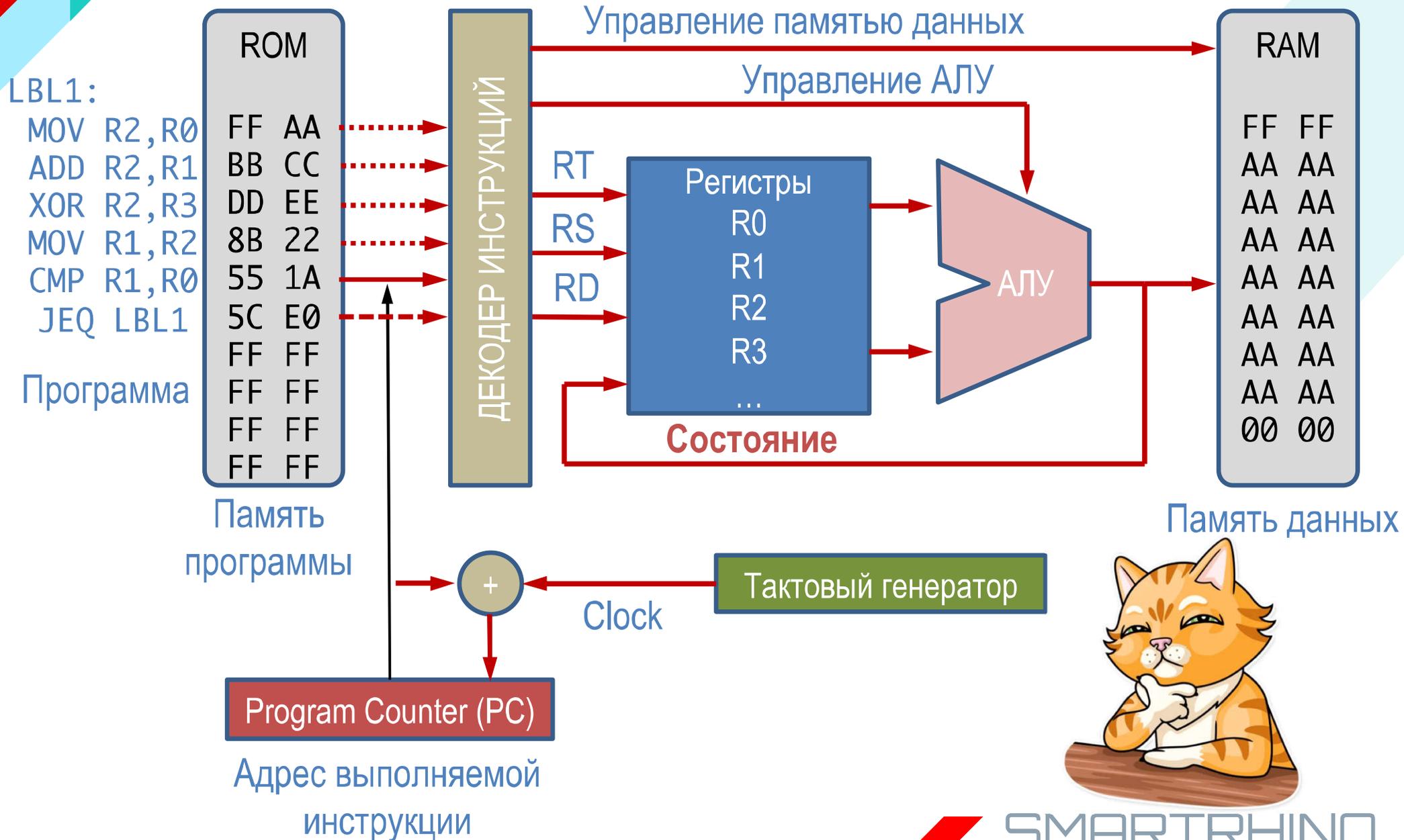


Состояние «СЛОМ» 😊

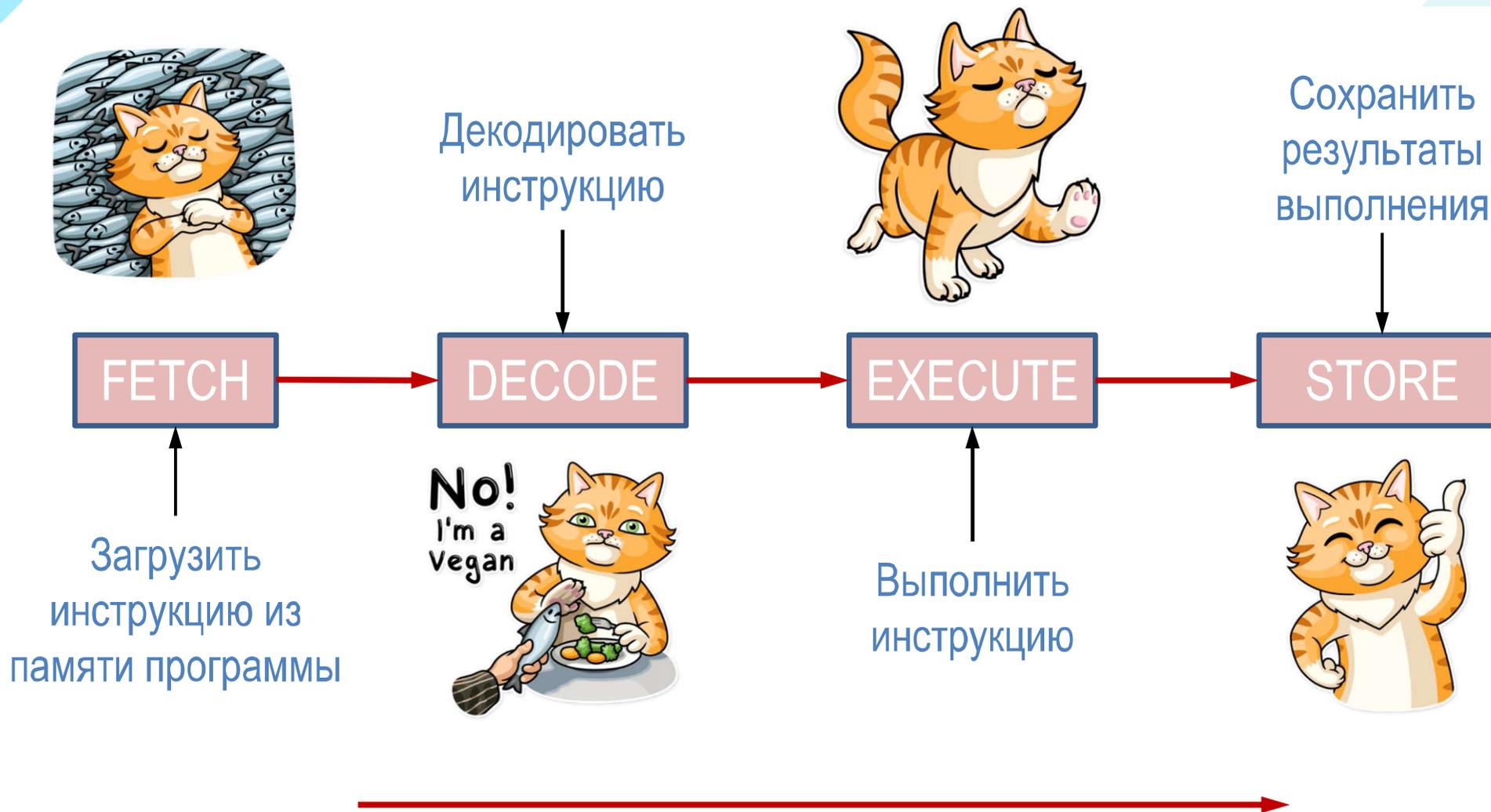


Машина состояний (Finite-State Machine, FSM)

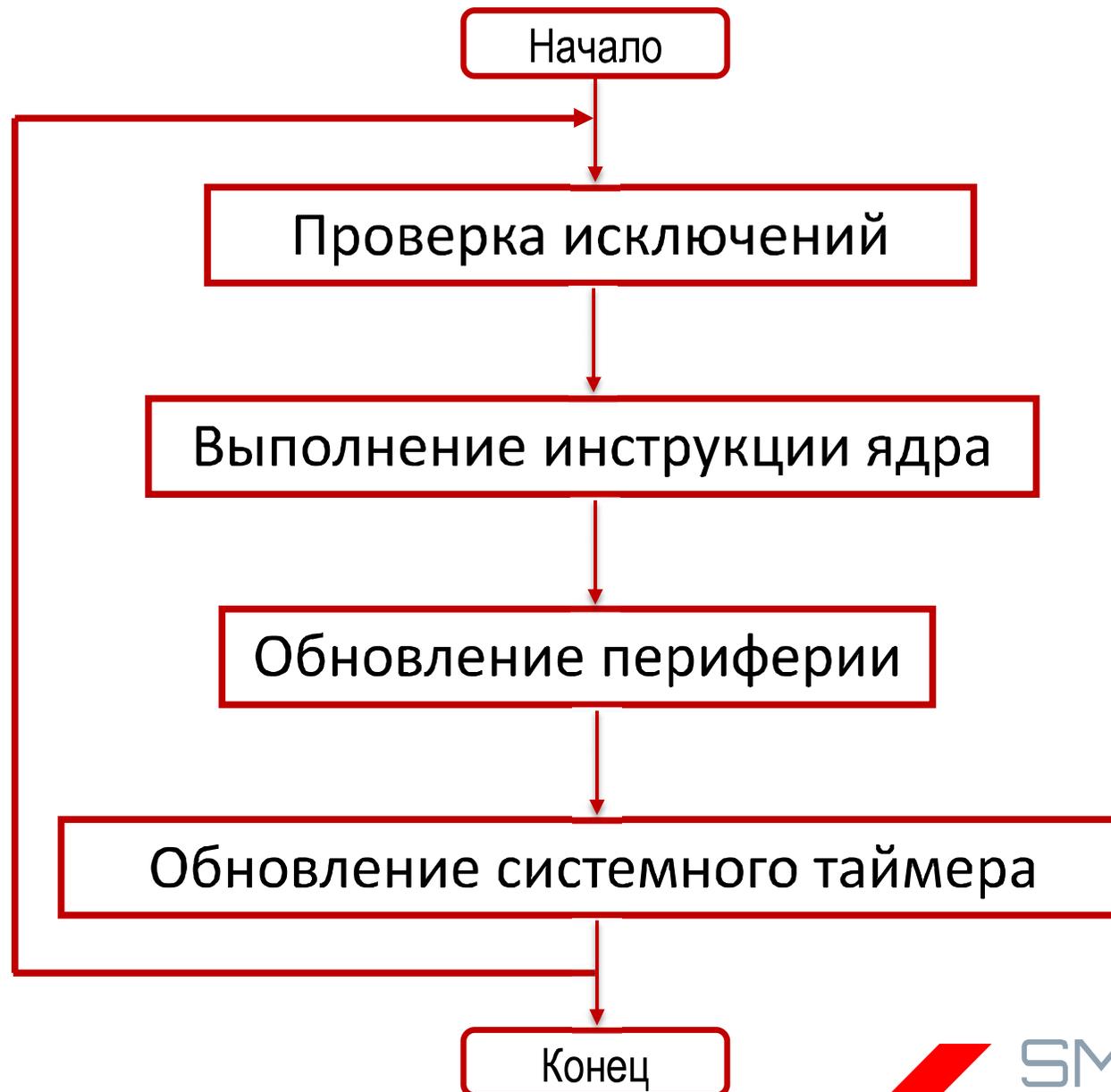
# Прям так просто?... Нет



# А как работает?

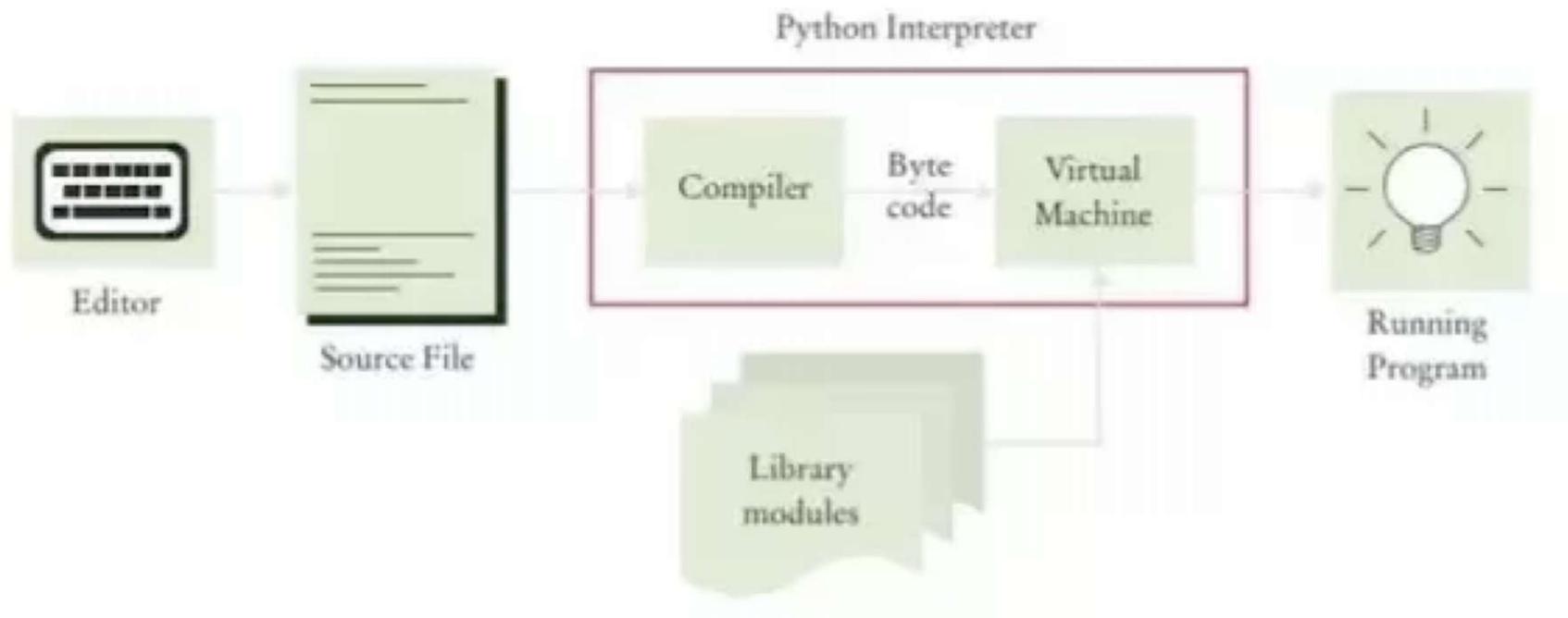


# Как работает эмулятор: основной цикл



# Как работает эмулятор... интерпретатор

## How The Python Interpreter Works



# Выполнение инструкций

Что делает «процессор» за один шаг выполнения?

- Извлечение байтов инструкции из памяти
- Декодирование инструкции
- Выполнение инструкции

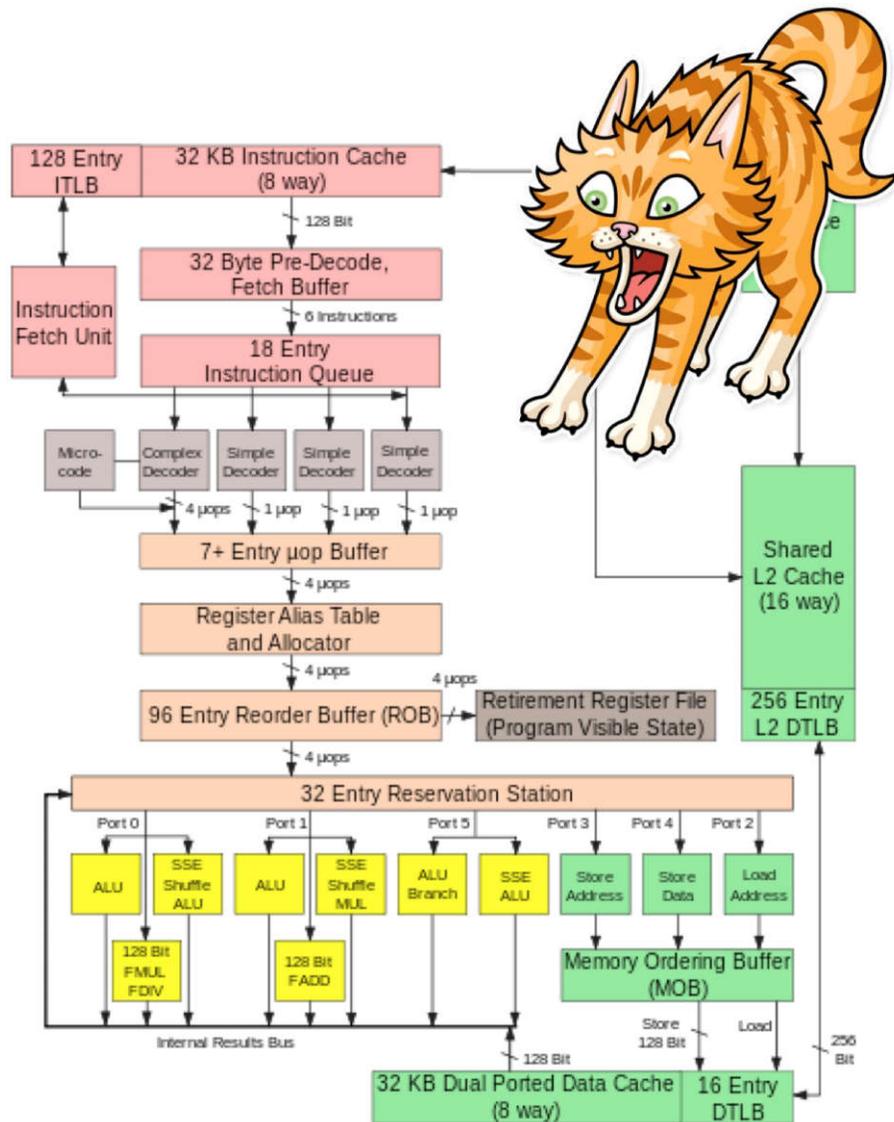
# Блок схема процессора



Intel® 64 and IA-32 Architectures  
Software Developer's Manual

Volume 2A:  
Instruction Set Reference, A-L

NOTE: The Intel® 64 and IA-32 Architectures Software Developer's Manual consists of nine volumes:  
Block Architecture (Order Number 253645); Instruction Set Reference, A-L (Order Number 253646);  
Instruction Set Reference, P-Q (Order Number 253647); Instruction Set Reference, R-Z (Order Number  
253648); Instruction Set Reference (Order Number 254929); System Programming Guide, Part 1 (Order  
Number 251858); System Programming Guide, Part 2 (Order Number 251859); System Programming  
Guide, Part 3 (Order Number 305011); System Programming Guide, Part 4 (Order Number 303021). Refer  
to all nine volumes when evaluating your design needs.



Intel Core 2 Architecture



Order Number: 253646-080US  
September 2011

Всего несколько томов по  
2000 страниц каждый!

# Как эмулятор выбирает нужную инструкцию?

Разработка прошивки

Язык Си

```
int x = data[4];
```

```
mov eax, [edx + 0x10]
```

Ассемблер

Прошивка

```
0x8b 0x42 0x10
```

Декодирование

```
private val e_opcode = x86InstructionTable(
  0x10, 0x10,
  { stream: x86OperandStream →
    val opcode = stream.readOpcode()
    ^lambda Pair(opcode[7..4], opcode[3..0]) },
  // 0
  // 1
  // 2
  // 3
  // 4
  // 5
  // 6
  // 7
  // 8
  // 9
  // A
  // B
  // C
  // D
  // E
  // F
)
```

	0	1	2	3	4	5	6	7	8
/*0*/	addDc,	addDc,	addDc,	addDc,	addDc,	addDc,	pushDc,	popDc,	orDc,
/*1*/	adcDc,	adcDc,	adcDc,	adcDc,	adcDc,	adcDc,	pushDc,	popDc,	sbbDc,
/*2*/	andDc,	andDc,	andDc,	andDc,	andDc,	andDc,	esOvr,	daaDc,	subDc,
/*3*/	xorDc,	xorDc,	xorDc,	xorDc,	xorDc,	xorDc,	ssOvr,	aaaDc,	cmpDc,
/*4*/	incDc,	incDc,	incDc,	incDc,	incDc,	incDc,	incDc,	incDc,	decDc,
/*5*/	pushDc,	pushDc,	pushDc,	pushDc,	pushDc,	pushDc,	pushDc,	pushDc,	popDc,
/*6*/	pushaDc,	popaDc,	null,	null,	fsOvr,	gsOvr,	operOvr,	addrOvr,	pushDc,
/*7*/	joDc,	jnoDc,	jbDc,	jnbDc,	jeDc,	jneDc,	jbeDc,	jaDc,	jsDc,
/*8*/	rm_dpnd,	rm_dpnd,	rm_dpnd,	rm_dpnd,	testDc,	testDc,	xchgDc,	xchgDc,	movDc,
/*9*/	nopDc,	xchgDc,	xchgDc,	xchgDc,	xchgDc,	xchgDc,	xchgDc,	xchgDc,	cwdeDc,
/*A*/	movDc,	movDc,	movDc,	movDc,	movsDc,	movsDc,	cmpsDc,	cmpsDc,	testDc,
/*B*/	movDc,	movDc,	movDc,	movDc,	movDc,	movDc,	movDc,	movDc,	movDc,
/*C*/	cell_ShtRl,	cell_ShtRl,	retDc,	retDc,	lesDc,	ldsDc,	movDc,	movDc,	enterDc,
/*D*/	cell_ShtRl,	cell_ShtRl,	cell_ShtRl,	cell_ShtRl,	null,	null,	null,	null,	cell_d8,
/*E*/	loopnzDc,	loopzDc,	loopDc,	jecxzDc,	inDc,	inDc,	outDc,	outDc,	callDc,
/*F*/	lock,	null,	repnz,	repz,	null,	null,	cell_f6f7,	cell_f6f7,	null,

# Как процессор декодирует инструкцию?



## INSTRUCTION SET REFERENCE

### MOV—Move

Opcode	Instruction	Description
88 <i>lr</i>	MOV <i>r/m8,r8</i>	Move <i>r8</i> to <i>r/m8</i>
89 <i>lr</i>	MOV <i>r/m16,r16</i>	Move <i>r16</i> to <i>r/m16</i>
89 <i>lr</i>	MOV <i>r/m32,r32</i>	Move <i>r32</i> to <i>r/m32</i>
8A <i>lr</i>	MOV <i>r8,r/m8</i>	Move <i>r/m8</i> to <i>r8</i>
8B <i>lr</i>	MOV <i>r16,r/m16</i>	Move <i>r/m16</i> to <i>r16</i>
8B <i>lr</i>	MOV <i>r32,r/m32</i>	Move <i>r/m32</i> to <i>r32</i>
8C <i>lr</i>	MOV <i>r/m16,Sreg**</i>	Move segment register to <i>r/m16</i>
8E <i>lr</i>	MOV <i>Sreg,r/m16**</i>	Move <i>r/m16</i> to segment register
A0	MOV AL, <i>moffs8*</i>	Move byte at ( <i>seg:offset</i> ) to AL
A1	MOV AX, <i>moffs16*</i>	Move word at ( <i>seg:offset</i> ) to AX
A1	MOV EAX, <i>moffs32*</i>	Move doubleword at ( <i>seg:offset</i> ) to EAX
A2	MOV <i>moffs8*</i> ,AL	Move AL to ( <i>seg:offset</i> )
A3	MOV <i>moffs16*</i> ,AX	Move AX to ( <i>seg:offset</i> )
A3	MOV <i>moffs32*</i> ,EAX	Move EAX to ( <i>seg:offset</i> )
B0+ <i>rb</i>	MOV <i>r8,imm8</i>	Move <i>imm8</i> to <i>r8</i>
B8+ <i>rw</i>	MOV <i>r16,imm16</i>	Move <i>imm16</i> to <i>r16</i>
B8+ <i>rd</i>	MOV <i>r32,imm32</i>	Move <i>imm32</i> to <i>r32</i>
C6 <i>l0</i>	MOV <i>r/m8,imm8</i>	Move <i>imm8</i> to <i>r/m8</i>
C7 <i>l0</i>	MOV <i>r/m16,imm16</i>	Move <i>imm16</i> to <i>r/m16</i>
C7 <i>l0</i>	MOV <i>r/m32,imm32</i>	Move <i>imm32</i> to <i>r/m32</i>

# Как эмулятор декодирует инструкцию?

```
class MovDC(core: x86Core) : ADecoder<AX86Instruction>(core) {  
    override fun decode(s: x86OperandStream, prefs: Prefixes): AX86Instruction {  
        val opcode = s.last  
        val rm = RMDC(s, prefs)  
        val ops = when (opcode) {  
            0x88 → arrayOf(rm.m8, rm.r8)  
            0x89 → arrayOf(rm.mpref, rm.rpref)  
            0x8A → arrayOf(rm.r8, rm.m8)  
            0x8B → arrayOf(rm.rpref, rm.mpref)  
            0x8C → arrayOf(rm.m16, rm.r(rtyp = Regtype.SSR))  
            0x8E → {  
                if (core.cpu.mode == x86CPU.Mode.R32)  
                    prefs.operandOverride = true;  
                arrayOf(rm.r(rtyp = Regtype.SSR), rm.mpref)  
            }  
  
            0xA0 → arrayOf(al, s.mem8(prefs))  
            0xA1 → arrayOf(x86Register.gpr(prefs.opsize, x86GPR.EAX.id), s.mem(prefs))  
            0xA2 → arrayOf(s.mem8(prefs), al)  
            0xA3 → arrayOf(s.mem(prefs), x86Register.gpr(prefs.opsize, x86GPR.EAX.id))  
  
            0xB0, 0xB1, 0xB2, 0xB3, 0xB4, 0xB5, 0xB6, 0xB7 →  
                arrayOf(x86Register.gpr8(id: opcode % 0xB0), s.imm8)  
  
            0xB8, 0xB9, 0xBA, 0xBB, 0xBC, 0xBD, 0xBE, 0xBF →  
                arrayOf(x86Register.gpr(prefs.opsize, id: opcode % 0xB8), s.imm(prefs))  
  
            0xC6 → arrayOf(rm.m8, s.imm8)  
            0xC7 → arrayOf(rm.mpref, s.imm(prefs))  
            else → throw GeneralException("Incorrect opcode in decoder")  
        }  
        return Mov(core, s.data, prefs, *ops)  
    }  
}
```

## Description

Move *r8* to *r/m8*  
Move *r16* to *r/m16*  
Move *r32* to *r/m32*  
Move *r/m8* to *r8*  
Move *r/m16* to *r16*  
Move *r/m32* to *r32*  
Move segment register to *r/m16*  
Move *r/m16* to segment register  
Move byte at (*seg.offset*) to AL  
Move word at (*seg.offset*) to AX  
Move doubleword at (*seg.offset*) to EAX  
Move AL to (*seg.offset*)  
Move AX to (*seg.offset*)  
Move EAX to (*seg.offset*)  
Move *imm8* to *r8*  
Move *imm16* to *r16*  
Move *imm32* to *r32*  
Move *imm8* to *r/m8*  
Move *imm16* to *r/m16*  
Move *imm32* to *r/m32*

Intel ISet vol. x3

SMARTRHINO  
2019

# Как эмулятор выполняет инструкции?

## Описание выполнения инструкции "jge"

```
/**
 * Created by davydov_vn on 28.09.16.
 */
class Jge(core: x86Core, opcode: ByteArray, prefs: Prefixes, operand: AOperand<x86Core>):
    AX86Instruction(core, Type.COND_JUMP, opcode, prefs, operand) {
        override val mnem = "jge"

        override fun execute() {
            if(core.cpu.flags.sf == core.cpu.flags.of)
                core.cpu.regs.eip += op1.ssex(core)
        }
    }
}
```

## Описание выполнения инструкции "mov"

```
/**
 * Created by davydov_vn on 22.09.16.
 */
class Mov(core: x86Core, opcode: ByteArray, prefs: Prefixes, vararg operands: AOperand<x86Core>):
    AX86Instruction(core, Type.VOID, opcode, prefs, *operands) {
        override val mnem = "mov"

        override fun execute() {
            val a2 = op2.value(core)
            op1.value(core, a2)
        }
    }
}
```



# Проблемы создания эмулятора

- Разные процессорные архитектуры: *x86, mips, arm, ppc...*
- Различные режимы работы процессора: *real-mode, protected-mode, arm, thumb...*
- Особенность доступа к памяти: *mmu*
- Зависимость от периферии: *uart, spi, i2c, ethernet...*

...но архитектур достаточно ограниченное число и это не частая работа...



...много процессорных ядер уже реализовано...

...конфигурация под конкретное устройство...

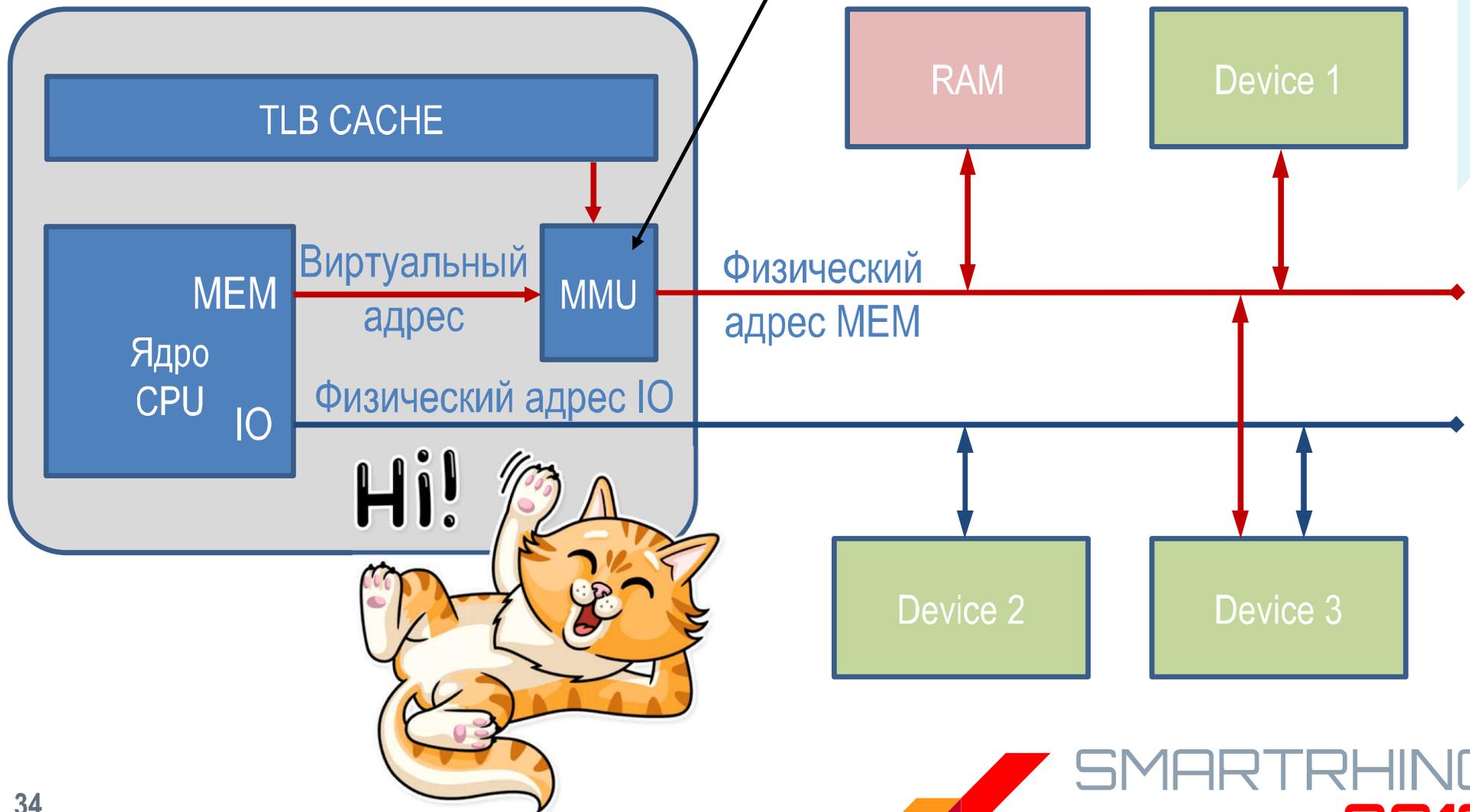
...удобство реализации периферии...

# Что за трансляция? И чем это грозит?

Тут прячется демон



Контроллер/процессор



# Пример «особой» трансляции ARM

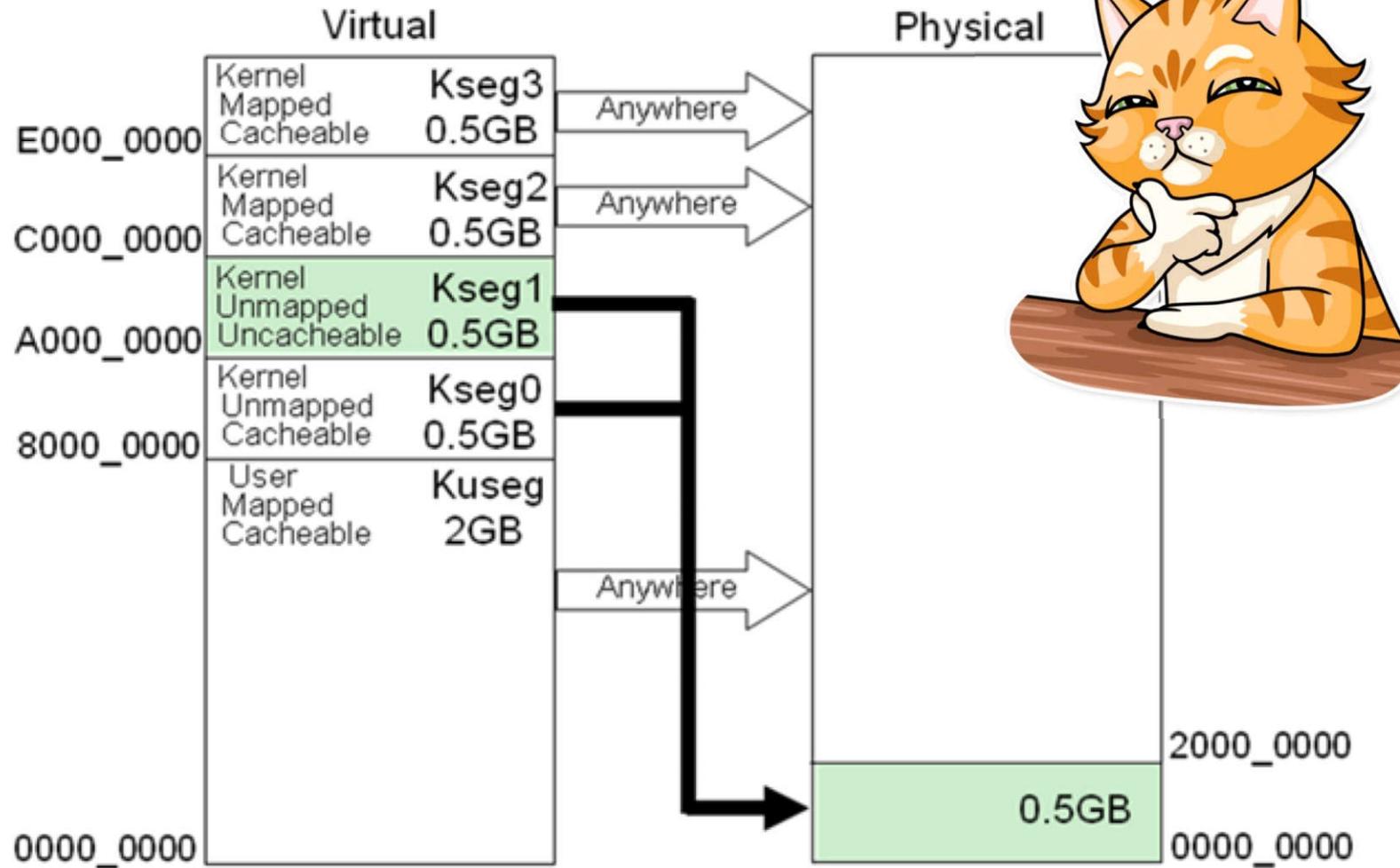
Ничего особого нету!

0x2000'0000 значит, что на физической шине будет 0x2000'0000

Not Today



# Пример «особой» трансляции MIPS



```
*((int*)0x80001024) = 0xDEAD;
*((int*)0xA0001024) = 0xBEEF;
```



```
*((int*)0x00001024) = 0xDEAD;
*((int*)0x00001024) = 0xBEEF;
```



# Трансляция адресов MIPS в эмуляторе

```
for (i in 0 until TLB.size) {
  val invMask = TLB[i].Mask.inv()
  val vpn = TLB[i].VPN2 and invMask
  val eaPage = ea[31..13].toInt() and invMask
  val isGlobal = TLB[i].G == 1
  val isIdEquals = TLB[i].ASID == entryHiASID
  if ((vpn == eaPage) && (isGlobal || isIdEquals)) {

    val evenOddBit = when (TLB[i].Mask) {
      0b0000000000000000 → 12
      0b0000000000000011 → 14
      0b0000000000001111 → 16
      0b0000000000111111 → 18
      0b0000000011111111 → 20
      0b0000001111111111 → 22
      0b0000111111111111 → 24
      0b0011111111111111 → 26
      0b1111111111111111 → 28
      else → -1
    }

    if (ea[evenOddBit] == 0L) {
      pfn = TLB[i].PFN0
      v = TLB[i].V0
      d = TLB[i].D0
    } else {
      pfn = TLB[i].PFN1
      v = TLB[i].V1
      d = TLB[i].D1
    }

    if (v == 0)
      throw MipsHardwareException.TLBInvalid(LorS, core.pc, ea)
    if (d == 0 && LorS == AccessAction.STORE)
      throw MipsHardwareException.TLBModified(core.pc, ea)

    if (found) {
      throw GeneralException("[${core.pc.hex8}] Double MMU TLB match for")
    }

    val msb = mips.PABITS - 1 - 12
    val lsb = evenOddBit - 12
    val page = pfn[msb..lsb].asULong
    val offset = ea[evenOddBit - 1..0]

    pAddr = (page shl evenOddBit) or offset
  }
}
```



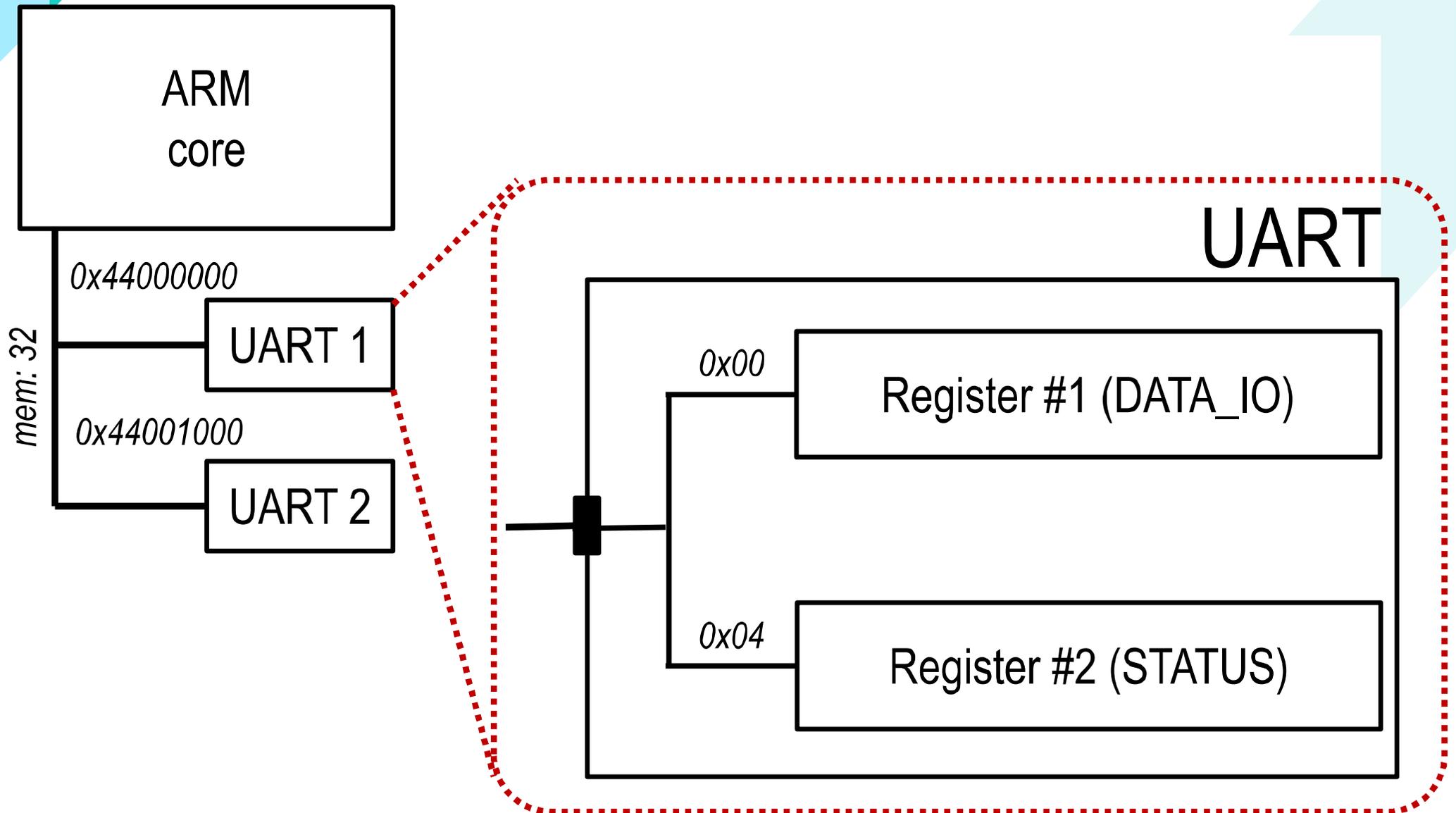
```
found ← 0
for i in 0..TLBEntries-1
  if ((TLB[i].VPN2 and not (TLB[i].Mask)) = (va31..13 and not (TLB[i].Mask))) and
    (TLB[i].G or (TLB[i].ASID = EntryHiASID)) then
    # EvenOddBit selects between even and odd halves of the TLB as a function of
    # the page size in the matching TLB entry. Not all page sizes need
    # be implemented on all processors, so the case below uses an 'x' to
    # denote don't-care cases. The actual implementation would select
    # the even-odd bit in a way that is compatible with the page sizes
    # actually implemented.
    case TLB[i].Mask
      0b0000 0000 0000 0000: EvenOddBit ← 12 /* 4KB page */
      0b0000 0000 0000 0011: EvenOddBit ← 14 /* 16KB page */
      0b0000 0000 0000 11xx: EvenOddBit ← 16 /* 64KB page */
      0b0000 0000 0011 xxxx: EvenOddBit ← 18 /* 256KB page */
      0b0000 0000 11xx xxxx: EvenOddBit ← 20 /* 1MB page */
      0b0000 0011 xxxx xxxx: EvenOddBit ← 22 /* 4MB page */
      0b0000 11xx xxxx xxxx: EvenOddBit ← 24 /* 16MB page */
      0b0011 xxxx xxxx xxxx: EvenOddBit ← 26 /* 64MB page */
      0b11xx xxxx xxxx xxxx: EvenOddBit ← 28 /* 256MB page */
      otherwise: UNDEFINED
    endcase
    if vaEvenOddBit = 0 then
      pfn ← TLB[i].PFN0
      v ← TLB[i].V0
      c ← TLB[i].C0
      d ← TLB[i].D0
      if (Config3RXI or Config3SM) then
        ri ← TLB[i].RI0
        xi ← TLB[i].XI0
      endif
    else
      pfn ← TLB[i].PFN1
      v ← TLB[i].V1
      c ← TLB[i].C1
      d ← TLB[i].D1
      if (Config3RXI or Config3SM) then
        ri ← TLB[i].RI1
        xi ← TLB[i].XI1
      endif
    endif
    if v = 0 then
      SignalException(TLBInvalid, reftype)
    endif
  endif
endfor
```

Псевдокод из MIPS vol.3

SMARTRHINO  
2019

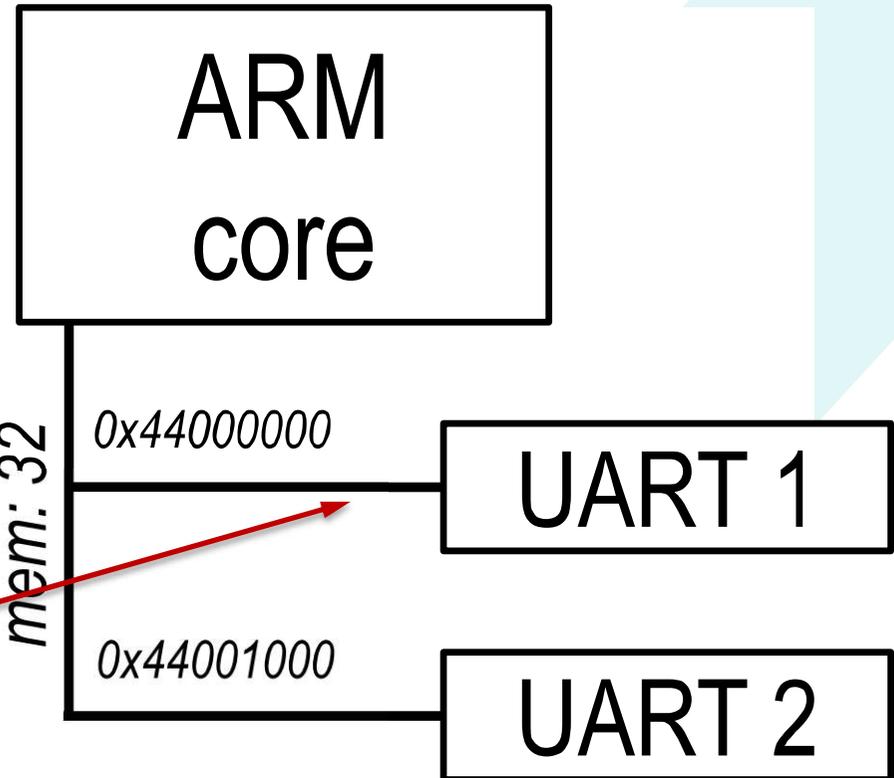


# Посмотрим ближе на UART



# Реализация периферии

```
08002B98 MOVS R0, huart
08002B9A MOVS R7, #1
08002B9C ADDS R0, #0x58 ; 'X'
08002B9E STRB R7, [R6]
08002BA0 MOVS R4, #0
08002BA2 STR pData, [huart,#0x54]
08002BA4 STRH R3, [R0]
08002BA6 MOVS R0, #0x22 ; ""
08002BA8 STR R4, [huart,#0x6C]
08002BAA STRB R0, [R2]
08002BAC LDR R0, [huart,#0x64]; hdma
08002BAE LDR R2, =(UART_DMAReceiveCplt+1)
08002BB0 STR R4, [R0,#0x34]
08002BB2 STR R2, [R0,#0x28]
08002BB4 LDR R2, =(UART_DMARxHalfCplt+1)
08002BB6 STR R2, [R0,#0x2C]
08002BB8 LDR R2, =(UART_DMAError+1)
08002BBA STR R2, [R0,#0x30]
08002BBC LDR R2, [huart]
08002BBE ADDS R2, #0x24 ; '$'
08002BC0 MOV R12, R2
08002BC2 MOVS R2, pData; DstAddress
08002BC4 MOV pData, R12; SrcAddress
08002BC6 pData = R2; uint8_t *
08002BC6 BL HAL_DMA_Start_IT
08002BCA MOVS R2, #0x80
08002BCC LDR R3, [huart]
08002BCE LSL R2, R2, #1
08002BD0 LDR R1, [R3]
08002BD2 STRB R4, [R6]
08002BD4 ORRS R2, R1
08002BD6 STR R2, [R3]
08002BD8 LDR R2, [R3,#8]
08002BDA MOVS R0, R4
08002BDC ORRS R7, R2
08002BDE MOVS R2, #0x40 ; '@'
08002BE0 STR R7, [R3,#8]
08002BE2 LDR R1, [R3,#8]
08002BE4 ORRS R2, R1
08002BE6 STR R2, [R3,#8]
```



Инструкция LDR ARM  
«загрузи значение по адресу в регистре R6»

# Реализация периферии

```
class Uart(parent: Module, name: String): Module(parent, name) {  
    inner class Ports : ModulePorts( module: this ) {  
        val mem = Slave( name: "mem", BUS30 )  
    }  
    override val ports = Ports()  
  
    private val uartTx = File( pathname: "temp/uart.log" ).outputStream()  
  
    val DATA_IO = object : Register(ports.mem, address: 0x00, BYTE,  
        name: "UART_IO" ) {  
        override fun read(ea: Long, ss: Int, size: Int): Long = 0  
        override fun write(ea: Long, ss: Int, size: Int, value: Long) {  
            uartTx.write(value.toInt())  
            uartTx.flush()  
        }  
    }  
  
    val STATUS = object : Register(ports.mem, address: 0x04, BYTE,  
        name: "STATUS", writable = false) {  
        override fun read(ea: Long, ss: Int, size: Int): Long = 0  
    }  
}
```

Определение портов периферии

Начальная инициализация

Определение первого регистра

Определение второго регистра

# Как это работает всё вместе?

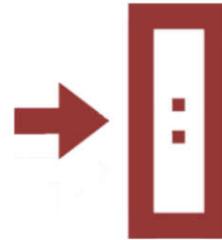
Конфигурация устройства  
(JSON)



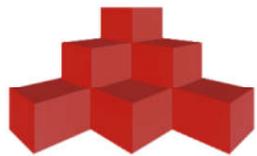
Это эмулятор  
[kory.cat](http://kory.cat)



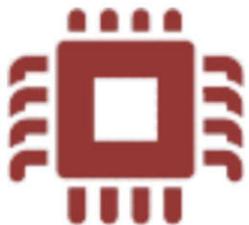
GDB RSP



eclipse



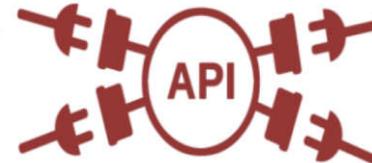
Набор модулей



Данные ПЗУ устройства



Снапшот (снимок состояния эмулятора)



# Как в итоге это происходит?

The screenshot displays the Remote GDB debugger interface. The main window shows a disassembled function with assembly code and control flow graph (CFG) nodes. The CFG nodes are labeled with addresses and instructions, such as `loc_8000B0` and `loc_8000B2`. The assembly code includes instructions like `HAL_RCC_ClockConfig` and `HAL_RCC_GetSystemClockFreq`. The right panel shows the General registers window, displaying the values of registers R0 through R15. The bottom status bar indicates the debugger is attached to a process (pid=4294967294) and is synchronized with the PC and Hex View-1.

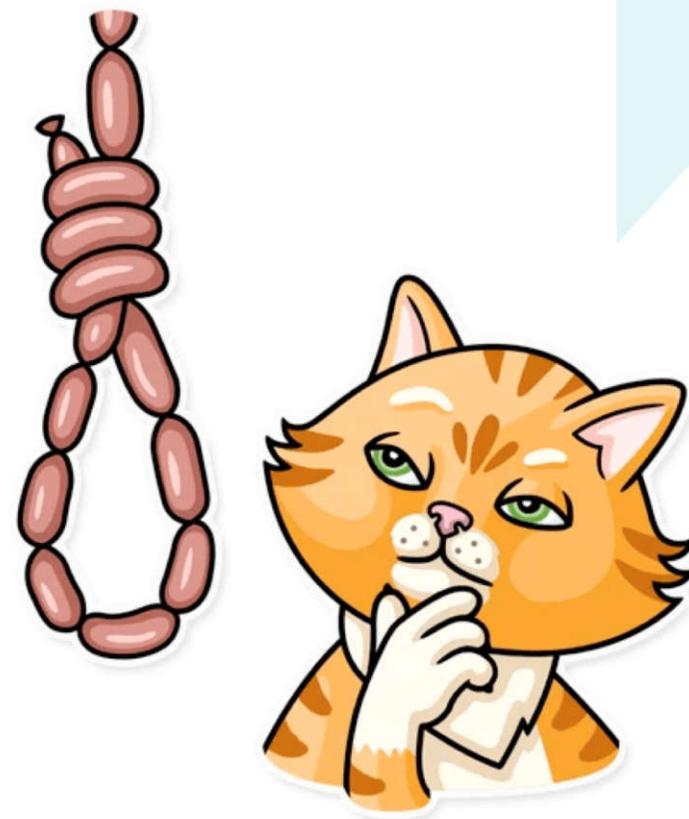
## Взаимодействие по GDB RSP

```
15:37:53 INFO [Module.initializeAndRes]: Setup debugger to dbg[ARMDebugger] for top[STM32F042]
15:37:53 WARN [Module.initializeAndRes]: Tracer wasn't found in top[STM32F042]...
15:37:53 INFO [Module.initializeAndRes]: Module top[STM32F042] is successfully initialized and reset as a top cell!
15:37:53 INFO [Kopycat.open]: Starting virtualization of board top[STM32F042] with arm[ARMCore]
15:37:53 INFO [Kopycat.open]: State:
15:37:53 INFO [ANetworkThread.run]: GDB_SERVER thread started on GDB_SERVER [127.0.1.1:23946]
15:37:53 INFO [ANetworkThread.run]: GDB_SERVER waited for clients on 23946...
15:37:53 WARN [KopycatStarter.main]: Loading Python CLI failed using embedded one...
15:37:53 WARN [AConsole.run]: Python initialization error, terminating console...
15:37:53 INFO [REST.<init>]: Starting Javalin RESTful API on port 6666
Argparse > 15:38:13 INFO [ANetworkThread.run]: Client Socket[addr=192.168.76.93,port=44258,localport=23946] connected to GDB_SERVER
[08001D7C] 00004813 LDR r0, 0x4C
```

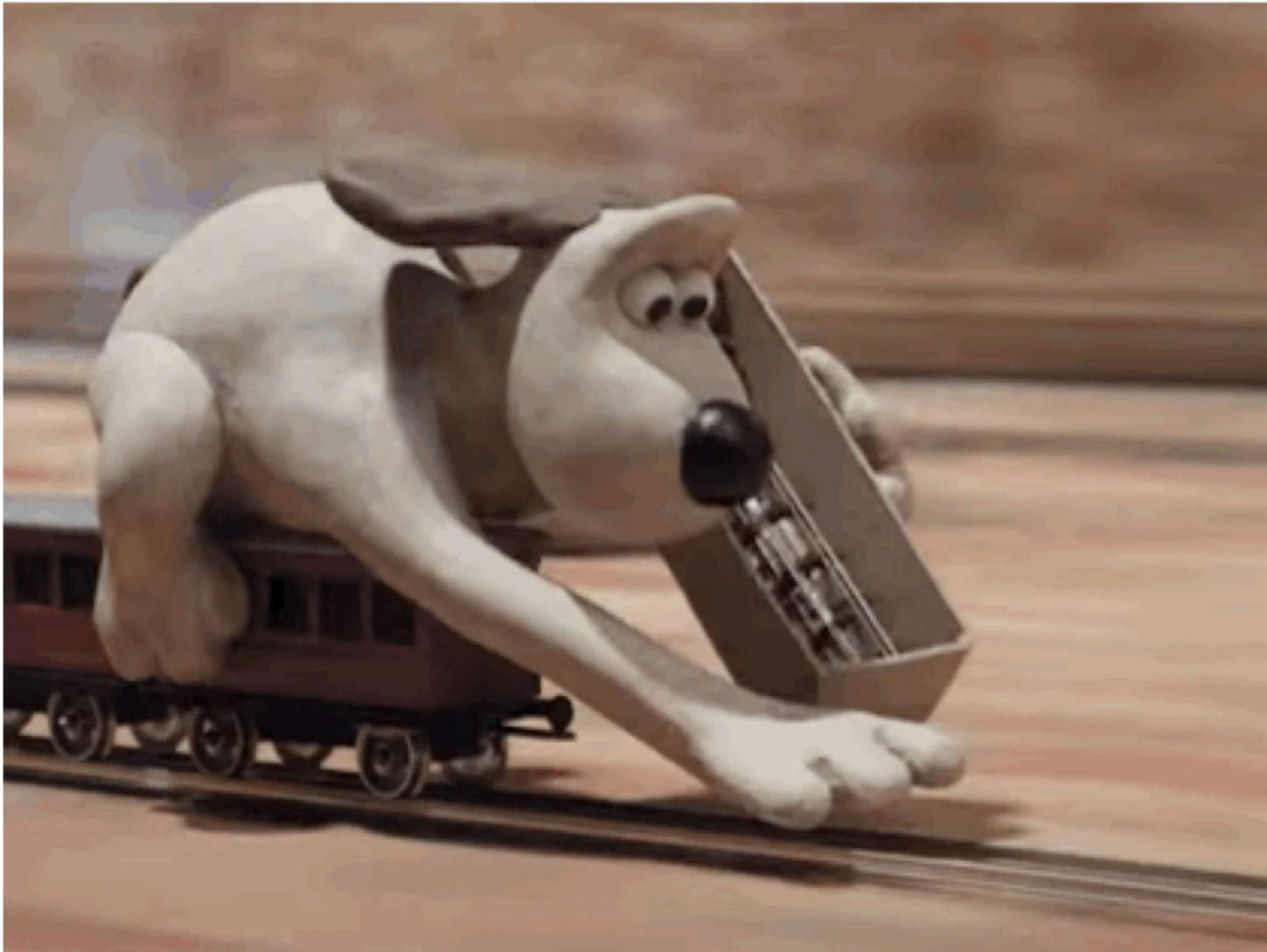
# Эмулируем не только свое...

...или не все устройства и производители одинаково полезны

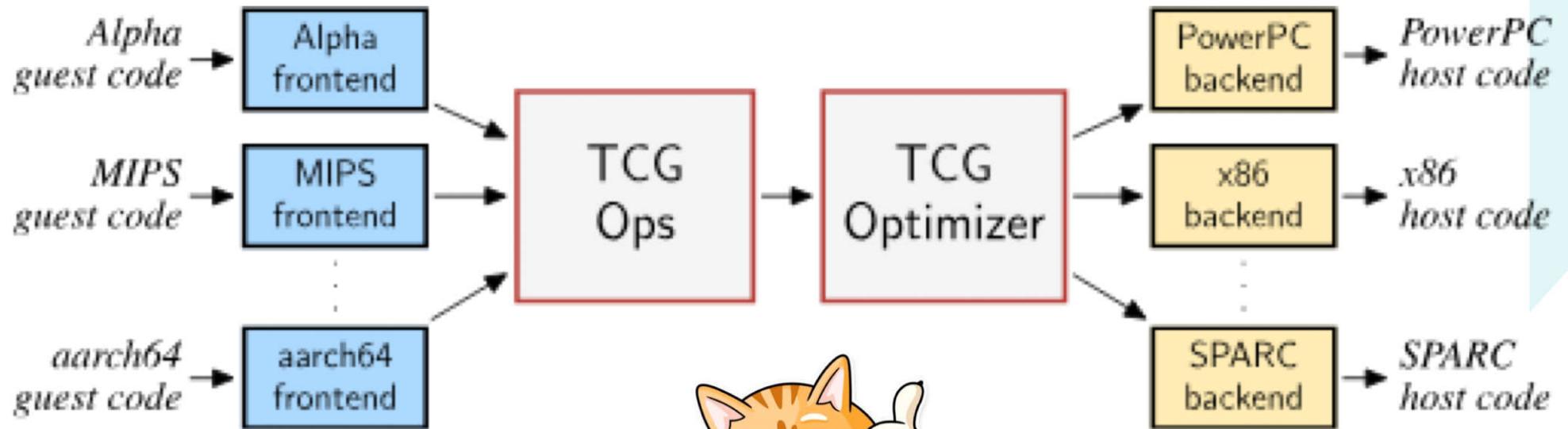
- Анализ сжатых, зашифрованных данных, размещение данных в адресном пространстве
- Анализ прошивок на наличие в них чего-то «лишнего»
- Анализ встроенного вредоносного ПО
- Полезность устройства по шкале от 1 до 10



# За кадром QEMU: TCG и JIT-компиляция



# QEMU: Tiny Code Generator



Инструкции гостя превращаются в набор микро-инструкций эмулятора, после чего для них проводится оптимизация

## Итого: из чего всё состоит

1. Набор инструкций ядра
2. Копроцессор и систему прерываний
3. FPU
4. Транслятор адресов MMU
5. Периферийные модули (RAM, ROM, UART, SPI, и т.д.)
6. Взаимодействие между периферийными модулями
7. Интерфейс внешнего отладчика
8. Кнопка запустить...



# Заключение

Можно избежать значительной части ошибок в программном обеспечении сложных технических систем, если есть возможность поработать с этой системой до её появления на свет...



**EMULATION DONE... SEGMENTATION FAULT**